

## **Multi-Objective Scheduling Problem to Reduce Job Tardiness and to Balance Workload for Workers Possessing Different Skill Levels**

Masahiro Arakawa<sup>1\*</sup>, Masahiko Fuyuki<sup>1</sup>, and Ichiro Inoue<sup>2</sup>

<sup>1</sup> Faculty of Engineering, Kansai University, Suita, Osaka 564-8680, Japan

{arakawa,fuyuki}@iecs.kansai-u.ac.jp

<sup>2</sup> Faculty of Management, Kyoto Sangyo University, Kita-Ku Kamigamo, 603-8555, Japan

inoue@cc.kyoto-su.ac.jp

**Abstract.** In many make-to-order production processes, manufacturing techniques based on worker's proficiency are introduced to flexibly produce various products. Worker's proficiency influences production lead time and product quality. It is most efficient to assign workers possessing higher skill level as many operations as possible in order to meet production deadlines. However, this assignment process creates an imbalanced workload among workers with different skill levels. In this study, we examine a scheduling problem and present a method aimed to reduce job tardiness and to balance workload among workers. To resolve this problem, job assignments to machines and worker assignments to operations guide in this scheduling process. We propose a hybrid type of scheduling method based on a genetic algorithm to minimize multi-objective functions: average job tardiness and peak to peak value of operation time among workers. To efficiently search for the significant combinations of these assignments, the proposed method is introduced in which the control of worker assignments belongs to control of job assignments. In addition, a hybrid type of local search procedure is adopted to control worker assignments in it. This paper examines characteristics of the proposed hybrid Genetic Algorithms on several simple job shop models. Numerical results demonstrate that the proposed method can generate Pareto solutions in a short time.

**Keywords:** Scheduling, Multi-objective functions, Genetic algorithm, Job tardiness, Local search, Workload balancing.

---

\* Corresponding Author. Tel.: +81-6-6368-0821, Email: arakawa@iecs.kansai-u.ac.jp.

## **1 Introduction**

The diversification of customer needs has prompted manufacturers to implement make-to-order production processes that can handle multiple-items and small lot-size. In many factories under make-to-order production system, production relies on the workers' skill levels. Under such circumstances, skilled workers are likely to be assigned more work in order to meet the production deadline. However, it is clear that this approach causes an imbalance of workloads between workers with higher skills, who will be assigned more operations, and workers with lower skills. This characteristic indicates a tradeoff between a balance of workloads among workers and reduction of job tardiness in a scheduling problem including simultaneous control of job and worker assignments.

In this study, we propose a scheduling method aimed to reduce job tardiness and to balance workload among workers by simultaneously assigning jobs to machines and workers to operations in scheduling process. Workers selected for assignment are independent on the order of operations in each job, so that the number of combined variables for worker assignments takes a large number. It causes that combination of job assignment and worker assignments takes a large number to generate feasible schedules. Therefore, we must propose a procedure to efficiently search for the best solutions and incorporate it into the scheduling method.

Numbers of past studies have examined the scheduling problem focusing on job and worker assignments. In these studies, job and assignments were individually controlled. For example, a genetic algorithm was used for both assignment procedures [1], whereas it was only used for the job assignments procedure and the heuristic process was used for worker assignment procedure [2]. However, these studies treated a single objective function and they did not discuss the imbalance in worker workloads. Furthermore, they focused on generating feasible schedules incorporating the constraints of workers assigned to specific operations.

In our study, a hybrid scheduling method based on a genetic algorithm is developed to resolve the multi-objective functions of job tardiness and imbalance of workloads among workers. In our method, worker assignments are based on the priority of jobs assigned to each available machine. We used one of two methods for worker assignments in the hybrid Genetic Algorithm: a genetic algorithm and a hybrid local search. The hybrid local search method involves two types of worker assignment procedures which are aimed to minimize average tardiness and the peak to peak value of operation time among workers.

In this paper, properties of the hybrid Genetic Algorithm are explained and the method's performance is investigated on simple job shop models. The numerical results show that the algorithm generates high quality Pareto solutions and that the hybrid local search procedure can improve solutions and reduce computation time.

## 2 Representation of Worker Skills

The worker skills have influence on operation properties both qualitatively and quantitatively: the quality of operations influences the quality of products and the quantity of operations operation time. Since operation time influences scheduling problems, in this study we focus on operation time influenced by workers' skill levels. The operation time of an operation carried out by workers possessing different skill levels is defined by the following equation:

$$P_{ijk} = \eta_{kij} \bar{p}_{ij}. \quad (1)$$

$P_{ijk}$  denotes the operation time of  $j$ -th operation of job  $i$  when worker  $k$  are assigned to the operation. Here,  $i, j, \bar{p}_{ij}$ , and  $\eta_{kij}$  denote job number, operation number, standard operation time of  $j$ -th operation of job  $i$ , and efficiency of worker  $k$  to process  $j$ -th operation of job  $i$ , respectively. Skill level of worker  $k$  to process  $j$ -th operation of job  $i$  corresponds to the efficiency,  $\eta_{kij}$ . When a worker is not available to process specific operations, efficiency of the worker for the operations is defined as infinity.

## 3 Objective Functions and Constraints

Our test experiment result shows that a scheduling problem, which includes simultaneous control of job assignment and worker assignment, involves a tradeoff between reduction of job tardiness and a balance of workers' workload. We introduce two objective functions related to this tradeoff: average tardiness and peak-to-peak value of operation time among all workers. In this paper, average tardiness is symbolized as  $L$ . The peak-to-peak value of operation time among the workers is referred to as "peak-to-peak value of operation time" and is symbolized as  $\delta P$ , hereafter. These criteria are defined as the following:

$$L = \frac{1}{n} \sum_{i=1}^n \max(L_i, 0), \quad (2)$$

$$L_i = C_i - d_i, \quad (3)$$

$$\delta P = P_{\max} - P_{\min}, \quad (4)$$

$$P_{\max} = \max\{P_k \mid k = 1, \dots, K\}, \quad (5)$$

$$P_{\min} = \min\{P_k \mid k = 1, \dots, K\}, \quad (6)$$

$$P_k = \sum_{i=1}^n \sum_{j=1}^{m_i} \bar{p}_{ij} \eta_{kij} \delta_{kij}. \quad (7)$$

Here,  $C_i$  and  $d_i$  denote completion time of job  $i$  and due-date of job  $i$ , respectively.  $n$  and  $m_i$  denote the number of jobs and the number of operations of job  $i$ .  $K$  denotes the number of workers. When worker  $k$  is assigned to  $j$ -th operation of job  $i$ ,  $\delta_{kij}$  takes 1. Otherwise,  $\delta_{kij}$  takes 0. The constraints of the problem can be modeled as follows:

$$\sum_k \sum_{\ell} a_{ijk\ell} = 1, \tag{8}$$

$$t_{ij+1} - t_{ij} \geq \sum_k \sum_{\ell} (a_{ijk\ell} p_{ijk}) \quad \forall i, j, \tag{9}$$

$$t_{ij} \geq r_i \quad \forall i, j, \tag{10}$$

$$t_{i'j'} - t_{ij} \geq \sum_{\ell} (a_{ijk\ell} p_{ijk}) \vee t_{ij} - t_{i'j'} \geq \sum_{\ell} (a_{i'j'k\ell} p_{i'j'k}) \tag{11}$$

$$\forall i, i', j, j', k, (i' \neq i) \vee (i = i' \wedge j \neq j'),$$

$$t_{i'j'} - t_{ij} \geq \sum_k (a_{ijk\ell} p_{ijk}) \vee t_{ij} - t_{i'j'} \geq \sum_k (a_{i'j'k\ell} p_{i'j'k}) \quad \forall i, i', j, j', \ell, i' \neq i. \tag{12}$$

Here,  $t_{ij}$  denotes start time of  $j$ -th operation of job  $i$ .  $a_{ijk\ell}$  takes 1 if worker  $k$  is assigned to  $j$ -th operation of job  $i$  at machine  $\ell$ , otherwise, it takes zero.  $r_i$  denotes release time of job  $i$ . Equation (9) denotes the constraint for assigning successive operations of job  $i$  to workers and machines without overlap of these operations. Equation (11) denotes the constraint for assigning less than single operation to worker  $k$  at any time, and equation (12) the constraint for assigning single or no operation to machine  $\ell$  at any time.

## 4 Hybrid Genetic Algorithm

### 4.1 Representation of Chromosomes

A worker assigned to an operation does not depend on operations, to which the worker is previously assigned, in this paper's scheduling problem. When the number of jobs, machines, and workers take  $n$ ,  $m$ , and  $s$ , respectively, the number of all combinations with these variables which are needed to evaluate schedules is estimated to be  $(n!)^m \cdot s^{m \cdot n}$ . It denotes that the number of the variable combinations related to worker assignment increases by the exponent  $mn$ . Therefore, we need to introduce a procedure to effectively search for combinations of job and worker assignments to generate Pareto solutions.

In order to avoid the explosion of the number of combination of these assignments to search for Pareto solutions, we first separate the controls for these assignments into two individual controls of job and worker assignments, and then incorporate the individual controls into a hybrid procedure based on a genetic algorithm. Hereafter, the hybrid procedure is denoted as ‘Hybrid Genetic Algorithm’ or ‘HGA’ method. Figure 1 shows coding space given by combinations of two types of chromosomes in different methods. A pair of two different chromosomes can generate a single schedule. The left figure shows a procedure to search for pairs of the chromosomes modifying elements in both chromosomes in a genetic algorithm process. It is a popular procedure and is called as ‘Simple GA’ method or ‘SGA’ method, hereafter. The right figure shows the procedure of the HGA method. In the HGA method, job and worker assignments are controlled by each individual chromosome. Job assignments are mainly controlled by the genetic algorithm framework. Worker assignments to operations are controlled on the condition of a single chromosome for job assignment.

Figure 2 shows a schematic diagram of the HGA method. ‘ChromA’ and ‘ChromB’ denote a chromosome for job assignment control and a chromosome for worker assignment control, respectively. The plural chromosomes of ChromB belong to a single chromosome of ChromA. A pair of a single chromosome of ChromA and a single chromosome of ChromB yields a single schedule. Fig.2 also shows an evolutionary process of ChromA in the HGA method from  $n$ -th generation to  $(n+1)$ -th generation. From the population of ChromA in the  $n$ -th generation, a specific number of chromosomes are selected for the  $(n+1)$ -th generation. Then, new chromosomes of ChromA are generated by using crossover and mutation with the selected chromosomes. Finally, we yield a new population of ChromA by combining the selected chromosomes and the generated chromosomes for the  $(n+1)$ -th generation. The new chromosomes of ChromB are generated in each chromosome of ChromA and new schedules are generated from the pairs of ChromA and ChromB.

Figures 3(a) and (b) represent ChromA and ChromB. Operations-based representation is adopted for ChromA shown in Fig.3(a). The job numbers, which are identical with the number for the job operations, are set in a one-dimensional array. In addition, the order of job numbers indicates the priority order of jobs to be assigned to each machine. Job numbers on the left side means that the jobs are assigned prior to jobs symbolized by job numbers presented on the right side of Fig3(a). As shown in Fig.1 and 2, the genetic algorithm is adopted to evolutionary procedure for ChromA.

On the other hand, a two-dimensional array is used for representation of ChromB. The array’s rows indicate jobs and the columns indicate each job’s operations. The worker number assigned to each job operation is represented in each array element. Fig.3(b) shows that worker 3 is assigned to the 2nd operation of job 2. To operate genes in ChromB chromosomes in the evolutionary process, one of two types of procedures is adopted; a genetic algorithm or a hybrid local search method. The hybrid local search method is proposed in this study and explained in section 4.3.

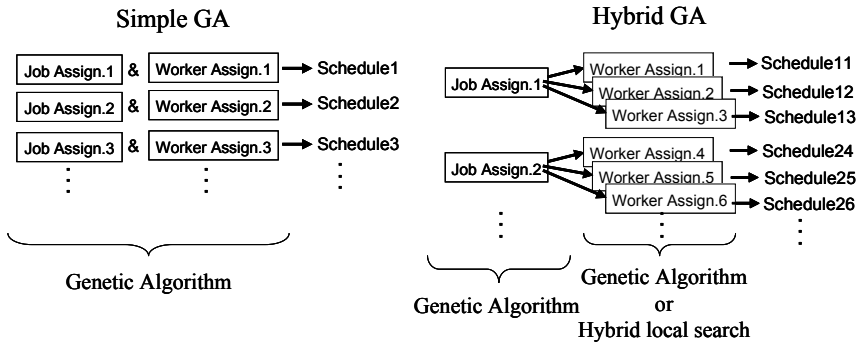


Fig. 1. Cording Space of job assignment and worker assignment in two types of Genetic Algorithms

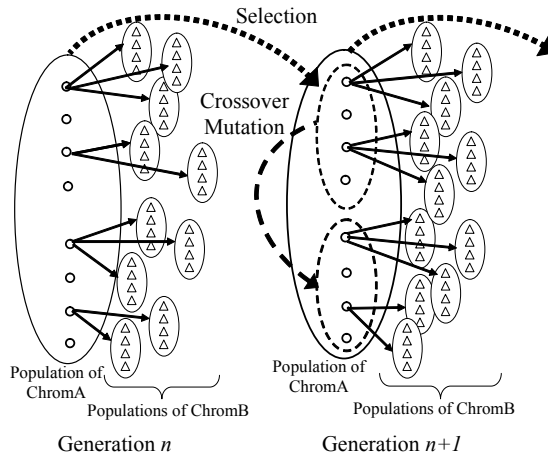


Fig. 2. Schematic diagram of evolutionary process of HGA method

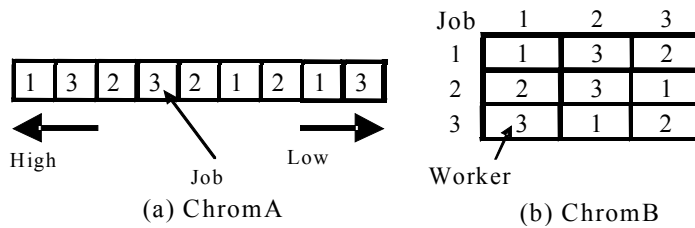


Fig. 3. Example of chromosomes (3 jobs, 3 machines, 3 workers)

## 4.2 Properties of a Hybrid Genetic Algorithm

Figure 4 shows a flow chart of the HGA method and illustrates the evolutionary procedure for populations of ChromA. “*Proc* $\times$ *n*” denotes *n* iterations of procedure ‘*Proc*’ and ‘*Proc*’ includes generation and evaluation of schedules. In this study, either the genetic algorithm or the hybrid local search method is adopted to procedure ‘*Proc*’. To select chromosomes of ChromA for a successive generation, we use Pareto preservation and parallel selection. The detail of the HGA method is shown below.

### The HGA method

- (1) Maximum generations and the number of chromosomes in the population of ChromA take ‘MaxGene’ and ‘popA’, respectively.
- (2) Generation is initialized as gene=0. Here, ‘gene’ denotes the current generation.
- (3) The initial population of ChromA chromosomes is generated and is named set ‘z0’.
- (4) Procedure ‘*Proc*’ is executed for all chromosomes in the population.
- (5) All chromosomes of ChromB in all chromosomes of ChromA in set z0 are ranked.
- (6) Generation is updated: gene = gene+1.
- (7) A new population of ChromA chromosomes is prepared as set ‘z1’ and it takes  $z1 = \phi$ . Chromosomes of popA are selected from set z0 and are added to set z1 in order of the following procedure.
  - (7-1) Selection using rank for sharing
  - (7-2) Pareto preservation
  - (7-3) Parallel selection
- (8) All chromosomes of ChromB in all chromosomes of ChromA in set z1 are ranked.
- (9) New chromosomes of ChromA are generated through to the following process.
  - (9-1) The number of chromosomes is initialized as pop=0, and all chromosomes in set z0 are removed:  $z0 = \phi$ . Here, ‘pop’ denotes the number of chromosomes in a set.
  - (9-2) The chromosomes of ChromA are selected as parent chromosomes from set z1 using the roulette wheel selection process.
  - (9-3) A crossover is executed to generate offspring chromosomes.
  - (9-4) A mutation is executed with using uniform random number.
  - (9-5) The offspring are added to set z0.
  - (9-6) The number of chromosomes in set z1 is updated.
  - (9-7) When  $pop < popA$ , go back to (9-2). When  $pop > popA$ , chromosomes in set z1 is deleted to make  $pop = popA$ , and go to (10). Otherwise, go to (10).
- (10) Procedure ‘*Proc*’ is executed for all chromosomes of ChromA in set z0.
- (11) All chromosomes in set z1 are transferred into set z0.
- (12) All chromosomes of ChromB in all chromosomes of ChromA in set z0 are ranked.
- (13) When gene < MaxGene, go to (6). Otherwise, the procedure is completed.

The Fonseca procedure [3] is used to calculate ranks of chromosomes. In the roulette wheel selection in step(9-2), the ChromA chromosomes are selected by using ranks of all ChromB chro-

mosomes in all chromosomes of ChromA in the population. Probability defined by equation (13) is used for the roulette wheel selection process.

$$\Pr\{r_\ell\} = \frac{S_r - rank_\ell}{S_r(n-1)} \tag{13}$$

$$S_r = \sum_{\ell=1}^v rank_\ell \tag{14}$$

Here,  $v$  and  $rank_\ell$  denote the total number of chromosomes of ChromB, and rank of  $\ell$ -th chromosome of ChromB, respectively. The following treatments are introduced in the HGA method:

(Treatment1) The chromosomes of ChromB, which yield Pareto solutions in each chromosome of ChromA, are reserved in the successive generation,

(Treatment2) The Giffler-Thomson method [4] is adopted for job assignment to machines. The method is simplified as ‘GT method’, hereafter, and

(Treatment3) The elements in each chromosome of ChromA are modified to correspond to the order of job selection in a schedule.

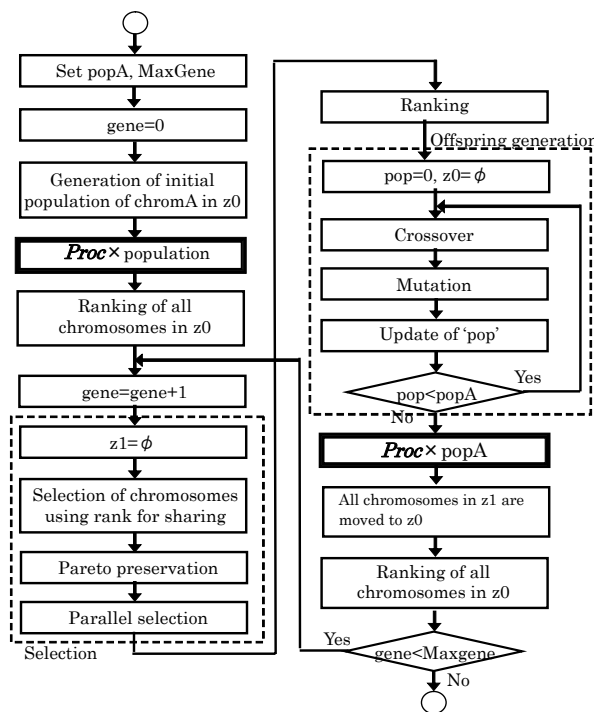


Fig. 4. Flowchart of HGA method

Treatment1 aims at reducing the number of chromosomes investigated for ranking. In the ranking process, all ChromB chromosomes in all ChromA chromosomes need to be compared with

each other. Because ChromB chromosomes generating Pareto solutions in each ChromA chromosome are related to ChromB chromosomes generating Pareto solutions in all ChromA chromosomes by equation (15), ChromB chromosomes which do not generate Pareto solutions in all chromosome of ChromA can be deleted in the worker assignments process.

$$q_A \succeq q_{BA_j} \quad (\forall q_A \in A, \forall q_{BA_j} \in BA_j) \quad (15)$$

Here,  $A$  represents a set of ChromB chromosomes that generate Pareto solutions in all ChromA chromosomes and  $BA_j$  represents a set of ChromB chromosomes that generate Pareto solutions in schedules generated using the  $j$ -th ChromA chromosome.  $a \succeq b$  indicates that  $a$  is not inferior to  $b$ . Pareto solutions are present in a set of this problem's active schedules. Active schedules are generated in the process of job assignment by Treatment2. The number of gene representations in ChromA chromosomes used to search for Pareto solutions is reduced by Treatment3. Treatment2 and Treatment3 are executed in the '**Proc**' procedure.

### 4.3 Methods for Worker Assignment

#### 4.3.1 Characteristics of worker assignment methods

In the '**Proc**' procedure of the HGA method in Fig.4, elements of ChromB chromosome are manipulated and schedules generated by using pairs of ChromA and ChromB chromosomes are evaluated. As for the '**Proc**' procedure, we adopt two types of methods described as follows;

(Procedure1) Genetic algorithm for the multi-objective optimization problem:

The genetic algorithm is named the 'LGA' method. The HGA method, into which the LGA method is incorporated, is named the 'HGA-LGA' method. A procedure of the LGA method is identical with that of the HGA method shown in Fig.4, when GT method and evaluation of schedules are introduced in '**Proc**'.

(Procedure2) Hybrid local search method composed of local search methods to minimize job tardiness and to minimize peak-peak value of operation time among workers:

Either of these local search methods is alternately selected and utilized to assign workers to operations. The hybrid local search method is named 'HLS' method. The HGA method, into which the HLS method, is incorporated is named the 'HGA-HLS' method.

#### 4.3.2 Hybrid local search method for worker assignment

The HGA-LGA method requires long computation time to search for Pareto solutions due to using probability for worker assignments. Therefore, The HLS method is introduced for worker assignment in order to seek Pareto solutions of high quality efficiently. Fig.5 shows a flowchart of the HLS

method. In this flowchart, the LSRL method is a local search method that is used to minimize average job tardiness and the LSWLB method is used to the method to minimize peak-to-peak value of operation time among workers. Either the LSRL method or the LSWLB method is alternately selected and it is continuously executed to control worker assignment aiming at searching a wide area for Pareto solutions using a small number of search points.

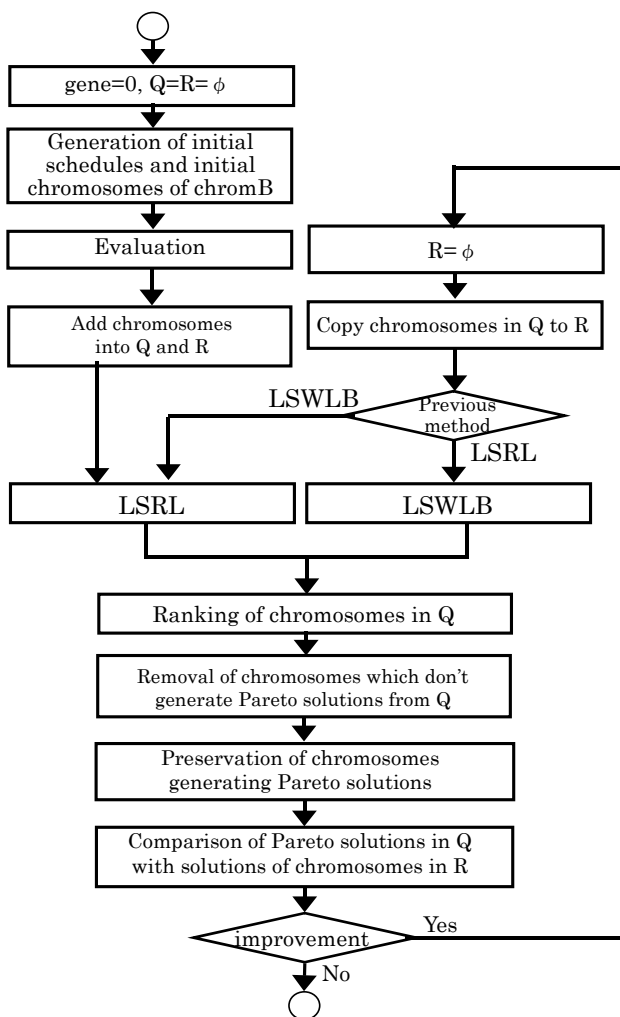


Fig. 5. Flowchart of HLS method

Figure 6 shows a schematic diagram of the HLS method's search process focusing on a single solution. Both axes of the coordinate system indicate the values of different objective functions. Any direction required to seek solutions consists of a combination of directions of independent axes on the coordinate system and Pareto solutions are alternately sought in either of the directions of the axes. The candidates for the Pareto solutions sought in the local search are used as starting

points for the other local search. A set of ChromB chromosomes that consist of the candidates is continuously updated. When the number of candidates and values of the criteria of all candidates in a set are identical with these of candidates in the previous set, the HLS method is complete.

The HLS method is shown as follows:

#### The HLS method

- (1) The number of chromosomes in population for ChromB chromosomes is initialized as  $gene=0$ . Sets of ChromB chromosomes generating Pareto solutions for the successive and the current generation sets are prepared and named Q and R, respectively. Then, these sets are initialized as  $Q=\phi$  and  $R=\phi$ .
- (2) The initial schedules and the initial arrays of ChromB are generated. Here, workers are determined so that each operation of all jobs completes earliest when the operation is selected from a priority order of jobs represented in ChromA in the GT method process.
- (3) The initial schedules are evaluated.
- (4) The chromB chromosomes are added to both Q and R.
- (5) The LSRL method is executed, go to step (7).
- (6) The LSWLB method is executed.
- (7) All ChromB chromosomes in set Q are ranked.
- (8) The chromosomes that do not yield Pareto solutions are removed from set Q.
- (9) If the values of all chromosomes' criteria in set R are identical with these of all chromosomes in set Q, go to step (13). Otherwise, go to step (10).
- (10) All chromosomes in set R are removed:  $R=\phi$ .
- (11) All chromosomes in set Q are copied into set R.
- (12) If the current local search method is the LSRL method, go to step (6). Otherwise, go back to step (5).
- (13) The HLS method completes.

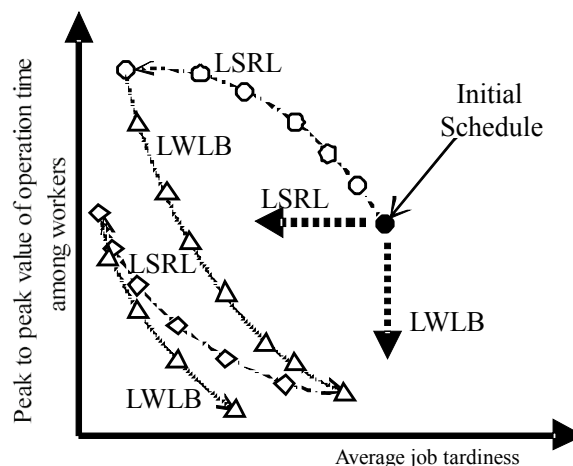


Fig. 6. Schematic diagram of search process of HLS method

4.3.3 Local search method to reduce average job tardiness

In order to reduce job tardiness in a schedule, workers assigned to operations of tardy jobs are continuously exchanged with other workers until the schedule is not improved. In the LSRL method, the operation assigned to worker  $k_{ij}$  is exchanged for that of worker  $k_x$  according to the following constraints:

$$k_{\hat{i}\hat{j}} \in \{k_{ij}\}, \tag{16}$$

$$L_{\hat{i}} < L_i \wedge 0 < L_i, \tag{17}$$

$$r_{k_{\hat{i}\hat{j}}} < C_{ij} \wedge ST_{ij} < C_{k_{\hat{i}\hat{j}}}, \tag{18}$$

$$f_A(k_{\hat{i}\hat{j}}) = \max(r_{k_{\hat{i}\hat{j}}}, r_{ij}) + \eta_{k_{\hat{i}\hat{j}}ij} \bar{p}_{ij}, \tag{19}$$

$$k_B \in \{k_{ij} \mid \neg eq.(17) \vee \neg eq.(18)\}, \tag{20}$$

$$f_B(k_B) = r_{ij} + \eta_{k_Bij} \bar{p}_{ij}, \tag{21}$$

$$k_x = \arg \min_{k_{\hat{i}\hat{j}}, k_B} \left\{ \min(f_A(k_{\hat{i}\hat{j}}), f_B(k_B)) \mid f_A(k_{\hat{i}\hat{j}}) < C_{ij}, f_B(k_B) < C_{ij} \right\}. \tag{22}$$

Here, worker  $k_{ij}$  is assigned to the  $j$ -th operation  $O_{ij}$  of job  $i$ ,  $r_{ij}$  denotes start time of operation  $O_{ij}$ , and  $r_{k_{\hat{i}\hat{j}}}$  denotes the possible start time of worker  $k_{\hat{i}\hat{j}}$  assigned to the  $\hat{j}$ -th operation  $O_{\hat{i}\hat{j}}$  of job  $\hat{i}$ . Fig.7 is a schematic diagram of Gantt chart explaining the conditions of the exchange of workers assigned to operations which involve tardy jobs. When the period of processing-time of a specific operation is overlapped with the period of processing time of operation  $O_{ij}$  a worker assigned to the specific operation is available to be exchanged with a worker assigned to operation  $O_{ij}$ .  $C_{ij}$  and  $ST_{ij}$  denote the completion time of operation  $O_{ij}$  and the start time of operation  $O_{ij}$ , respectively. Equation (18) shows that a specific operation overlapped with an operation of a tardy job is chosen for workers' exchange. In addition, equation (22) guarantees that completion time of an exchanged operation comes earlier than the original completion time of the operation. Since these constraints draw reduction of tardiness from tardy jobs, iterative exchange process would reduce average tardiness of the schedule. The LSRL procedure takes the follows:

The LSRL method

- (1) The maximum number of chromosomes in a population takes Maxpop.
- (2) Set Q for ChromB is transferred from the HGA process.
- (3) Sets S and T for ChromB are determined and set to be empty: S=  $\phi$ , T=  $\phi$ .

- (4) A single chromosome of ChromB is extracted from set Q, and the chromosome is added into set T. Go to step (5). When set Q is empty, go to step (18).
- (5) List of job number, JL, is determined to be empty:  $JL = \phi$ .
- (6) A schedule is generated by using the ChromB chromosome extracted in step (4) or (17).
- (7) Job numbers of tardy jobs are added into list JL.
- (8) Average job tardiness of the schedule is determined as the current best solution  $\hat{L}_{best}$ .
- (9) The number of chromosomes in a temporary population is initialized:  $pop=0$ .
- (10) When the value of 'pop' is identical with 'Maxpop', go to (14). Otherwise, go to step (11).
- (11) The number of chromosomes in the temporary population is updated:  $pop=pop+1$ .
- (12) The chromosomes of ChromB are generated based on the following process:
  - (12-1) All elements of a ChromB chromosome extracted in step (4) are copied to a temporary chromosome named 'ChromBX',
  - (12-2) Job  $i$  is selected from list JL,
  - (12-3) Operation  $O_{ij}$  of job  $i$  is selected using a random number,
  - (12-4) Worker  $k_x$  and an operation assigned by worker  $k_x$  are chosen by using the equations between (16) and (22). Then, worker  $k_{ij}$  assigned to operation  $O_{ij}$  is exchanged with worker  $k_x$ , and the start and completion times of operation  $O_{ij}$  are modified. When worker  $k_{ij}$  is assigned to another operation, the operation's start and completion times are modified,
  - (12-5) The elements of ChromBX chromosome are modified based on a worker exchange,
  - (12-6) When an extracted job is the last element in list JL, go to step (12-7). Otherwise, go to step (12-2), and
  - (12-7) The ChromBX chromosome is added to set S.
- (13) Return to step (10).
- (14) Schedules are generated by using all chromosomes in set S combined with a ChromA chromosome. Then, criteria values of all schedules are calculated.
- (15) The best schedule is extracted from all schedules with regard to job tardiness and the criterion value is determined as  $L_{best}$ .
- (16) All elements of set S are transferred to set T.
- (17) When  $\hat{L}_{best} \geq L_{best}$ ,  $\hat{L}_{best}$  takes  $L_{best}$  and a chromosome corresponding to  $L_{best}$  is determined as ChromB chromosome for the hybrid local search; if this is the case, go to step(5). Otherwise, go to step (4).
- (18) When all chromosomes in set T are transferred to set Q, return to the HLS method.

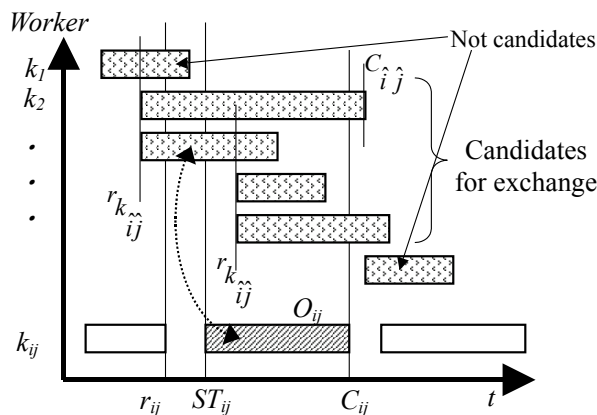


Fig. 7. Gantt chart explaining selection of an operation for change of worker

4.3.4 Local search method to reduce peak-to-peak value of operation time among all workers

In the LSWLB method, worker  $k_{\max}$  and worker  $k_{\min}$  are iteratively exchanged until peak-to-peak value of workers' operation time can not be reduced. Here,  $k_{\max}$  and  $k_{\min}$  represent a worker with the maximum operation time and a worker with the minimum operation time, respectively. The peak-to-peak value of workers' operation time  $\delta P$  is defined as the following equations:

$$\delta P = P_{k_{\max}} - P_{k_{\min}} \tag{23}$$

$$P_k = \sum_{i=1}^m \sum_{j=1}^n \bar{p}_{ij} \eta_{kij} \delta_{kij} = \sum_{s=1}^{s_k} \bar{p}_{ks}^* \eta_{ks}^* \tag{24}$$

Here,  $P_k$  denotes the operation time of worker  $k$ .  $\bar{p}_{ks}^*$ ,  $\eta_{ks}^*$ , and  $s_k$  denote standard time of  $s$ -th operation  $O_{ks}^*$  assigned to worker  $k$ , the efficiency of worker  $k$  for operation  $O_{ks}^*$ , and the number of available workers to be assigned to operation  $O_{ks}^*$ , respectively.

When worker  $k_{\max}$  assigned to operation  $O_{k_{\max} s}^*$  is exchanged with worker  $k_{\min}$ , total operation time of each worker is calculated in the following equations:

$$P_{k_{\max}}^{n+1} = P_{k_{\max}}^n - \sum_{\hat{s}=1}^{\hat{s}^*} \bar{p}_{k_{\max} \hat{s}}^* \eta_{k_{\max} \hat{s}}^* \tag{25}$$

$$P_{k_{\min}}^{n+1} = P_{k_{\min}}^n + \sum_{\hat{s}=1}^{\hat{s}^*} \bar{p}_{k_{\max} \hat{s}}^* \eta_{k_{\min} \hat{s}}^* \tag{26}$$

$n$  and  $n+1$  denote the condition before and after of the exchange of workers, respectively.  $\hat{s}$  and  $\hat{s}^*$  denote the order number of operations assigned to worker  $k_{\max}$  and the number of operations required to exchange workers, respectively. Operations  $O_{k_{\max} s}^*$  are continuously chosen from op-

erations assigned worker  $k_{\max}$  under  $\delta P > 0$  with using random number and the total number of the chosen operations is taken as  $\hat{s}^*$ . After workers are exchanged, the schedules are generated and the criteria values are calculated. As a result, the data on workers  $k_{\max}$  and  $k_{\min}$  are updated. Operations are selected and assigned to these workers in the same way. The selected workers are exchanged again. This process is repeated until the peak-to-peak value of the total operations time for all workers can not be further reduced. The LSWLB method takes the following steps:

#### The LSWLB method

- (1) Set Q of ChromB chromosomes generating Pareto solutions is transferred using the HGA method.
- (2) Set S for ChromB chromosomes is determined as empty:  $S = \phi$ .
- (3) After a single chromosome of ChromB is extracted from set Q, go to step (4). When set Q is empty, go to step (8).
- (4) A schedule is generated using the GT method and the extracted chromosome.
- (5) Worker  $k_{\max}$  and worker  $k_{\min}$  are identified in the schedule. The total operation times,  $PT_{\max}$  and  $PT_{\min}$ , are calculated with regard to workers  $k_{\max}$  and  $k_{\min}$ .
- (6) The peak-to-peak value of operation time among workers is calculated as  $\delta P_0 = PT_{\max} - PT_{\min}$ .
- (7) The elements of ChromB are modified according to the following steps:
  - (7-1) All elements of ChromB chromosome are copied to elements of a new chromosome of ChromBX.
  - (7-2)  $\delta P_0$  takes the value of  $\delta P$ :  $\delta P_0 = \delta P$ .
  - (7-3) Operation  $O_{k_{\max} s}^*$  assigned to worker  $k_{\max}$  is chosen using a random number. The worker assigned to the operation is exchanged with worker  $k_{\min}$ .
  - (7-4) Workers  $k_{\max}$  and  $k_{\min}$  are identified, and the value of  $\delta P$  is calculated for all workers.
  - (7-5) The elements of ChromBX chromosome are modified according to the exchange of workers.
  - (7-6) The ChromBX chromosome is added to set S.
  - (7-7) If  $\delta P \geq \delta P_0$ , go back to step (3). Otherwise, go to step (7-8).
  - (7-8) If all elements of ChromBX chromosome are copied to elements of a new chromosome of ChromBX, then return to step (7-2).
- (8) Schedules are generated using all chromosomes in set S and the criteria of all schedules are evaluated.
- (9) All chromosomes of ChromB in set S are transferred to set Q.
- (10) Return to the HLS method.

## 5 Evaluation of a Hybrid Genetic Algorithm

### 5.1 Models

In this chapter, the effectiveness of our hybrid Genetic Algorithm is evaluated on a simple job shop model. Three types of job shop models proposed by Fisher and Thompson [5] are utilized: ft06, ft10, and ft20. These original problems are created from a typical simple job shop process [6]. All job sequences and every operation time of jobs have been decided randomly in these problems. To evaluate the proposed algorithm for the tradeoff relationship, conditions of worker assignment and job due-dates are added to the original problems.

As for conditions for job due-date, due-date of job  $i$  is defined in equation (27) by TWK method [7]:

$$d_i = k_D \sum_{j=1}^{m_i} \bar{p}_{ij} + r_i \quad (27)$$

Here,  $k_D$  and  $r_i$  represent the due-date coefficient of job  $i$  and the release time of job  $i$ , respectively. Furthermore, all jobs are released at the identical time and the release time takes zero:  $r_i = 0$ .

As for conditions for worker assignment, the workers are placed in one of three levels. The coefficient  $\eta_{kij}$  depends on worker's skill level and it is not influenced by characteristics of operations. That is,  $\eta_{kij} = \eta_k$  is defined. Coefficient  $\eta_k$  takes one of three constant values, 0.8, 1.0, and 1.2. The constant values of 0.8, 1.0, and 1.2 correspond to the high skilled worker, the average skilled worker, and the low skilled worker, respectively.

Table1 shows conditions used in a numerical experiment. In the table, the number of jobs, the number of machines, the number of workers, and due-date coefficient of all jobs are shown. Worker levels 'H', 'A', and 'L' denote high-skilled, average-skilled, and low-skilled. Probability for mutation takes 0.05.

**Table 1.** Conditions for a numerical experiment

	Job	Machine	Worker (Level H,A,L)	Due-date coefficient
ft06	6	6	6 (2,2,2)	1.2
ft10	10	10	6 (2,2,2)	1.5
ft20	20	5	6 (2,2,2)	1.5

### 5.2 Procedure introduced in scheduling methods

To evaluate the hybrid Genetic Algorithm, we compare it with a Simple GA method. A two-point crossover is adopted in all types of genetic algorithms utilized in the numerical experiment with

regard to ChromA and ChromB chromosomes. As for ChromA, any two elements are chosen from the parent chromosomes. Then, job numbers are extracted at two elements and the numbers are exchanged. If all job numbers are not identified by all the number of the job operations after the crossover.

To prevent offspring's chromosomes from that the number of any job number is not same as the number of operations in the chromosomes, job numbers and positions of the numbers on parts of parent's chromosomes prepared for crossover are assigned to new chromosomes for offspring counting the number of operations for each job.

As for ChromB, a two-point crossover is used in a one-dimensional array for each job. In the mutation process, the swap and shift procedures are adopted in both types of chromosomes. A sharing procedure is introduced in the SGA, and HGA-HLS processes. However, the procedure is not introduced in HGA-LGA method because the solutions almost uniformly disperse on the Pareto front. Following this we use three processes - roulette wheel selection, Pareto reservation, and parallel selection - to select chromosomes for the successive generation.

### 5.3 Evaluation of the Hybrid Genetic Algorithm

To evaluate Pareto solutions obtained by the hybrid Genetic Algorithm, two types of measures [8] are introduced:

- > Measure of proximity of solutions to Pareto front,  $C(X_1, X_2)$ , and
- > Measure of diversity of Pareto solutions,  $S_B$ .

$C(X_1, X_2)$  and  $S_B$  are defined by the following equations:

$$C(X_1, X_2) = \frac{|\{a_2 \in X_2 \mid \exists a_1 \in X_1 : a_1 \preceq a_2\}|}{|X_2|} \quad (28)$$

$$S_B = \sqrt{\frac{1}{|X_2|} \sum_{i=1}^{|X_2|} (d_i - \bar{d})^2} \quad (29)$$

$$d_i = \|s_i - s_p\| \quad (30)$$

$$s_p = \arg \min_i \|s_i\| \quad (31)$$

$$s_i = \{y_{i1} / y_{z1}^{\max}, y_{i2} / y_{z2}^{\max}\}^T \quad (32)$$

Here,  $X_1, X_2$  denotes two different sets to compare.  $a \succeq b$  denotes that  $a$  takes superior criteria values to  $b$ , or  $a$  and  $b$  take the identical criteria values.  $|X_2|$  is the number of solutions in set  $X_2$ . The definition of  $S_B$  is modified to assess distribution of the Pareto solutions.

$d_i, \bar{d}$  are the Euclidean distance from the nearest solution from the origin of coordinate system for Pareto solutions, and average of  $d_i$ , respectively.  $y_{i1}, y_{z1}^{\max}$  are the value of objective function 1 of Pareto solution  $i$  obtained by each method, and the maximum value of objective function 1 of Pareto solution  $i$  obtained by the HGA-LGA method. When the value of  $C(X_1, X_2)$  takes a large number, the Pareto solutions in set  $X_2$  are superior to these in set  $X_1$ . When the value of  $S_B$  takes a large number, the distribution of the Pareto solutions widely spreads out in the Euclidean coordinate system.

As for ft06, Table 2 shows conditions utilized in three types of genetic algorithms. As for SGA method, population takes ten times as many as that used for HGA-LGA method and generation is determined as the condition of calculation time which is several times longer than that required in the HGA-LGA methods. Population and generation for HGA-HLS method are determined as the condition for the almost same elapsed time as that for the HGA-LGA method. To accurately estimate elapsed time in the calculation of HGA-HLS method is difficult because of iterative procedure to converge solutions in HLS method.

Table 3 shows proximity of the Pareto solutions between two different methods and Fig.8 shows the Pareto fronts obtained by different methods. In Fig.8, “One dimensional” denotes the minimum value of average job tardiness obtained by the HGA-HLS method for a single objective function in which only LSRL procedure is used. Table 4 shows the diversity of the Pareto solutions. Table 3 and Fig.8 show that the HGA-HLS and the HGA-LGA methods are superior to the SGA method with regard to  $C(X_1, X_2)$ , although the elapsed time of the SGA method is six times longer than the other methods. These results show that separation and hybrid control of the two types improves the quality of Pareto solutions and reduces calculation time.

Figure 8 shows that the Pareto solutions attained by the HGA-LGA method are uniformly distributed on the coordinate system and that this method is not required to introduce a sharing procedure. On the other hand, the HGA-HLS method produces the Pareto front which almost corresponds to that generated by the HGA-LGA method. However, since the number of Pareto solutions is less than that obtained by the HGA-LGA process, a gap of the distribution of Pareto solutions appears between the adjoined solutions in the vicinity of the coordinate system’s origin.

Figure 9 and 10 show the Pareto fronts obtained from the ft10 and the ft20 problems. Table 5 describes the condition used for the experiments. Table 6 and 7 show the proximity of solutions to Pareto front with regard to the ft10 and the ft20 problems, respectively.

**Table 2.** Numerical conditions for genetic algorithms (ft06)

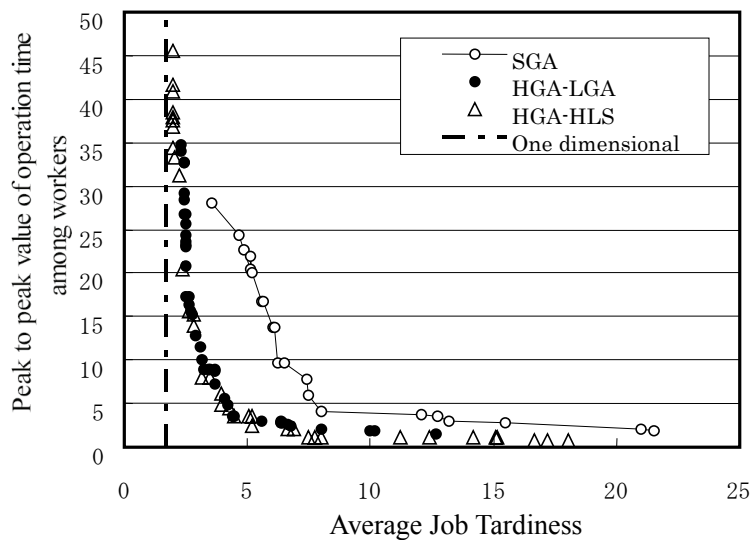
Method	ChromA		ChromB	
	Population	Generation	Population	Generation
SGA	500	2000	—	—
HGA-LGA	50	50	50	50
HGA-HLS	50	50	LSRL : Maxpop = 30	

**Table 3.** Proximity of solutions to the Pareto front and calculation time (ft06)

$X_1 \backslash X_2$	Simple GA	HGA-LGA	HGA-HLS	Time
SGA	—	0.86	0.72	2 hr 32 min
HGA-LGA	0.0	—	0.47	23 min 22 sec
HGA-HLS	0.0	0.07	—	18 min 57 sec

**Table 4.** Diversity of Pareto solutions (ft06)

SGA	HGA-LGA	HGA-HLS
0.327	0.271	0.40



**Fig. 8.** The Pareto fronts obtained by different methods (ft06)

Table 8 shows the computation time of these methods. The number of operations for the ft10 and the ft20 problems is larger than that of the ft06 problem, so that more computational time is required to make the critical values of the Pareto solutions converge. Therefore, we set that the generations for HGA-LGA and HGA-HLS methods are twice the generation required for the ft06 problem in these problems. These figures show that the HGA-LGA and HGA-HLS methods obviously generate the better Pareto solutions than the SGA method. Furthermore, the HGA-HLS method generates the solutions in half computational time required by the HGA-LGA method. As shown Fig.10, the Pareto solutions obtained by the HGA-HLS method are distributed in the areas of small value of individual objective functions. However, the Pareto front involves a large gap between adjoining solutions.

As for worker assignment, worker numbers in elements of ChromB chromosome are required to change for any worker numbers in order to evaluate all combinations of worker assignments, so that genetic operators used in the LGA process are difficult to search for ChromB chromosomes of Pareto solutions. Since the number of machines is not identical with the number of workers in the ft10 and the ft20 problems, these problems are more difficult to search for the Pareto solutions with using genetic operators than ft06 problem. Because worker numbers are directly chosen to improve Pareto solutions in the HLS process, the process is effective to adopt in HGA method. From Fig.9, 10 and Table.6, 7, and 8, we conclude that the HGA-HLS method is effective to resolve the following problems: the problem in which the number of workers is not identical with the number of machines and the problem which involves a large number of workers.

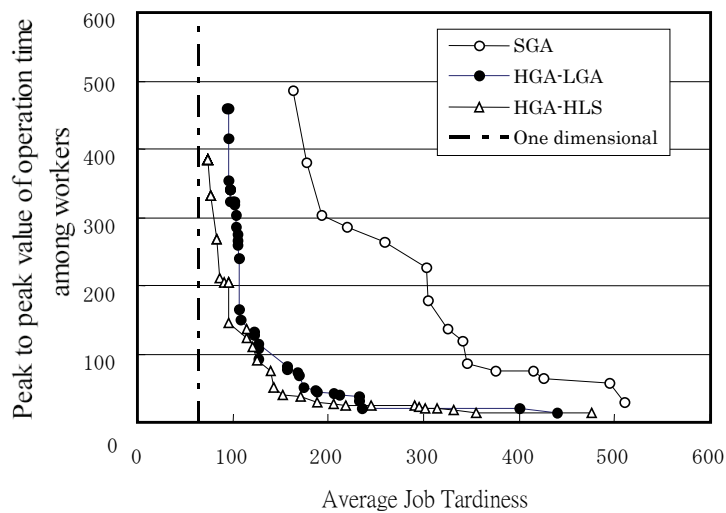


Fig. 9. The Pareto fronts obtained by different methods (ft10)

Table 5. Numerical conditions for genetic algorithms (ft10, ft20)

Method	ChromA		ChromB	
	Population	Generation	Population	Generation
SGA	500	2000	-	-
HGA-LGA	50	100	50	50
HGA-HLS	50	100	LSRL: Maxpop = 30	

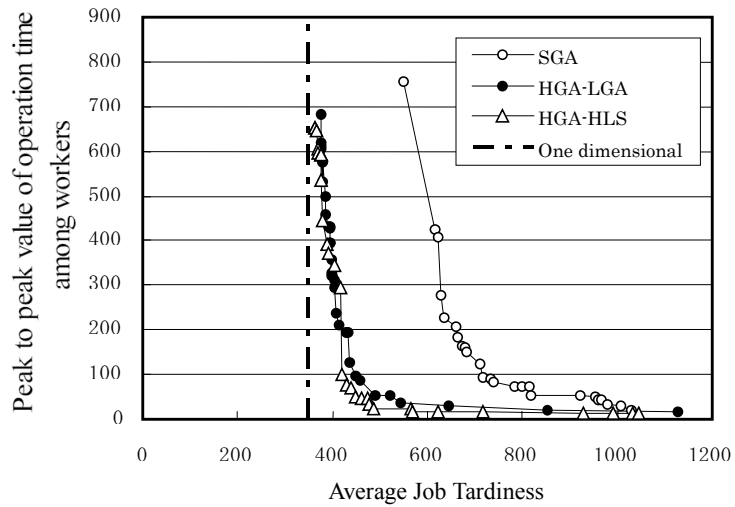


Fig. 10. The Pareto fronts obtained by different methods (ft20)

Table 6. Proximity of solutions to Pareto front (ft10)

$X_1 \backslash X_2$	SGA	HGA-LGA	HGA-HLS
SGA	-	1.0	1.0
HGA-LGA	0.0	-	0.82
HGA-HLS	0.0	0.026	-

Table 7. Proximity of solutions to Pareto front (ft20)

$X_1 \backslash X_2$	SGA	HGA-LGA	HGA-HLS
SGA	-	0.97	0.93
HGA-LGA	0.0	-	0.93
HGA-HLS	0.0	0.152	-

Table 8. Calculation time required to resolve ft10 and ft20 problems

Method	ft10	ft20
SGA	3hr 6min 25sec	3hr 19min 18sec
HGA-LGA	3hr 8min 31sec	3hr 18min 49sec
HGA-HLS	1hr 4min 24sec	1hr 35min 14sec

The HGA-HLS process also results in a Pareto front with large gaps as shown in Fig.10. This defect suggests that a function to search for new Pareto solutions in the gap area should be incorporated into the HGA-HLS method in order to improve the Pareto front. From these results, we summarize the characteristics of the proposed hybrid Genetic Algorithm as follows:

- (1) The hybrid Genetic Algorithm is effective in seeking Pareto solutions in a short calculation time in a scheduling problem that includes simultaneous controls of job and worker assignments,
- (2) The HGA-HLS process requires a shorter calculation time than the HGA-LGA process to generate the same grade of Pareto solutions for large scale problems,
- (3) The HGA-HLS process generates Pareto solutions of small values of individual objective functions and it provides higher diversity and better proximity of Pareto solutions than another HGA method. However, the Pareto front involves a large gap between adjoined solutions in the vicinity of the coordinate system's origin.

A personal computer with a Pentium4, 2.66 GHz processor and 512MByte RAM is used for this numerical experiment.

## **6 Conclusion**

In this study, we examine a scheduling problem including simultaneous assignments of jobs and workers. This problem is treated as a multi-objective problem to minimize average job tardiness and peak-to-peak value of workers' total operation time. To resolve the problem, we divided two types of assignments into individual controls and developed a hybrid Genetic Algorithm. The genetic algorithm framework is used to control job assignment and worker assignment is controlled based on each job priority order for job assignment. Two types of search procedures are adopted for worker assignment control; a genetic algorithm and a hybrid local search method. The hybrid local search method is constructed of local search methods for different single objective functions in order to efficiently search for worker assignment for Pareto solutions.

The hybrid Genetic Algorithm method is examined on simple job shop models to evaluate its performance. The results of numerical experiments show that the hybrid GA method can generate the better Pareto solutions than Simple GA method. As for worker assignment, the results show that the hybrid local search method can search for better Pareto solutions in a shorter computational time than a genetic algorithm. Since workers are selected by equations for constraints in the hybrid local search processes, the HGA-HLS method can be efficient to obtain Pareto solutions of high quality.

The proposed hybrid scheduling method can be easily installed into a parallel computing environment, so that it is expected to reduce calculation time required to generate Pareto solutions by using the parallel computing environment.

## Acknowledgements

This research was supported by Kansai University's Overseas Research program for the year of 2005.

## References

1. Treleven, M.: A Review of the Dual Resource Constrained System Research. *IEE Transactions*, 21, 3 (1989) 279-287
2. Elmaraghy, H.: A Genetic Algorithm Based Approach for Scheduling of Dual-Resource Constrained Manufacturing Systems. *Annals of the CIRP*, 48, 1 (1999) 369-372
3. Gen, M., Cheng, R. (eds.): *Genetic Algorithms & Engineering Optimization*. Wiley-Interscience Publication (2000) 118-123
4. Giffier, B., G.L.Thompson: Algorithms for Solving Production Scheduling Problems. *Operations Research*. 8(4) (1960) 487-503
5. Fisher, H.G., Thompson, L.: Probabilistic learning combinations of local job-shop scheduling rules, J.F.Muth, G.L.Thompson (eds.), *Industrial Scheduling*, Prentice Hall (1963) 225-251
6. Conway, R.W., Maxwell, W.L., Miller, L.W: *Theory of Scheduling*, Addison-Wesley (1967) 5-6
7. Conway, R.W., Maxwell, W.L., Miller, L.W: *Theory of Scheduling*, Addison-Wesley (1967) 328-334
8. Formiga, K.T.M., Chudhry, F.H., Cheung, P.B., Reis, L.F. R.: Optimal Design of Water Distribution System by Multi-objective Evolutionary Methods, *Fonseca, C.M.et al.(Eds.), EMO2003, LNCS 2632* (2003) 677-691

## **An Improvement Heuristic for the Timetabling Problem**

Aldy Gunawan\*, Kien Ming Ng, and Kim Leng Poh

Department of Industrial and Systems Engineering, Faculty of Engineering,  
National University of Singapore, 1 Engineering Drive 2, S(117576), Singapore  
{g0500710, isenkm, isepohkl}@nus.edu.sg

**Abstract.** This paper formulates a timetabling problem, which is often encountered in a university, as a mathematical programming model. The proposed model combines both teacher assignment and course scheduling problems simultaneously, which causes the entire model to become more complex. We propose an improvement heuristic algorithm to solve such a model. The proposed algorithm has been tested with several randomly generated datasets of sizes that are comparable to those occurring in a university in Indonesia. The computational results show that the improvement heuristic is not only able to obtain good solutions, but is also able to do so within reasonable computational time.

**Keywords:** Timetabling, mathematical programming, improvement heuristic algorithm.

### **1 Introduction**

Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space time, in such a way as to satisfy as nearly as possible a set of desirable objectives [1]. One of the key applications of timetabling is in educational timetabling. In this paper, we focus solely on the course timetabling problem at the university level.

The course timetabling problem is further classified into five sub-problems: teacher assignment, class-teacher timetabling, course scheduling, student scheduling, and classroom assignment [2]. The teacher assignment problem focuses on allocating teachers to courses, without considering the allocation of courses to time periods. The course scheduling problem only focuses on allocating courses to time periods. It is often assumed that the allocation of teachers to courses has been done earlier before the actual scheduling of time periods to courses.

---

\* Corresponding Author. Tel.: +65-6516-2208, Fax: +65-6516-1434, Email: g0500710@nus.edu.sg.

This paper attempts to address both teaching assignment and course scheduling problems simultaneously, and a model combining these two sub-problems is proposed. Another emphasis of this paper is to develop a simple improvement heuristic to tackle this timetabling problem. Such a timetabling problem that we address has arisen in the context of a university in Indonesia.

The paper proceeds as follows: Section 2 presents the timetabling problem and gives a brief literature review of this problem, as well as some of its basic requirements. Section 3 describes a mathematical programming model for the problem, together with the computational difficulties of the model. A proposed heuristic is then described in Section 4, and the computational results and comparisons are reported in Section 5. Finally, Section 6 gives some conclusions and possible directions for future research.

## 2 The Timetabling Problem

The course timetabling problem discussed in this paper appears in the context of a university in Indonesia. A number of common definitions and terms for the problem are first explained in detail. A course refers to a subject taught one or more times within a week. Each course requires a certain number of consecutive time periods per week. Due to the capacity of the classrooms and the number of students registered, some courses have to be taught repeatedly by the same teacher or by different teachers. Each of these repeated courses is known as a course section.

Before the new semester starts, teachers are requested to decide the courses they are willing to teach, along with their preferred days and time periods to teach the courses. The primary problem faced is how to assign teachers to their preferred courses and course sections and then to schedule course sections to time periods over a week based on the teachers' preferences. Although the decisions on teacher assignment and course scheduling problems could mathematically be made in one step, in practice, this is generally done in two separate steps [3]. Most of the papers only focus on one of the sub-problems, such as Andrew and Collins [3], Breslaw [4], Harwood and Lawless [5], Schniederjans and Kim [6], Tillett [7], and Wang [8] that only focus on the teacher assignment problem.

The course scheduling problem has also been widely studied with many solution methods being proposed. Yu and Sung [9] applied a genetic algorithm for solving course scheduling problems, and simulated annealing was applied to course scheduling by Abramson [10] and Abramson et al. [11]. Hertz [12] proposed a tabu search algorithm for solving course scheduling problems, while Badri [13] proposed a model for both course scheduling and teacher assignment problems. Through a two-stage optimization procedure, the model in Badri [13] seeks to maximize teacher-course preferences in assigning teachers to courses, and then maximize teacher-time preferences in allocating courses to time periods.

Two common techniques for solving the timetabling problems are exact and heuristic methods. Exact methods include formulating and solving integer programming models of the timetabling problems. One of the recent applications of integer programming to the course scheduling problem

was presented by Daskalaki et al. [14]. We adopt a similar approach by formulating a mathematical programming model that considers both teacher assignment and course scheduling simultaneously. This is also an extension of the basic mathematical programming model proposed by Gunawan et al. [15], in which it is shown that timetabling problems with data sizes comparable to that of an institution can be solved with the help of this basic model.

For large problems, it could be difficult to find and prove the existence of an optimal solution, especially within short computing times [16]. It would be necessary to develop a heuristic approach in order to find a good solution within a reasonable amount of time. In general, the heuristic procedure consists of two phases [17]: initial and improvement phases. In the first phase, we try to find a feasible solution. If there is no feasible solution, we need to rearrange the assignments or relax some requirements or constraints. In the improvement phase, we improve the feasible solution obtained until a local optimum is reached. Some examples of heuristic approaches for solving the course scheduling problem were proposed by Aubin and Ferland [18], Loo et al. [19] and Wright [20].

The university timetabling problem has special features that highly depend on the university's characteristics, such as the courses taught, the teachers and the availability of resources as well. Several rules or requirements imposed on the problem are as follows:

- a. For each teacher, course and time preferences are obeyed.
- b. For each course, only one section can be conducted in every time period.
- c. Each teacher has to teach at least one course and cannot teach more than his/her maximum load. Here, the load refers to the number of courses that are assigned to the teacher.
- d. The number of teachers who can teach for each course is limited.
- e. All course sections have to be spread evenly throughout a week, so that for a particular course, only one section can be conducted every day.
- f. Each teacher can only teach at most one course section in a particular time period.
- g. The number of course sections taught cannot exceed the number of classrooms available during each time period.
- h. All sections for a particular course must be scheduled.
- i. Each course section can only be taught by one teacher.
- j. Each teacher will not be assigned courses that he/she is unable to teach.
- k. All the course sections taught by a teacher will be spread out evenly during a week.
- l. Each course section has to be scheduled in a certain number of time periods consecutively.

The first requirement is considered as a preference and will be incorporated into the objective function, while the rest will be regarded as constraints that cannot be violated.

### 3 The Proposed Mathematical Programming Model

In order to study the computational effort involved in solving the problem of interest, the following mathematical programming model is proposed. We define the following sets to be used in the model:

- $I$  set of all teachers
- $J$  set of all courses
- $K$  set of all course sections
- $L$  set of all days available
- $M$  set of all time periods available
- $J_i$  set of courses that could be taught by teacher  $i$  ( $i \in I$ )
- $K_j$  set of sections of course  $j$  ( $j \in J$ )

We also define the following data parameters for the model:

- $N_i$  maximum load (in terms of the number of courses) of teacher  $i$  ( $i \in I$ )
- $C$  number of classrooms available per time period
- $H_j$  number of time periods required for course  $j$  ( $j \in J$ )
- $PC_{ij}$  value given by teacher  $i$  on the preference to be assigned to teach course  $j$  ( $i \in I, j \in J$ )
- $PT_{ilm}$  value given by teacher  $i$  on the preference to be assigned to teach in day  $l$  and time period  $m$  ( $i \in I, l \in L, m \in M$ )
- $LT_j$  minimum number of teachers that could teach course  $j$  ( $j \in J$ )
- $UT_j$  maximum number of teachers that could teach course  $j$  ( $j \in J$ )
- $S_j$  number of sections of course  $j$  ( $j \in J$ )

Finally, the following decision variables will be required to define the problem:

- $X_{ijklm} = 1$  if teacher  $i$  teaches course  $j$  section  $k$  on day  $l$  and at time period  $m$ ; 0 otherwise ( $i \in I, j \in J, k \in K_j, l \in L, m \in M$ )
- $Y_{ijk} = 1$  if teacher  $i$  teaches course  $j$  section  $k$  on day  $l$ ; 0 otherwise ( $i \in I, j \in J, k \in K_j, l \in L$ )
- $U_{ijklm} = 1$  if teacher  $i$  teaches course  $j$  section  $k$  on day  $l$  and started at time period  $m$ ; 0 otherwise ( $i \in I, j \in J, k \in K_j, l \in L, m \in M$ )
- $P_{ij} = 1$  if teacher  $i$  teaches course  $j$ ; 0 otherwise ( $i \in I, j \in J$ )
- $L_i =$  number of course sections taught by teacher  $i$  ( $i \in I$ )
- $V_i =$  number of course sections taught by teacher  $i$  per day ( $i \in I$ )

For our problem, the objective function reflects a preference function that needs to be maximized. It refers to the total preferences of assigning courses to the teachers and the total preferences of assigning the days and time periods to the teachers for teaching the sections of the courses. Thus, the teachers can choose not only their course preferences, but they can also indicate their time preferences. We assume that these preferences are equally important. The objective function is described by the expression in equation (1):

$$\text{Maximize } \sum_{i \in I} \sum_{j \in J} PC_{ij} \times P_{ij} + \sum_{i \in I} \sum_{j \in J} \sum_{k \in K_j} \sum_{l \in L} \sum_{m \in M} PT_{ilm} \times X_{ijklm} \quad (1)$$

The following depicts some of the main constraints encountered in our timetabling problem.

$$\sum_{i \in I} \sum_{k \in K_j} X_{ijklm} \leq 1 \quad (j \in J, l \in L, m \in M) \tag{2}$$

Equation (2) ensures that for a particular course, only one section can be conducted in every time period.

$$P_{ij} = \left\lceil \sum_{k \in K_j} \sum_{l \in L} \frac{\left[ \sum_{m \in M} X_{ijklm} / H_j \right]}{S_j} \right\rceil \quad (i \in I, j \in J) \tag{3}$$

Equation (3) indicates that if teacher  $i$  teaches at least one section of course  $j$  during  $H_j$  consecutive time periods, the value of  $P_{ij}$  will be 1, meaning that teacher  $i$  teaches course  $j$ . Here,  $\lceil a \rceil$  denotes the smallest integer greater than or equal to  $a$

$$1 \leq \sum_{j \in J} P_{ij} \leq N_i \quad (i \in I) \tag{4}$$

Equation (4) represents the minimum and maximum number of courses taught by each teacher. It is assumed that each teacher has to teach at least one course.

$$LT_j \leq \sum_{i \in I} P_{ij} \leq UT_j \quad (j \in J) \tag{5}$$

Equation (5) limits the number of teachers who can teach for each course. The values of  $LT_j$  and  $UT_j$  are dependent on the number of sections for course  $j$ .

$$\sum_{m \in M} X_{ijklm} = Y_{ijkl} \times H_j \quad (i \in I, j \in J, k \in K_j, l \in L) \tag{6}$$

Equation (6) shows the relationship between variables  $Y_{ijkl}$  and  $X_{ijklm}$ . If teacher  $i$  teaches course  $j$  section  $k$  for  $H_j$  time periods on day  $l$ , the value of  $Y_{ijkl}$  is equal to 1.

$$\sum_{i \in I} \sum_{k \in K_j} Y_{ijkl} \leq 1 \quad (j \in J, l \in L) \tag{7}$$

Equation (7) ensures that for a particular course, at most one section can be taught each day.

$$\sum_{j \in J} \sum_{k \in K_j} X_{ijklm} \leq 1 \quad (i \in I, l \in L, m \in M) \tag{8}$$

Equation (8) ensures that each teacher can only teach at most one course section in a particular time period.

$$\sum_{i \in I} \sum_{j \in J} \sum_{k \in K_j} X_{ijklm} \leq C \quad (l \in L, m \in M) \tag{9}$$

Equation (9) represents the constraint that at each time period, the number of course sections taught could not be more than the number of classrooms available.

$$\sum_{i \in I} \sum_{k \in K_j} \sum_{l \in L} Y_{ijkl} = S_j \quad (j \in J) \tag{10}$$

Equation (10) states that all course sections must be scheduled.

$$\sum_{i \in I} \sum_{l \in L} Y_{ijkl} = 1 \quad (j \in J, k \in K_j) \tag{11}$$

Equation (11) ensures that each course section can only be taught by one teacher.

$$X_{ijklm} = 0 \quad (i \in I, j \notin J, k \in K_j, l \in L) \tag{12}$$

Equation (12) ensures that teachers will not be assigned courses that they are unable to teach.

$$\sum_{j \in J} \sum_{k \in K_j} \sum_{l \in L} Y_{ijkl} = L_i \quad (i \in I) \tag{13}$$

Equation (13) calculates the number of course sections taught by each teacher.

$$V_i = \left\lceil \frac{L_i}{5} \right\rceil \quad (i \in I) \tag{14}$$

Equation (14) calculates the upper bound on the number of course sections taught per day for each teacher.

$$\sum_{j \in J} \sum_{k \in K_j} Y_{ijkl} \leq V_i \quad (i \in I, l \in L) \tag{15}$$

Equation (15) ensures that the number of course sections taught by each teacher per day will not be more than the upper bound  $V_i$ . This constraint also tries to spread out evenly the total course sections taught by a particular teacher in a week.

The next three constraints deal with the requirement of consecutive time periods. We know that every course section requires a certain number of time periods. With these constraints, if course  $j$  section  $k$  taught by teacher  $i$  requires  $H_j$  consecutive time periods and the teacher is assigned to a given period of day  $l$  as the first period to be taught for the course section, then the teacher will also be assigned to the following  $(H_j - 1)$  periods.

$$\sum_{t=0}^{(H_j-1)} X_{ijkl(m+t)} \geq H_j \times U_{ijklm} \quad (i \in I, j \in J, k \in K_j, l \in L, m \in \{0, \dots, |M| - H_j\}) \tag{16}$$

In order to facilitate the modeling of this requirement, the variable  $U_{ijklm}$  is introduced. Equation (16) expresses the constraint that each course section has to be scheduled in  $H_j$  time periods consecutively. If section  $k$  of course  $j$  taught by teacher  $i$  is assigned to time period  $m_1$  of day  $l$ , then the following  $(h_j - 1)$  time periods should be assigned to the same course section. In this case, the variable  $U_{ijklm}$  would be 1, meaning that the particular course section is started at time period  $m_1$ .

$$\sum_{i \in I} \sum_{l \in L} \sum_{m=0}^{(|M|-H_j)} U_{ijklm} = 1 \quad (j \in J, k \in K_j) \tag{17}$$

Equation (17) ensures that for each course section, only one variable  $U_{ijklm}$  is equal to 1, meaning that the starting time period of a particular course section is only one time period.

$$\sum_{i \in I} \sum_{l \in L} \sum_{m \in M} X_{ijklm} = H_j \quad (j \in J, k \in K_j) \tag{18}$$

The number of time periods allocated to each course section has to be equal to the requirement of the course, and this is expressed by equation (18).

$$U_{ijklm} = 0 \quad (i \in I, j \in J, k \in K_j, l \in L, m \in \{(|M|-H_j+1), \dots, (|M|-1)\}) \tag{19}$$

A particular course section could not be started in certain time periods if the remaining time periods are less than the number of time periods required. In this case, we set the relevant variables  $U_{ijklm}$  to be zero.

$$U_{ijklm} = 0 \quad (i \in I, j \notin J_i, k \in K_j, l \in L, m \in \{0, \dots, (|M|-H_j)\}) \tag{20}$$

Similar to equation (12), equation (20) ensures that teachers will not be assigned to certain time periods for courses that they are unable to teach.

Note that equations (3) and (14) involve nonlinear functions of the decision variables but these can always be linearized by adding some decision variables and constraints. The maximum number of decision variables and constraints in the proposed mathematical model are  $(2|I||J||K||L||M| + |I||J||K||L| + |I||J|+2|I|)$  and  $(3|I||J||K||L||M| + 2|I||J||K||L| + |I||J||L||M| + |I||L||M| + |I||J| + 3|J||K| + |I||L| + |J||L| + |L||M| + 4|I| + 3|J|)$ , respectively.

To test the performance of the proposed model, it is implemented in ILOG OPL Studio 4.2 on a 2.6GHz Pentium IV PC with 512MB RAM that runs in Microsoft Windows XP operating system. Several data sets are generated in such a way that the data sets correspond to differing values of several parameters. Here, the parameters that were varied are the number of teachers, the number of courses, the number of sections for each course, the maximum load and the number of classrooms available.

The number of days per week is assumed to be five days and the number of time periods per day is eight periods except for problem type  $5 \times 5$ . Each data set consists of five randomly generated data instances. Two different types of data sets are generated and they are known as Group I and Group II data sets respectively. For Group I data sets, the number of sections for each

course is set to a fixed number, while the number of sections for each course varies in the Group II data sets. Tables 1 and 2 summarize the characteristics of each data set. For all the data sets, we also set the minimum and maximum number of teachers who can teach a particular course as shown in Table 3.

**Table 1.** The characteristics of Group I data sets

Data set	Number of teachers	Number of courses	Number of sections	Number of days	Number of time periods per day	Maximum load per teacher	Number of classrooms available
5×5_1	5	5	2	5	4	1	4
5×5_2	5	5	2	5	4	2	4
10×10_1	10	10	2	5	8	1	4
10×10_2	10	10	2	5	8	2	4
15×15_1	15	15	2	5	8	1	6
15×15_2	15	15	2	5	8	2	6
20×20_1	20	20	2	5	8	1	8
20×20_2	20	20	2	5	8	2	8

**Table 2.** The characteristics of Group II data sets

Data set	Number of teachers	Number of courses	Minimum number of sections	Maximum number of sections	Number of days	Number of time periods per day	Maximum load per teacher	Number of classrooms available
10×20_1	10	20	2	3	5	8	4	10
10×20_2	10	20	2	4	5	8	4	10
20×30_1	20	30	2	3	5	8	3	15
20×30_2	20	30	2	4	5	8	3	15
20×40_1	20	40	2	3	5	8	4	15
20×40_2	20	40	2	4	5	8	4	15
30×60_1	30	60	2	3	5	8	4	20
30×60_2	30	60	2	4	5	8	4	20

**Table 3.** The minimum and maximum number of teachers for each course

Number of sections	Minimum number of teachers (LT)	Maximum number of teachers (UT)
1	1	1
2	1	2
3	1	2
4	2	3

The maximum size of problems solvable within reasonable time is rather small due to the huge number of constraints and decision variables involved. Tables 4 and 5 summarize the average best known/optimal objective function values obtained and the average CPU time required to obtain the solutions for Group I and Group II data sets, respectively.

From Table 4, we observe that the average CPU time required to obtain the solution increases rapidly when the maximum load increases from one to two courses. The average objective function value of the optimal solutions is also increased when the maximum load is increased. This is because the chances for each teacher to be assigned the courses and time periods preferred will be higher and each course can be taught by more than one teacher. Similar observations can also be found from the results in Table 5.

**Table 4.** Computational results of Group I data sets

Data set	Average objective function value	Average CPU time (in seconds)
5×5_1	1,052	2.03
5×5_2	1,216	2.17
10×10_1	2,184	18.47
10×10_2	2,712	31.79
15×15_1	3,170	136.93
15×15_2	4,166	431.94
20×20_1	4,368	337.98
20×20_2	5,774	4,212.92

**Table 5.** Computational results of Group II data sets

Data set	Average objective function value	Average CPU time (in seconds)
10×20_1	7,766	1,183.11
10×20_2	7,934	1,273.15
20×30_1	10,875	9,746.36
20×30_2	13,468	36,019.99
20×40_1	-	-
20×40_2	-	-
30×60_1	-	-
30×60_2	-	-

The best known/optimal solution can be found for data sets with the number of teachers = 10 within a few minutes. For 20 teachers, we observed that the CPU time increases rapidly as shown in Table 5. The solutions can only be found within three to six hours. However, the optimal

solution for all data instances in data sets 20×40\_1, 20×40\_2, 30×60\_1 and 30×60\_2 could not be computed within the time limit of 24 hours.

These numerical results indicate that the computing time required to find an optimal solution to the problem becomes prohibitively large when the problem size increases. This also experimentally supports the theoretical results on the NP-completeness of timetabling problems [21]. Consequently, many heuristic algorithms were proposed and developed to deal with the timetabling problem in the literature as mentioned in the previous section. In the next section, we also propose a heuristic that can handle large problem sizes.

## 4 The Proposed Heuristic

Our problem consists of two common sub-problems in the university course timetabling problem: teacher assignment and course scheduling problems. Our proposed heuristic will solve these sub-problems iteratively, and it comprises of three main phases: (1) pre-processing, (2) construction, and (3) improvement (see Figure 1). Each of these phases is further described below.

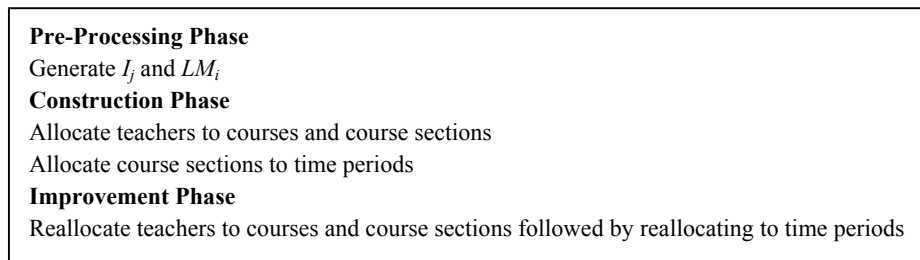


Fig. 1. The proposed heuristic algorithm

### 4.1 Pre-Processing Phase

Two processes are involved in this phase. The given course preferences for each teacher  $i$  being allocated to each course  $j$  in such a way that each course  $j$  has a list of teachers who are willing to teach, are sorted in non-increasing order, which is called  $I_j$ . The given time period preferences (day  $l$  and time period  $m$ ) for each teacher  $i$  are also sorted in non-increasing order, which is called  $LM_i$ . The time complexity for these processes is  $O(|I|^2|J|)$  and  $O(|I||L|^2|M|^2)$  respectively.

### 4.2 Construction Phase

In this phase, a feasible solution is constructed by accommodating as many course and time period preferences as possible. Each course should be allocated to the teacher who has the highest preference and scheduled to the time periods with the highest time period preferences as well. The

entire phase is started with the allocation of teachers to courses and course sections (teacher assignment problem).

In order to generate a feasible initial solution, a *Checking procedure* has to be conducted. The purpose of this procedure is to prevent a teacher from teaching more than the maximum load allowed (in terms of the number of courses). When the selected teacher has reached or exceeded the maximum load, the next teacher with less preference will be considered. In this construction phase, we assume that the number of selected teachers for each course depends on the minimum requirement (see Table 3).

However, it is possible that we will not be able to find any teacher from the list  $I_j$ . For this case, we can choose a teacher in list  $I_j$  who has the lowest number of courses allocated by relaxing requirement  $b$  in Section 2. However, an infeasible initial solution could be generated. Some teachers have to teach more than the maximum load, while other teachers might not teach any course, thereby resulting in infeasibility because of violation of the requirement  $c$  in Section 2.

Due to the infeasibility issue in the teacher assignment sub-problem above, we can try to improve it before continuing to the next step. We classify teachers into several groups: teachers who teach more than the maximum load allowed (Group 1), teachers who teach equal to or less than the maximum load allowed (Group 2) and teachers who are not allocated to any course (Group 3). The number of teachers in group  $s$  is represented as  $|Group_s|$ , for  $s = 1, 2$  or  $3$ . The aim is to replace teachers from Group 1 with other teachers until none of the teachers is in Group 1, and to allocate at least one course to the teachers in Group 3 as well. When we replace a teacher in Group 1, we have to consider all the courses taught by him/her and find another teacher who has the minimum load. A similar idea is used for allocating a teacher in Group 3. A course that is currently taught by a teacher who has the maximum load is chosen and would be allocated to a teacher from Group 3. These additional steps are conducted until none of the teachers is in Groups 1 and 3, or the total number of iterations reaches a predetermined maximum number of iterations.

The complexity of the above process is  $O(|I|^2|J|)$ . It is possible to redesign the procedure and make it more efficient, such as by ignoring the maximum and minimum load constraints of equation (4) in the *Checking procedure*. For this case, the complexity can be reduced to  $O(|I||J|)$ . However, the possibility of obtaining an infeasible solution might be higher. It would require some additional efforts, such as increasing the number of iterations or increasing the computational time in the improvement phase, in order to achieve better solutions. Also, for special cases such as when the number of sections for each course is set to 1, the complexity will be automatically reduced to  $O(|I||J|)$ .

For the initial allocation of time periods, each teacher has the time periods sorted based on his/her preferences. The number of time periods allocated to each teacher is dependent on the number of courses and course sections allocated during course assignment. The idea of the proposed algorithm is almost similar to course assignment. We allocate the time periods to teachers based on their time preferences.

Initially, we choose the first time period with the highest preference as the starting time period. The *Checking procedure* has to be invoked in order to address the feasibility issues. A similar

situation with the teacher assignment sub-problem will be faced if until the last time period in the list, the course section has not been allocated yet. For this situation, we have to relax some constraints and try to find other time periods. Due to the relaxed constraints, it turns out that some courses might be conducted more than once on the same day, or some time periods have the number of course sections taught that exceed the capacity, or some teachers do not have evenly spread out schedules. These courses, time periods and teachers are kept in Excess lists 1, 2 and 3, respectively. The number of members in an Excess List  $e$  is represented as  $|EL_e|$ , for  $e = 1, 2$  or  $3$ .

In order to deal with these infeasibility issues, the solution could be improved before continuing to the next phase. This is achieved by reallocating some courses and course sections to new time periods in such a way that all excess lists become empty. It is important to note that an infeasible initial solution might still occur. However, with the additional steps, the chances of infeasibility would be less. The worst case time complexity of this course scheduling process is  $O(|I||J||K||L||M|)$ . Again, this time complexity can be reduced through efficient procedures or for some special cases.

After performing the entire processes described above, the course and time objective function values are then calculated, before adding them together to obtain the total objective function value. The details of these calculations are explained next:

$$\begin{aligned} & \text{Course objective function value} = \\ & \sum_{i \in I} \sum_{j \in J} PC_{ij} \times P_{ij} + \left[ |Group_1| + |Group_3| \right] \times PENALTY1 \end{aligned} \tag{21}$$

$$\begin{aligned} & \text{Time objective function value} = \\ & \sum_{i \in I} \sum_{j \in J} \sum_{k \in K} \sum_{l \in L} \sum_{m \in M} PT_{ilm} \times X_{ijklm} + \left[ |EL_1| + |EL_2| + |EL_3| \right] \times PENALTY2 \end{aligned} \tag{22}$$

$$\begin{aligned} & \text{Total objective function value} \\ & = \text{course objective function value} + \text{time objective function value} \end{aligned} \tag{23}$$

Here, the terms PENALTY1 and PENALTY2 reflect the penalty values for the violations of the respective requirements listed in Section 2.

### 4.3 Improvement Phase

In this phase, after an initial solution is obtained from the previous phase, two operations are performed in order to seek further improvement. These two operations are re-allocation of teachers to courses and course sections, followed by re-scheduling of these changes into days and time periods. The initial solution is treated as a starting solution for the improvement phase.

All the operations in this phase explore the possible neighborhood to select the next movement. The first operation is started by choosing course  $j$  randomly followed by finding another teacher without violating the maximum load constraint. Two possible alternatives are considered: the new

teacher will be added to the list of teachers who teach course  $j$  or the new teacher will replace the teacher who has been allocated to course  $j$ . If the number of teachers who teach course  $j$  is still less than the maximum number of teachers allowed,  $UT_j$ , the two alternatives are chosen randomly. Otherwise, only the last alternative can be chosen.

The second operation is to schedule the changes in teacher assignment. The new teacher is allocated to the previous day and time periods scheduled for the previous teacher, if possible. We have to make sure that at these time periods, the new teacher has not been scheduled yet and he has an evenly spread out schedule as well. Otherwise, we have to find a new set of days and time periods by considering some constraints, such as the length of course should not exceed the last time period on a day.

If these two operations can give a better objective function value, we will update the starting solution. Otherwise, we return to the previous starting solution. These two operations are performed until the total number of iterations reaches a predetermined maximum number of iterations. Finally, the solution of the problem is the best solution obtained so far. In general, the time complexity of the neighborhood exploration is  $O(|I||J||K||L||M|)$ .

Though the entire heuristic described here is a simple improvement heuristic, it is worth noting that the procedures in the improvement phase allow for various types of modifications, such as incorporating metaheuristic procedures to the improvement phase. Such suitable hybridization of the heuristic methods may exhibit significantly better performance in terms of solution quality and the time to obtain the solutions, as noted by Purchinger and Raidl [22].

## 5 Computational Results

To evaluate the performance of the improvement heuristic, we compare the solutions obtained by it with the solutions obtained by solving the integer programming model. The data sets mentioned in Section 3 were used again in our computational experiments on the heuristic. The entire heuristic was coded in C++ and tested on the Intel Pentium IV 2.6 GHz PC with 512 MB RAM under the Microsoft Windows XP Operating System. The parameters of the proposed algorithms are chosen experimentally to ensure a compromise between the computational time and the solution quality. The values of the parameters used in the computational study are summarized as follows: the number of iterations in Construction Phase (teacher assignment problem) =  $40|I|$ , the number of iterations in Construction Phase (course scheduling problem) =  $20|J|$ , the number of iterations in Improvement Phase =  $|I||J||L||M|$ , and PENALTY1, PENALTY2 = 1,000.

Table 6 and 7 summarize the results obtained from the proposed heuristic for Group I and Group II data sets, respectively. A comparison of the objective function values and computational times (in seconds) obtained by the heuristic and the integer programming model was also presented. The percentage of solution deviation from best known/optimal objective function value is defined by  $Pct = (best\ known/optimal\ objective\ function\ value - objective\ function\ value\ obtained\ by\ the\ heuristic) / (best\ known/optimal\ objective\ function\ value) \times 100$ .

**Table 6.** The comparison of the heuristic results and the optimal solutions (Group I data sets)

Data set	Solution obtained by OPL		Heuristic		Average
	Average	Average CPU	Average	Average CPU	<i>Pct</i> of
	objective	time (in	objective	time	objective
	function	seconds)	function	(in seconds)	function
	value		value		value
5×5_1	1,052	2.03	924	0.01	12.17
5×5_2	1,216	2.17	988	0.01	18.75
10×10_1	2,184	18.47	1,896	0.03	13.19
10×10_2	2,712	31.79	2,106	0.13	22.35
15×15_1	3,170	136.93	2,818	0.11	11.10
15×15_2	4,166	431.94	3,050	0.43	26.79
20×20_1	4,368	337.98	3,956	0.27	9.43
20×20_2	5,774	4,212.92	4,100	1.14	28.99

**Table 7.** The comparison of the heuristic results and the optimal solutions (Group II data sets)

Data set	Solution obtained by OPL		Heuristic		Average
	Average	Average CPU	Average	Average CPU	<i>Pct</i> of
	objective	time (in	objective	time	objective
	function	seconds)	function	(in seconds)	function
	value		value		value
10×20_1	7,766	1,183.11	6,228	0.36	19.80
10×20_2	7,934	1,273.15	6,394	0.29	19.41
20×30_1	10,875	9,746.36	8,186	2.83	24.73
20×30_2	13,468	36,019.99	10,246	4.33	23.92
20×40_1	-	-	10,340	5.76	-
20×40_2	-	-	12,880	11.90	-
30×60_1	-	-	16,064	69.13	-
30×60_2	-	-	18,962	78.58	-

We observe that the proposed heuristic can yield good solutions with the percentage of solution deviation from the best known/optimal solutions being less than 29%. We also notice that the proposed heuristic is able to find the solutions within a reasonable amount of computational time. Some problems that were unsolved by the ILOG OPL Studio software can also be easily solved by the proposed heuristic, as can be seen in data sets such as that of 20×40\_1, 20×40\_2, 30×60\_1, and 30×60\_2.

**Table 8.** Load distribution of teachers

Data Set	Maximum load	Average percentage of teachers having the following load				Load variance
		1	2	3	4	
5×5_1	1	100%	0%	0%	0%	0.00
5×5_2	2	72%	28%	0%	0%	0.16
10×10_1	1	100%	0%	0%	0%	0.00
10×10_2	2	41.33%	25.33%	0%	0%	0.23
15×15_1	1	100%	0%	0%	0%	0.00
15×15_2	2	64%	36%	0%	0%	0.22
20×20_1	1	100%	0%	0%	0%	0.00
20×20_2	2	74%	26%	0%	0%	0.18
10×20_1	4	8%	30%	34%	28%	0.84
10×20_2	4	6%	20%	30%	44%	0.85
20×30_1	3	24%	46%	30%	0%	0.53
20×30_2	3	20%	39%	41%	0%	0.56
20×40_1	4	13%	27%	35%	25%	0.95
20×40_2	4	14%	15%	28%	43%	1.13
30×60_1	4	9.33%	30.67%	34%	26%	0.88
30×60_2	4	6.67%	21.33%	35.33%	36.67%	0.84

Another observation of interest is on the load distribution of the teachers with respect to the maximum load that we specify (see Table 8). For Group I data sets, we notice that the proposed heuristic distributes the load evenly to teachers. The load variances obtained are reasonably small. However, for Group II data sets, the load variances are found to be large with only a small percentage of teachers having very small load.

## 6 Conclusions

We have proposed a new mathematical programming model for a timetabling problem that combines two sub-problems, teacher assignment and course scheduling, simultaneously. This model allows the possibility that courses could be taught by more than one teacher and each course might consist of more than one sections.

An improvement heuristic was proposed for solving the timetabling problem that cannot be easily solved by commercial software. The proposed heuristic solves these sub-problems iteratively. The computational results show that the proposed heuristic yields good solutions within reasonable computational time. The results obtained from the proposed heuristic were also compared against solutions of an integer programming approach.

As possible future research directions, we can look into ways of improving the proposed heuristic to obtain solutions of better quality even when the problem size is very large. This would include the hybridization of the proposed heuristic with other types of heuristics as mentioned previously. Since each university has different characteristics and requirements, the mathematical programming model and the proposed heuristic can also be extended to adapt to these distinct characteristics and requirements.

## References

1. Wren, A.: Scheduling, timetabling and rostering – A special relationship? In E. Burke and P. Ross (ed), Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science **1153**, Springer-Verlag, New York (1996) 46-75
2. Carter, M. W., Laporte, G.: Recent developments in practical course timetabling. In E. Burke and M. Carter (ed), Practice and Theory of Automated Timetabling II, Lecture Notes in Computer Science **1408**, Springer-Verlag, New York (1998) 3-19
3. Andrew, G. M., Collins, R.: Matching faculty to courses. *College and University* **46** (1971) 83-89
4. Breslaw, J. A.: A linear programming solution to the faculty assignment problem. *Socio-Economic Planning Sciences* **10** (1976) 227-230
5. Harwood, G. B., Lawless, R. W.: Optimizing faculty teaching schedules. *Decision Science* **6** (1975) 513-524
6. Schniederjans, M. J., Kim, G. C.: A goal programming model to optimize departmental preference in course assignments. *Computers and Operations Research* **14** (1987) 87-96
7. Tillett, P. I.: An operations research approach to the assignment of teachers to courses. *Socio-Economic Planning Sciences* **9** (1975) 101-104
8. Wang, Y. Z.: An application of genetic algorithm methods for teacher assignment problems. *Expert Systems with Applications* **22** (2002) 295-302
9. Yu, E., Sung, K. S.: A genetic algorithm for a university weekly courses timetabling problem. *International Transactions in Operational Research* **9** (2002) 703-717
10. Abramson, D.: Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management Science* **37** (1991) 98-113
11. Abramson, D., Krishnamoorthy, M., Dang, H.: Simulated annealing cooling schedules for the school timetabling problem. *Asia-Pacific Journal of Operational Research* **16** (1999) 1-22
12. Hertz, A.: Finding a feasible course schedule using tabu search. *Discrete Applied Mathematics* **35** (1992) 255-270
13. Badri, M. A.: A two-stage multiobjective scheduling model for [faculty-course-time] assignments. *European Journal of Operational Research* **94** (1996) 16-28
14. Daskalaki, S., Birbas, T., Housos, E.: An integer programming formulation for a case study in university timetabling. *European Journal of Operational Research* **153** (2004) 117-135

15. Gunawan, A., Ng, K.M., Poh, K.L.: A Mathematical Programming Model for a Timetabling Problem. *Proceedings of the International Conference on Scientific Computing, June 2006, Nevada, USA* (2006)
16. Costa, D.: A tabu search algorithm for computing an operational timetable. *European Journal of Operational Research* **76** (1994) 98-110
17. Eiselt, H.A., Laporte, G.: Combinatorial optimization problems with soft and hard requirements. *The Journal of the Operational Research Society* **38** (1987) 785-795
18. Aubin, J., Ferland, J.A.: A large scale timetabling problem. *Computers and Operations Research* **16** (1989) 67-77
19. Loo, E.H., Goh, T.N., Ong, H.L.: A heuristic approach to scheduling university timetables. *Computers and Education* **10** (1985) 379-388
20. Wright, M.: School timetabling using heuristic search. *The Journal of the Operational Research Society* **47** (1996) 347-357
21. Even, S., Itai, A., Shamir, A.: On the complexity of timetable and multicommodity flow problems. *SIAM Journal of Computing* **5** (1976) 691-703
22. Puchinger, J. Raidl, G.R.: Combining metaheuristics and exact algorithms in combinatorial optimization: a survey and classification. In J. Mira and J.R. Alvarez (ed), *IWINAC 2005, Lecture Notes in Computer Science* **3562**, Springer-Verlag, New York (2005) 41-53

## **A Comparison between Algorithms VNS with PSO and VNS without PSO for Job-shop Scheduling Problems**

Pisut Pongchairerks\* and Voratas Kachitvichyanukul

Industrial Engineering and Management, School of Engineering and Technology,  
Asian Institute of Technology, Pathumthani 12120, Thailand  
{st101122,voratas}@ait.ac.th

**Abstract.** Based on literature, one of the efficient meta-heuristic algorithms for the classical job-shop scheduling problem in terms of solution quality is the hybrid algorithm between Particle Swarm Optimization (PSO) algorithm and Variable Neighborhood Search (VNS) algorithm, namely  $PSO_{VNS}$ . This paper demonstrates that the performance of  $PSO_{VNS}$  is contributed only by VNS. This is proven by comparing the results from  $PSO_{VNS}$  and those of VNS in the same set of benchmark instances. The Comparison shows that a much simpler VNS procedure without PSO can produce same solution quality with much faster computation time.

**Keywords:** job-shop scheduling, particle swarm optimization, variable neighborhood search, makespan.

### **1 Introduction**

Scheduling is a decision making process that is important in most manufacturing and service industries. It is used in many areas such as production planning, logistic and transportation, information processing, and communication. Many scheduling problems are complex and cannot be solved to optimality in polynomial time. Among those problems, the classical job-shop scheduling problem (JSP) is the most well-known and yet still be a challenging one for scholar to solve. JSP can be stated as follows; there is a set of jobs and a set of machines. Each job consists of a chain of operations, each of which must be processed during an uninterrupted period of a given length on a given machine. Each machine can handle at most one operation at a time. With the above givens, JSP moves on the objective which requires findings a feasible schedule in order to mini-

---

\* Corresponding Author. Tel.: +66 (0)2-524-7413, Email: st101122@ait.ac.th.

mize the last completion time of the given jobs. The last completion time will be called makespan throughout this paper.

JSP is known to be extremely hard. Hence, in order to find good quality solutions within reasonable computation times, several meta-heuristic algorithms have been applied to solve the problem such as Tabu Search algorithms [1, 2], Simulated Annealing algorithms [3, 4], Genetic algorithms [5, 6], Ant Colony Optimization algorithm [7], and the hybridized algorithm of Particle Swarm Optimization (PSO) and Variable Neighborhood Search (VNS) named  $PSO_{VNS}$  [8]. Among these algorithms,  $PSO_{VNS}$  is one of the efficient methods of JSP, because it shows very good solutions in several benchmark instances. On the other hand,  $PSO_{VNS}$  is complicated and may consume a long computation time. Moreover, the advantages of the hybridization of PSO and VNS in  $PSO_{VNS}$  have not been identified and the contribution of each algorithm (PSO and VNS) to the performance of  $PSO_{VNS}$  is yet unknown.

This research aims to evaluate the partitioned algorithms, PSO and VNS, and study each algorithm's contribution to the performance of  $PSO_{VNS}$  algorithm. In order to study the mentioned contributions, this research compares the results generated by the algorithms  $PSO_{VNS}$ , VNS (a slightly modified algorithm of VNS where VNS is a part of  $PSO_{VNS}$ ), and  $PSO_{VNS}$  without VNS in the same set of benchmark instances. The comparison will be done in terms of solution quality and computation time.

The remainder of this paper is divided into four sections. Section 2 talks about this research' background where JSP and some algorithms were discussed. All of which aims at giving the readers a clearer illustration as to the understanding of this proposed research. Section 3 introduces VNS algorithm which is a slightly modified algorithm of VNS where VNS is a part of  $PSO_{VNS}$ . Section 4 shows the numerical results comparing the three algorithms;  $PSO_{VNS}$ ,  $PSO_{VNS}$  without VNS, and VNS mentioned in Section 3. Finally, Section 5 concludes the findings of the research.

## 2 Preliminaries

In order to provide a basic and common understanding regarding the past development of the scheduling problem as well as the algorithms developed prior to the introduction of this paper's proposed algorithm seen in the later sections, this section presents a description of the job-shop scheduling problem (JSP), the standard PSO, the standard VNS, and  $PSO_{VNS}$  algorithm as follows:

### 2.1 JSP Description

The classical job-shop scheduling problem or JSP [9] is stated as follows; there are  $n$  jobs  $J_1, J_2, J_3, \dots, J_i, \dots, J_n$  and  $m$  machines  $M_1, M_2, M_3, \dots, M_m$ . Each job  $J_i$  consists of a chain of operations  $O_{i1}, O_{i2}, O_{i3}, \dots, O_{ij}, \dots, O_{im}$  which have to be processed in this order. The operation  $O_{ij}$  has to be processed by a predefined machine during  $p_{ij}$  time units without preemption. To undergo the process, these following constraints have to be satisfied. (1) Each machine cannot simultaneously

handle two or more jobs at any given time period. (2) At one given time period, each job cannot be processed on more than one machine. For the above statement, the problem requires finding a feasible allocation of the operations to time intervals on the given machines such that makespan is minimized. Allocation of all operations to time intervals is likewise known as schedule.

Three well-known classes of job-shop schedules are defined below:

**Semi-active schedule:** a feasible schedule such that no operation can be started earlier without altering the processing sequence.

**Active schedule:** a semi-active schedule such that no operation can be started earlier without delaying some other operation.

**Non-delay schedule:** an active schedule such that no machine is kept idle when it can start processing some operation.

With respect to solution space, the set of all active schedules may be better than that of semi-active schedules because the set of all active schedules is much smaller in number than the set of all semi-active schedules; moreover, the optimal schedule is always an active schedule. In comparison between the set of all active schedules and that of all non-delay schedules, the set of all non-delay schedules is much smaller in number than the set of all active schedules, but unfortunately, the set of all non-delay schedules may not contain an optimal schedule.

## 2.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population-based stochastic search algorithm. The standard PSO algorithm was originally developed by Kennedy and Eberhart [10] and later modified by Shi and Eberhart [11] with the introduction of an inertia weight ( $w$ ), and has become the most popular version of PSO ever used. The underlying motivation for the development of PSO algorithm is the social behavior of animals such as fish schooling and bird flocking. The concept of the implementation of PSO is based on social sharing of information among individuals in a population. The PSO algorithm starts with a population of particles. Each particle consists of a potential solution (called position in PSO) and a velocity. Each particle moves around the  $D$ -dimensional search space with a velocity which is dynamically adjusted based on the moving experience of its own and that of its neighbors. To describe PSO algorithm, the following notation and terminology are defined:

**Particle:** an element in a population. It has a position and a velocity; it knows its position and also the objective function value of this position; it also remembers its personal best position as well as its neighbor's best position and the objective function value.

**Population:** a set of  $K$  particles with corresponding indices  $\{1, \dots, r, \dots, K\}$  in the PSO system such that  $r \in$  positive integers, also known as swarm.

**Position:** Let  $X_i(t)$  denote the position of the  $i^{\text{th}}$  particle at the  $t^{\text{th}}$  iteration and is represented by a  $D$  dimensional point  $X_i(t) = (x_{i1}(t), \dots, x_{id}(t), \dots, x_{iD}(t))$ , where  $x_{id}(t)$  is the position value of the  $d^{\text{th}}$  dimension of the  $i^{\text{th}}$  particle at the  $t^{\text{th}}$  iteration. Each position  $X_i(t)$  may directly or indirectly represent a solution of a specific problem.

**Fitness value:**  $f(X_i(t))$  represents the objective function value, called fitness value, of the solution of a specific problem decoded from the position  $X_i(t)$ .

**Velocity:**  $V_i(t)$  denotes the velocity of the  $i^{\text{th}}$  particle at the  $t^{\text{th}}$  iteration and is represented by a vector of  $D$  dimensions as  $V_i(t) = (v_{i1}(t), \dots, v_{id}(t), \dots, v_{iD}(t))$ , where  $v_{id}(t)$  is the velocity value of the  $d^{\text{th}}$  dimension of the  $i^{\text{th}}$  particle at the  $t^{\text{th}}$  iteration.

**Maximum Velocity:**  $V_{\max}$  defines a boundary of the velocities. Each  $v_{id}(t)$  cannot be outside the range  $[-V_{\max}, V_{\max}]$ .

**Inertia weight:**  $w(t)$  is a parameter employed to control the impact of the previous velocities of  $K$  particles on the current velocities.

**Constriction coefficient:**  $\delta$  is a parameter that enables control over the convergence properties of a particle swarm system.

**Personal best position:**  $P_i$  denotes the position found by the  $i^{\text{th}}$  particle that contains the best fitness value and is represented by  $P_i = (p_{i1}, \dots, p_{id}, \dots, p_{iD})$ . In a minimization problem,  $P_i = X_i(1)$  at the 1<sup>st</sup> iteration. After 1<sup>st</sup> iteration ( $t > 1$ ),  $P_i = X_i(t)$  if  $f(X_i(t)) < f(P_i)$ . Otherwise, the personal best position stays at  $P_i$ .

**Global best position:**  $P_g$  is a position of a particle with the best fitness value in a population, where  $P_g = (p_{g1}, \dots, p_{gd}, \dots, p_{gD})$ . In a minimization problem, let  $P_B(t)$  be the best particle position with the best fitness value where

$$f(P_B(t)) = \min_{i=1, \dots, r, \dots, K} f(P_i)$$

At the 1<sup>st</sup> iteration,  $P_g = P_B(1)$ . After the 1<sup>st</sup> iteration ( $t > 1$ ),  $P_g = P_B(t)$  if  $f(P_B(t)) < f(P_g)$ . Otherwise, the personal best position stays at  $P_g$ .

**Acceleration constant:** the acceleration constant tied in each best position (e.g., personal best position and global best position) is a parameter that impacts the velocity value. It, therefore, has a strength to control the velocity value when the constant's value is adjusted.

$c_p$  = the acceleration constant of the personal best position, and

$c_g$  = the acceleration constant of the global best position.

Below are the steps for the standard PSO algorithm:

- (1) Set current iteration  $t = 1$ . Initialize the random positions and velocities of  $K$  particles in a population in a  $D$ -dimensional search space;
- (2) For each particle, decode its position into a solution of a specific problem and evaluate the objective function value of the solution as the fitness value of the position. The decoding procedure is specific to each problem;
- (3) Update the personal best positions;
- (4) Update the global best position;
- (5) For each particle, its velocities and positions are updated according to equations (1) and (2); where,  $rnd$  and  $Rnd$  represent uniform random numbers over the interval  $[0, 1]$ ;

$$v_{id}(t+1) = \left. \begin{aligned} & \delta(w(t)v_{id}(t) + c_p \text{rnd}(p_{id} - x_{id}(t)) + c_g \text{Rnd}(p_{gd} - x_{id}(t))) \\ & \left. \begin{cases} -V_{\max} & \text{if } v_{id}(t+1) \leq -V_{\max} \\ V_{\max} & \text{if } v_{id}(t+1) \geq V_{\max} \end{cases} \right\} \quad (1) \end{aligned}$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \quad (2)$$

- (6) If the stopping criterion is met, stop. Otherwise, set  $t = t + 1$  and repeat the process from step (2).

### 2.3 Variable Neighborhood Search

Variable Neighborhood Search [12, 13], also called VNS, is a meta-heuristic approach for combinatorial optimization which exploits systematically the idea of neighborhood change both in the descent to local optima and in the escape from the spaces which contain them. The VNS development is motivated on account of three following observations; (1) a local optimum with respect to one neighborhood structure may not be as necessary for another; (2) a global optimum is a local optimum with respect to all possible neighborhood structures; and (3) local optima with respect to one or several neighborhoods are relatively close to each other. Concerning the observations, VNS was developed based on the idea of neighborhood change in the search [12, 13], and the implementation procedure of the general VNS can be seen in Figure 1.

**Initialization.** Select the set of neighborhood structures  $N_k, k = 1, \dots, k_{\max}$ , that will be used in the shaking step, and the set of neighborhood structures  $N_l, l = 1, \dots, l_{\max}$ , that will be used in the local search; find an initial solution  $x$ ; choose a stopping criterion;

**Repeat** the following steps until the stopping criterion is met:

- (1) Set  $k = 1$ ;
- (2) Repeat the following steps until  $k = k_{\max}$ ;
  - (2.1) Shaking. Generate a point  $x'$  randomly from the  $k^{\text{th}}$  neighborhood  $N_k(x)$  of  $x$ ;
  - (2.2) Local search
    - (2.2.1) Set  $l = 1$ ;
    - (2.2.2) Repeat the following steps until  $l = l_{\max}$ ;
      - Find the best neighbor  $x''$  of  $x'$  in  $N_l(x')$ ;
      - If  $f(x'') < f(x')$ , set  $x' = x''$  and  $l = 1$ ; otherwise  $l = l + 1$ ;
    - (2.2.3) If the local optimum is better than the incumbent,  $x = x''$  and  $k = 1$ ; otherwise,  $k = k + 1$ ;

Fig. 1. The steps of general VNS

## 2.4 PSO<sub>VNS</sub> on JSP

Based on literature, PSO<sub>VNS</sub> algorithm has been successfully applied to several types of scheduling applications such as the single machine total weighted tardiness problems [14], permutation flow-shop scheduling problems [15], and job-shop scheduling problems [8]. PSO<sub>VNS</sub> is a hybrid algorithm between PSO and VNS because it is the standard PSO [11] that applies VNS as local search procedure to improve the global best position of PSO at every iteration. PSO<sub>VNS</sub> [8] as an application of JSP will be called JSP-PSO<sub>VNS</sub> throughout this paper. JSP-PSO<sub>VNS</sub> requires a mapping procedure [8] to decode a particle position into a semi-active schedule. A semi-active schedule is a solution generated from this algorithm.

The important steps of implementing JSP-PSO<sub>VNS</sub> (i.e., solution decoding and population initialization) and the parameter setting of JSP-PSO<sub>VNS</sub> are presented below:

### 2.4.1 Solution Representation

In order to decode a particle position into a semi-active schedule, the particle position is translated into an operation-based permutation by using Smallest Position Value (SPV) rule [8], and this operation-based permutation represents a semi-active schedule with the procedure of Cheng et al. [16].

### 2.4.2 Initial Population

Initially,  $x_{id}(1)$  and  $v_{id}(1)$  ( $i = 1, \dots, r, \dots, K$ ;  $d = 1, \dots, r, \dots, mn$ ;  $r \in$  positive integers) is generated randomly in the range  $[-4, 4]$ . Remind that  $m =$  the number of given machines and  $n =$  the number of given jobs.

### 2.4.3 Parameter Setting

The values of the parameters used in [8] are presented as follows:

- Population size  $K = mn$ ;
- Acceleration constants  $c_p = c_g = 2$ ;
- Maximum velocity  $V_{\max} = 4$ ;
- Constriction coefficient  $\delta = 0.7$ ;
- Inertia weight  $w(1) = 0.9$  and  $w(t+1) = 0.975w(t)$ ; In addition, if  $w(t) < 0.4$ ,  $w(t) = 0.4$ .

## 3 VNS on JSP

The VNS algorithm proposed in this section is a slightly modified version of the VNS algorithm where VNS is a part of JSP-PSO<sub>VNS</sub> [8]. The mentioned modification is just that, at each iteration,

the step to transform the global best position of PSO part into the initial solution of VNS part and the step to transform the final solution of VNS part back into the global best position of PSO part are deleted, because these two steps are not necessary now. This proposed VNS algorithm is called JSP-VNS throughout this paper. All solutions generated from JSP-VNS are represented by operation-based permutations [16], where all of them are in a set of semi-active schedules.

### 3.1 Solution Representation

In an  $n$ -job/ $m$ -machine instance, an operation-based permutation, represented  $\pi = (\pi_1, \pi_2, \pi_3, \dots, \pi_{mn})$ , is a sequence of  $mn$  integer numbers; the permutation  $\pi$  contains the index of each given job for exactly  $m$  times. In order to decode an operation-based permutation  $\pi$  into a semi-active schedule, operations must be scheduled one at a time. An operation is schedulable if all those operations which precede it have already been scheduled. Since there are  $mn$  operations, the algorithm will iterate through  $mn$  stages. At stage  $t$ , let  $P_t$  = the partial schedule of the  $(t - 1)$  scheduled operation. The decoding procedure is presented as follows:

- (1) Set  $t = 1$  with  $P_1$  being null;
- (2) Choose an operation with the job index  $\pi_t$  and no predecessors; then,  $O^*$  will represent this chosen operation;
- (3) Add the operation  $O^*$  to  $P_t$  with the earliest time that this operation can be started, and set  $t = t + 1$ ;
- (4) If  $t < mn$ , repeat the process from step (2). Otherwise, stop.

In order to illustrate solution representation, an example of translating an operation-based permutation  $\pi = (1, 1, 2, 3, 2, 1, 3, 3, 2)$  into a 3-job/3-machine job-shop schedule is presented here. The details of instance in this example are shown in Table 1, thus the Gantt chart for solution decoded from the defined permutation is presented in Figure 2.

**Table 1.** Example of a 3-job/3-machine problem

Processing Time				Machine Sequence			
Job	Operations			Job	Operations		
	1	2	3		1	2	3
$J_1$	2	1	2	$J_1$	$M_1$	$M_3$	$M_2$
$J_2$	3	3	2	$J_2$	$M_2$	$M_3$	$M_1$
$J_3$	3	1	3	$J_3$	$M_1$	$M_2$	$M_3$

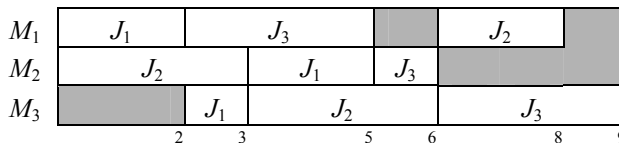


Fig. 2. Gantt chart for the solution decoded from the defined permutation

### 3.2 Algorithm

There are two main stages in this algorithm, namely local search stage and shaking stage. The local search stage uses two neighborhood structures, the interchange structure and insertion structure. In order to generate a neighbor of an operation-based permutation  $\pi$  (i.e., solution representation) of each structure mentioned above, the following orders are presented as follows:

**Order interchange:** SWAP( $\pi$ ) means interchanging the two number indices  $u$  and  $v$  of the operation-based permutation  $\pi$ . Let  $u$  and  $v$  be random numbers on the integer interval  $[1, 2, 3, \dots, mn]$ , where  $u \neq v$ . The new permutation generated from this order interchange structure is a neighbor of  $\pi$ .

**Order insertion:** INSERT( $\pi$ ) means removing the number from the  $u^{th}$  index of  $\pi$  and then insert this number in the  $v^{th}$  index. Let  $u$  and  $v$  be random numbers on the integer interval  $[1, 2, 3, \dots, mn]$ , where  $u \neq v$ . The new permutation generated from the order insertion structure is a neighbor of  $\pi$ .

In the shaking stage, JSP-VNS uses only one neighborhood structure, where a neighbor of  $\pi$  in this neighborhood structure can be generated by using order insertion and order interchange, twice respectively. This stage is denoted as SWAP(SWAP(INSERT(INSERT( $\pi$ ))))). The pseudo-code of JSP-VNS is given in Figure 3. Let  $f(\pi)$  be the makespan of the schedule decoded from permutation  $\pi$ .

```

Randomly generate an initial solution  $\pi^0$ ;
Iteration  $t = 0$ ;
do {
     $\pi = \text{SWAP}(\text{SWAP}(\text{INSERT}(\text{INSERT}(\pi^0))))$ ;
    loop = 0;
    do {
        lcount = 0;
        lmax_method = 2;
        do {
            if (lcount == 0) {  $\pi^1 = \text{SWAP}(\pi)$ ; }
            if (lcount == 1) {  $\pi^1 = \text{INSERT}(\pi)$ ; }
            if ( $f(\pi^1) \leq f(\pi)$ ) { lcount = 0;  $\pi = \pi^1$ ; }
            else { lcount++; }
        } while (lcount < lmax_method);
        loop++;
    } while (loop <  $mn(mn - 1)$ );
    if ( $f(\pi) \leq f(\pi^0)$ ) {  $\pi^0 = \pi$ ; }
    t++;
} while (Stopping criterion is not met);

```

Fig. 3. Pseudo-code of JSP-VNS

## 4 Numerical Experiments

In order to evaluate the contribution of each partitioned algorithm (i.e., PSO and VNS) to the performance of  $\text{PSO}_{\text{VNS}}$ , this section compares the results generated by the algorithms  $\text{JSP-PSO}_{\text{VNS}}$ , JSP-PSO, and JSP-VNS. These three algorithms are presented in brief below:

- $\text{JSP-PSO}_{\text{VNS}}$  is introduced by [8] and also mentioned in Section 2.4
- JSP-VNS is the VNS algorithm slightly modified from the VNS part of  $\text{JSP-PSO}_{\text{VNS}}$ . It is mentioned in Section 3
- JSP-PSO is the  $\text{JSP-PSO}_{\text{VNS}}$  without VNS part

These three algorithms are tested on a same environment in order to make a fair comparison. This environment is presented below:

- Each algorithm is coded in C# and executed on a Pentium M processor 1.60 GHz with 256 MB of RAM
- Each algorithm is evaluated through the set of test instances given from two sources of standard JSP test problems: Lawrence [17] instances la01 to la40, and Taillard [18] instances ta11 & ta12
- On each test instance, each algorithm will stop when any of three following conditions is met; (1) the optimal solution is found, (2) the best found solution is not improved within 250 iterations, and (3) the 1,000<sup>th</sup> iteration is reached
- On each test instance, each algorithm is repeated for five runs in order to collect the following results; the best makespan found, the average makespan found, and the average computation time. Moreover, the best solution values from Nowicki and Smutnicki [2] are also used to compare, because Tabu Search algorithm of [2] is very well-known and its results are usually used on comparison in many published papers.

The results of the comparison are shown in Table 2. In this table, information in each column can be defined as follows:

- Column "Instance" shows the name of test instance
- Column "Size" presents the problem size  $m \times n$  where  $m$  = number of machines and  $n$  = number of jobs
- Column "Opt (UB)" shows the optimal solution value in the specified instance. Otherwise, it shows the upper bound of the optimal solution value in parenthesis if the optimal solution has been unknown.
- $Z_{\text{NS}}$  presents the best solution value found by [2] in the specified instance
- For each of three algorithms (i.e.,  $\text{JSP-PSO}_{\text{VNS}}$ , JSP-VNS, and JSP-PSO),  $Z$  presents the best found solution value over five runs, Avg presents the average solution value found over five runs, and Time is the average computation time (sec.) for five runs.

According to information from Table 2, in terms of solution quality, the Tabu Search from [2],  $\text{JSP-PSO}_{\text{VNS}}$ , and JSP-VNS generate very good solutions on most instances, while JSP-PSO can generate good solutions only on small instances. In terms of computation time, JSP-PSO consumes lowest computation times in comparison; however, its solutions are very low quality in

terms of makespan. Compare JSP-VNS to JSP-PSO<sub>VNS</sub>, JSP-VNS consumes shorter computation times on most instances.

Table 3 shows a summary of the results from Table 2. This table show information as follows; the problem type refers to a set of the instances with a same size ( $m \times n$ ),  $\Delta$  refers to the average deviation (%) between the optimal solution value (or its upper bound value) and the best solution value found by each algorithm, Time refers to the average computation time in second unit of each algorithm in the defined problem type. Table 3 points out that JSP-VNS outperforms other algorithms in terms of solution quality. The comparison of JSP-VNS to other algorithms is presented below:

- Compare JSP-VNS to Tabu Search [2], it is shown that there are two problem types where JSP-VNS performs better, five problem type where both performs equally, and only one problem type where Tabu Search [2] performs better
- Compare JSP-VNS to JSP-PSO<sub>VNS</sub>, there are three problem types where JSP-VNS performs better, and both perform equally on other problem types
- Compare JSP-VNS to JSP-PSO, JSP-VNS performs much better on all problem types.

**Table 2.** Results from the benchmark instances of [17, 18]

Instance	Size $m \times n$	Opt (UB)	$Z_{NS}$	JSP-PSO <sub>VNS</sub>			JSP-VNS			JSP-PSO		
				Z	Avg	Time	Z	Avg	Time	Z	Avg	Time
la01	5x10	666	666*	666*	670	18	666*	666	0.3	666*	671	1
la02	5x10	655	655*	655*	656	17	655*	655	2	704	734	1
la03	5x10	597	597*	597*	603	81	597*	602	35	630	664	1
la04	5x10	590	590*	590*	597	54	590*	593	23	619	641	2
la05	5x10	593	593*	593*	593	0.4	593*	593	0.3	593*	593	0.2
la06	5x15	926	926*	926*	926	1	926*	926	1	926*	930	2
la07	5x15	890	890*	890*	891	51	890*	890	1	922	957	4
la08	5x15	863	863*	863*	863	1	863*	863	1	884	895	4
la09	5x15	951	951*	951*	951	1	951*	951	1	951*	971	3
la10	5x15	958	958*	958*	958	2	958*	958	1	958*	958	1
la11	5x20	1222	1222*	1222*	1222	4	1222*	1222	4	1222*	1233	8
la12	5x20	1039	1039*	1039*	1039	4	1039*	1039	4	1039*	1050	5
la13	5x20	1150	1150*	1150*	1150	4	1150*	1150	4	1150*	1155	5
la14	5x20	1292	1292*	1292*	1292	6	1292*	1292	4	1292*	1292	1
la15	5x20	1207	1207*	1207*	1207	3	1207*	1207	3	1305	1332	9
la16	10x10	945	945*	945*	945	294	945*	946	461	1047	1065	7
la17	10x10	784	784*	784*	784	125	784*	784	35	865	884	6
la18	10x10	848	848*	848*	854	196	848*	854	228	888	947	7
la19	10x10	842	842*	842*	849	563	842*	846	286	958	984	9
la20	10x10	902	902*	902*	905	421	902*	905	320	995	1053	7

Instance	Size $m \times n$	Opt (UB)	$Z_{NS}$	JSP-PSO <sub>VNS</sub>			JSP-VNS			JSP-PSO		
				Z	Avg	Time	Z	Avg	Time	Z	Avg	Time
la21	10x15	1046	1047	1046*	1052	3253	1047	1057	2749	1293	1308	27
la22	10x15	927	927*	927*	941	3068	927*	928	2037	1102	1169	26
la23	10x15	1032	1032*	1032*	1032	24	1032*	1032	10	1210	1232	28
la24	10x15	935	939	935*	942	3047	937	941	3558	1129	1156	26
la25	10x15	977	977*	984	988	2592	977*	981	2717	1190	1225	32
la26	10x20	1218	1218*	1218*	1218	109	1218*	1218	63	1453	1517	88
la27	10x20	1235	1236	1240	1259	7695	1235*	1252	5179	1556	1623	75
la28	10x20	1216	1216*	1216*	1216	4791	1216*	1216	1471	1443	1518	97
la29	10x20	1152	1160	1163	1174	11730	1163	1171	11100	1465	1520	70
la30	10x20	1355	1355*	1355*	1355	28	1355*	1355	39	1566	1630	77
la31	10x30	1784	1784*	1784*	1784	153	1784*	1784	125	1987	2089	264
la32	10x30	1850	1850*	1850*	1850	142	1850*	1850	117	2143	2174	304
la33	10x30	1719	1719*	1719*	1719	146	1719*	1719	121	1919	2017	309
la34	10x30	1721	1721*	1721*	1721	139	1721*	1721	115	2022	2066	260
la35	10x30	1888	1888*	1888*	1888	160	1888*	1888	124	2152	2258	237
la36	15x15	1268	1268*	1268*	1271	13928	1268*	1269	8520	1560	1612	86
la37	15x15	1397	1407	1397*	1410	8785	1397*	1407	7084	1670	1742	108
la38	15x15	1196	1196*	1201	1205	14155	1201	1207	9599	1495	1551	99
la39	15x15	1233	1233*	1233*	1236	9737	1233*	1237	7802	1543	1619	83
la40	15x15	1222	1229	1224	1226	11678	1224	1227	12398	1540	1576	119
ta11	15x20	(1364)	—	1386	1396	41628	1380	1394	28701	1826	1863	223
ta12	15x20	(1367)	1377	1377	1379	32317	1377	1385	31288	1814	1899	196

Note: (1)  $Z_{NS}$  refers to results from [2].

(2) A solution is marked \* if it is found to be the optimal solution value.

**Table 3.** Summary from results of Table 2

Problem type	$\Delta_{NS}$ (%)	JSP-PSO <sub>VNS</sub>		JSP-VNS		JSP-PSO	
		$\Delta$ (%)	Time	$\Delta$ (%)	Time	$\Delta$ (%)	Time
5 × 10	0.00	0.00	34	0.00	12	3.58	1
5 × 15	0.00	0.00	11	0.00	1	1.21	3
5 × 20	0.00	0.00	4	0.00	4	1.62	6
10 × 10	0.00	0.00	320	0.00	266	9.99	7
10 × 15	0.10	0.14	2397	0.06	2214	20.46	28
10 × 20	0.16	0.27	4871	0.19	3571	21.34	81
10 × 30	0.00	0.00	148	0.00	120	14.06	275
15 × 15	0.26	0.12	11657	0.12	9081	23.75	99
15 × 20	—	1.17	36973	0.95	29995	33.29	210

Note:  $\Delta_{NS}$  refers to average deviation between the upper bound value and the best solution value found by the algorithm from [2].

## 5 Conclusions

This research compares the performance of JSP-PSO<sub>VNS</sub>, JSP-VNS, and JSP-PSO, where JSP-PSO and JSP-VNS are two partitioned algorithms from JSP-PSO<sub>VNS</sub>. The numerical results show that JSP-VNS outperforms other algorithms. Thus, JSP-VNS applied on JSP as proposed in this paper is more efficient than JSP-PSO<sub>VNS</sub>. Moreover, it is also interpreted that VNS shoulders a greater contribution to the performance of JSP-PSO<sub>VNS</sub>.

In addition, although JSP-PSO does not perform well in the comparison, it does not mean that PSO, when it stands alone, always performs poorly on JSP. For this, the following suggestions to improve the performance of PSO on JSP are presented; (1) that future papers are recommended to use one of the recent variants of PSO [19, 20, 21] which indicated a better performance than that of the standard PSO; and (2) that a smaller (than the set of semi-active schedules generated from the PSO<sub>VNS</sub> algorithm) solution space (e.g., a set of all active schedules, a set of all non-delay schedules, etc.) would exclude schedules with unnecessary delay times, hence giving a better solution quality in terms of makespan. On the same directions, even though JSP-VNS shows highest performances in comparison, the algorithm may be improved by using some improved neighborhood structures, and using a better solution space such as the set of all active schedules.

## References

1. Taillard, E.D.: Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing* **6** (1994) 108-117

2. Nowicki, E., Smutnicki, C.: A fast taboo search algorithm for the job-shop problem. *Management Science* **42** (1996) 797-813
3. Kolonko, M.: Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operational Research* **113** (1999) 123-136
4. Satake, T., Morikawa, K., Takahashi, K., Nakamura, N.: Simulated annealing approach for minimizing the makespan of the general job-shop. *International Journal of Production Economics* **60-61** (1999) 515-522
5. Bierwirth, C., Mattfeld, D.C.: Production scheduling with genetic algorithms. *Evolutionary Computation* **7** (1999) 1-17
6. Gonçalves, J.F., Mendes, J.M., Resende, M.G.C.: A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research* **167** (2005) 77-95
7. Blum, C., Sampels, M.: An ant colony optimization algorithm for shop scheduling problems. *Journal of Mathematical Modelling and Algorithms* **3** (2004) 285-308
8. Tasgetiren, M.F., Sevkli, M., Liang, Y.-C., Yenisey, M.M.: Particle swarm optimization and differential evolution algorithms for job shop scheduling problems. *International Journal of Operations Research special issue on Scheduling in Manufacturing* (2007) in press
9. French, S.: *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-shop*. John Wiley & Sons, New York (1982)
10. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. *International Conference on Neural Networks IV*, Piscataway, NJ (1995) 1942-1948
11. Shi, Y., Eberhart, R.C.: A modified particle swarm optimizer. *Proceedings of the IEEE International Conference on Evolutionary Computation*, Piscataway, NJ (1998) 69-73
12. Mladenović, N., Hansen, P.: Variable neighborhood search. *Computers and Operations Research* **24** (1997) 1097-1100
13. Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications. *European Journal of Operational Research* **130** (2001) 449-467
14. Tasgetiren, M.F., Sevkli, M., Liang, Y.-C., Gencyilmaz, G.: Particle swarm optimization algorithm for single machine total weighted tardiness problem. *Proceedings of the 2004 Congress on Evolutionary Computation*, Portland, Oregon (2004) 1412-1419
15. Tasgetiren, M.F., Sevkli, M., Liang, Y.-C., Gencyilmaz, G.: Particle swarm optimization algorithm for permutation flowshop sequencing problem. *Proceedings of the 4<sup>th</sup> International Workshop on Ant Colony Optimization and Swarm Intelligence*, Brussels, Belgium (2004) 382-390
16. Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job shop scheduling problems using genetic algorithms I. *Journal of Computers and Industrial Engineering* **30** (1996) 983-997
17. Lawrence, S.: *Supplement to Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques*. Pittsburgh, PA: GSIA, Carnegie Mellon University (1984)
18. Taillard, E.: Benchmarks for basic scheduling problems. *European Journal of Operational Research* **64** (1993) 278-285
19. Mendes, R., Kennedy, J., Neves, J.: Watch thy neighbor or how the swarm can learn from its environment. *Proceedings of the IEEE Swarm Intelligence Symposium 2003*, Indianapolis, Indiana (2003) 88-94
20. Veeramachaneni, K., Peram, T., Mohan, C. Osadciw, L.A.: Optimization using particle swarms with near neighbor interactions. *Proceedings of the Genetic and Evolutionary Computation Conference* (2003) 110-122
21. Pongchairerks, P., Kachitvichyanukul, V.: Particle swarm optimization with multiple social learning structures. *The 36<sup>th</sup> CIE Conference on Computer & Industrial Engineering*, Taipei, Taiwan (2006) 1556-1567

## **A Prototype of Scheduling for Anthropocentric Manufacturing System**

Bandit Suksawat \*, Weiming He, Hiroyuki Hiraoka, and Tohru Ihara

Department of Precision Mechanics, Faculty of Science and Engineering, Chuo University,

1-13-27 Kasuga, Bunkyo-ku, Tokyo, 112-8551, Japan

bandit@mail-ihara.mech.chuo-u.ac.jp, ihara18@mech.chuo-u.ac.jp

**Abstract.** The proposed anthropocentric manufacturing system (AMS) scheduling method consists of a pre-process and a post-process. The pre-process is an estimation of the degree of job and technicians' characteristics in one lot of jobbing by a leader, using a skill-based manufacturing thought model. The post-process consists of skill level evaluation algorithm and GA algorithm. The estimated results are valued to be technician's skill level on each job, using skill level evaluation algorithm; and GA initial populations construct the chromosomes by randomly selection high skill level technicians and their operable machines. An example of one lot of special parts is used for scheduling by this proposed method. The best result performs a maximum total skill level index (TSI) and a total earliness index (TEI); and a minimum man-hour index (MHI). The simulation results also ensure that the algorithm can solve the scheduling problems by limiting approximately 40% of manufacturing cell resources.

**Keywords:** Genetic algorithm, 3M scheduling, skill-based scheduling, anthropocentric manufacturing system.

### **1 Introduction**

Scheduling is a problem of allocating limited resources to operations over time, and the goal of scheduling is to find an appropriate allocation-schedule, which maximizes certain performance [1]. The maximum performance of scheduling will help manufacturers to respond to market demands quickly and to run plants efficiently. The modern manufacturing system, a flexible manufacturing

---

\* Corresponding Author. Tel. & Fax: 813-38171822, Email: bandit@mail-ihara.mech.chuo-u.ac.jp.

cell (FMC), is a group of machines, working together to perform a set of functions on a particular part or product. A manufacturing cell can produce more than one family part as long as each part can be completely processed in that cell [2]. The advanced manufacturing technology, however, does not improve productivity unless it is accompanied with a coherent strategy and operated by skilled or capable technicians. The integration of people and technology in the factory according to an anthropocentric manufacturing system (AMS) is accepted as an essential to improve productivity [3].

The AMS, an extension version of FMC, is important to customer production as a make-to-order production type. This manufacturing cell is capable to produce a variety parts in small lots by skilled technicians with attitudinal skills working in an autonomous working environment such as working with computer numerical control (CNC) machines. The CNC machines have large magazines, which is capable to holding a variety of tools and automatic tool changers [4]. However, the allocation of skilled technicians on the suitable jobs is difficult and complicated due to varieties of technicians, job characteristics, machine types and machine models. Thus, the efficient allocation of skilled technicians is a problem that limits the AMS scheduling for enterprises.

The scheduling problem is generally derived the schedule associated with evaluation of worker's skill level and adapted to genetic algorithms (GA) scheduling. For this purpose, firstly, the evaluation of job and technicians' characteristics, using skill-based manufacturing thought model is introduced. Secondly, the skill level evaluation is presented. The high skilled level technicians and their operable machine tools adapted to GA approach are utilized to solve the scheduling problem. Thirdly, the simulation of scheduling is performed to find minimum manufacturing cell resources for employing with this proposed scheduling method. This article is organized as follows: section one comprises the introduction and background of the study. Section 2 presents the scheduling method of AMS including pre-process and post-process of scheduling. Section 3 describes an example of this scheduling and simulation experiments. The scheduling and simulations results are presented in section 4; and the conclusions are summarized in section 5.

## 2 Scheduling Method of Anthropocentric Manufacturing System

The production of shop floor is done and each job is produced specifically to meet the customers' specification. All operation processes of a job is handled by a skilled technician and completed within a machine. Thus, the scheduling problem characteristics of this production are probably defined as discrete, automated groups of open job shop or batch shop [5]. This paper also deals with the scheduling problem of a single manufacturing cell, assuming that the activities related to the cell formation have been completed.

The AMS scheduling method is divided into two processes. The first is pre-process of scheduling, which the FMC leader as an expert skilled technician determines the degree of job and the technicians' characteristics for each job, based on the skill-based thought model. The second is post-process of scheduling, which the compatibility of the job and the technicians' characteristics

is evaluated to determine the technicians' skill level for each job. This evaluated skill level algorithm is integrated to GA scheduling, in order to allocate the appropriate skilled technicians to the operable machine tools and jobs. The main contribution of this proposed scheduling method is described as follows.

### 2.1 Pre-process of Scheduling

The pre-process of scheduling is an evaluation method of the degree of job and technicians' characteristics in one lot of jobbing productions with skill-based manufacturing thought model. The thought model (mental model), consisting of idea extraction, concept and design, shows the route of decision-making processes of skilled workers, based on their knowledge and skills. The model, therefore, can solve many problems, related to skill-based manufacturing in cells such as process planning and cell concept design [6]. The thought model is also a fundamental aspect of AMS solutions for the problems in manufacturing cells.

The first step of the pre-process is a consideration of all orders process, based on group technology in order to classify the jobs. After that, the FMC leader takes a job balance and job distribution to a cell using skill-based manufacturing thought model as shown in Figure 1. This model is extracted from reviewing data among small enterprises in Japan. All nodes are determined, based on Japanese manufacturing culture and process design [7] [8]. The process of job balance is considered from amount of experience based on measuring the skill on the jobs, a due-date specifying expected finish date, production cost, available capacity of facilities, and quantity of production parts as represented in the center of the model. The aim of job balance is to equally spread the load of each jobbing in such a way that maximizes cell resource utilization while minimizing the total job throughput time. The job distribution is an evaluation of the degree of job and technicians' characteristics, consisting of quickness, difficulty, complexity, thoroughness and reliability of work. The drawings and production details of jobs such as size, shape, material, processing

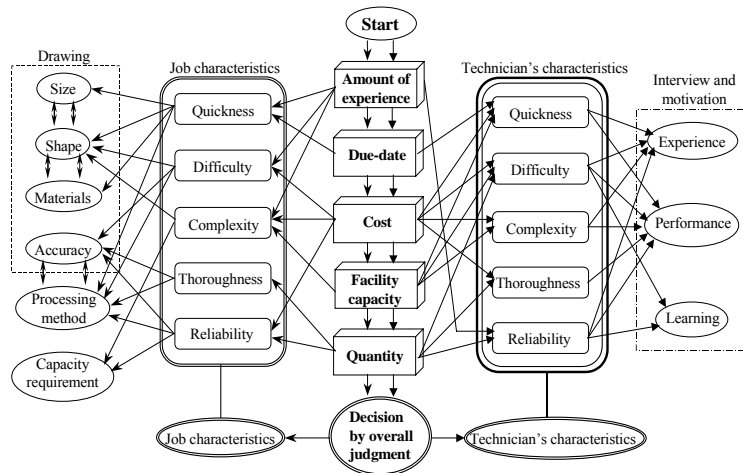


Fig. 1. Skill-based manufacturing thought model

method, capacity requirement and accuracy are matched to job characteristics. The technicians are motivated to the points in the production process and machine tools operation for each job. The technicians are observed and interviewed to evaluate their experiences, performances and learning skills in order to determine each technician’s characteristics. The FMC leader evaluates both job and technicians’ characteristics by decision-making, based on experience and cognitive psychology of production [9] [10]. The semantic differential [11] is used to help the FMC leader to determine the degrees (ratings) of each subject of job and technicians’ characteristics. The ratings of semantic differential have five levels [12], consisting of 1, 2, 3, 4 and 5 which is represented very low, low, medium, high and very high, respectively.

## 2.2 Post-process of Scheduling

The post-process is an application of algorithms to schedule appropriate jobs to skilled technicians and their operable machine over limited time. This process consists of skill level evaluation and adaptive GA scheduling.

### 2.2.1 Skill Level Evaluation

The technician’s skill level evaluation method is a matching of the compatibility of evaluated degree of job and technicians’ characteristics from pre-process scheduling. This evaluation method uses an algorithm as shown in Figure 2 (a). In this algorithms, *char* means the content of the job and technicians’ characteristics; numerical of *char* = 1, 2, 3, 4 and 5 represents quickness, difficulty, complexity, thoroughness and reliability, respectively. The example of skill level evaluation for technician ID01 on Job01 is shown in Figure 2 (b). The result shows that evaluated skill level is level 4. The definition of skill level 1, 2, 3, 4 and 5 is defined as unskilled, semi-skilled, skilled, skilled with addition and technical skilled, respectively [13]. This proposed algorithm is attached at the beginning of GA algorithm as presented in Figure 3.

```

procedure: Technician’s skill level evaluation
input: job and technicians’ characteristics
output: technician’s skill level on each job
n : number of jobs, t : number of technicians
begin
  for (job = 1; job ≤ n)
    for (tech = 1; tech ≤ t)
      for (char = 1; char ≤ 5)
        if (techchar ≥ jobchar) then {CompareSubjectchar = 1}
        end
        SkillLeveltech,job = ∑char=1char=5 CompareSubject;
      end
    end
  end
end
  
```

(a) Skill level evaluation algorithm

**An example of skill level evaluation**  
(Technician ID01’s skill level on Job01)

characteristic subjects	ID01 VS Job01
1:Quickness	2 = 2
2:Difficulty	3 < 4
3:Complexity	4 = 4
4:Thoroughness	4 > 2
5:Reliability	3 = 3
<b>Evaluated skill level</b>	<b>= 4</b>

(b) Extracted evaluation an example

Fig. 2. Illustrations of skill level evaluation method

2.2.2 GA based AMS Scheduling

Genetic algorithms (GA) are stochastic search techniques for approximating optimal solutions within complex search spaces. The technique is based upon the theory of evolution, in which the fitness of an individual determines its ability to survive and reproduce [14] [15].

In this research, the general GA procedure is used. The technician’s skill level evaluation method is attached before randomly initial populations. The algorithm is illustrated in Figure 3. The high skilled technicians (levels 3-5) and their operable machine tools, related to the machining method of each job, are randomly selected for each chromosome. Each operation is encoded into a gene, represented by an alphanumeric string. Each chromosome is divided in to  $n$  bits (jobs); and each job is distributed into three sub-bit strings that represent the machine type ( $T$ ), the machine model ( $M$ ), and the technician ( $E$ ). We define the fitness evaluation of the chromosomes as maximum number of non-delayed jobs of candidate solutions on each chromosome. A traditional roulette wheel selection operator is used to select a pair of individuals from the current populations in order to reproduce new initial populations. After a new population of chromosomes is selected, it will be processed by genetic operation process, using crossover and mutation operations. One point crossover and random change in one chromosome of mutation are used. The termination of the genetic searching process is determined, when allocated working time of all jobs are equal to or less than a due-date, then the genetic searching process will be stopped. Otherwise, the process will proceed to the next generation with new populations.

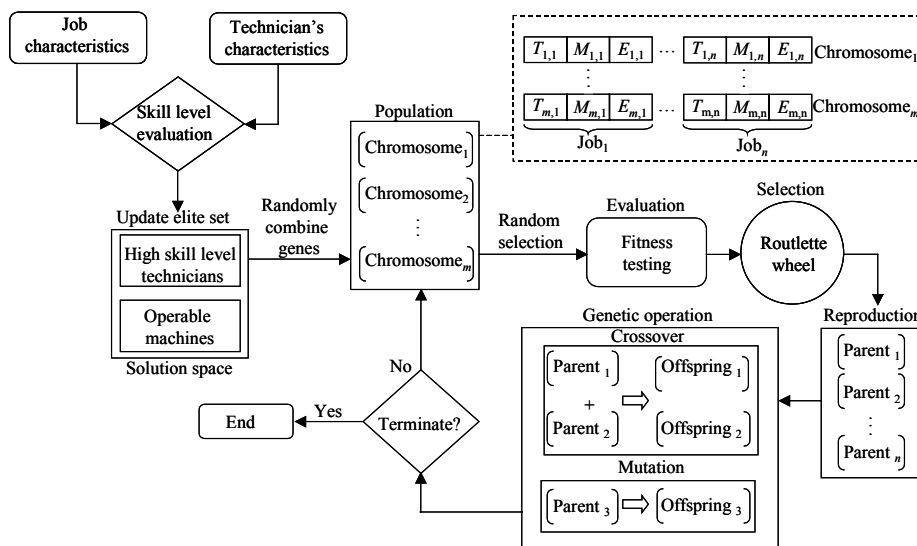


Fig. 3. Flow chart of post-process in AMS scheduling

### 3 An Example of AMS Scheduling and Simulation Experiments

We performed the scheduling to confirm effectiveness of the proposed AMS scheduling and evaluate performance of the scheduling results, which relate to the quality and quantity of FMC. The minimum FMC resources are also examined by simulation of scheduling. This aims to ensure the implementation of designed algorithms within the limitation number of FMC resources.

**Table 1.** Manufacturing cell resources including of machine types and models, and operators

Technician	CNC Milling		CNC Lathe		Milling Center (MC)			Turning Center (TC)			
	Model No.		Model No.		Model No.			Model No.			
	III	IV	III	IV	II	III	IV	I	II	III	IV
ID01	☑	☑	☑	☑	✗	✗	✗	☑	☑	☑	☑
ID02	✗	☑	✗	✗	✗	☑	☑	✗	✗	✗	✗
ID03	☑	☑	☑	☑	☑	☑	☑	✗	✗	✗	☑
ID04	☑	☑	✗	☑	✗	✗	☑	✗	☑	☑	☑
ID05	✗	☑	☑	☑	☑	☑	☑	✗	✗	✗	✗
ID06	✗	☑	☑	☑	✗	✗	✗	✗	✗	☑	☑
Resource	75.00 %		75.00 %		50.00 %			41.67 %			

☑ : Operable; ✗ : Inoperable

**Table 2.** An example of a lot of job, estimated job and technicians' characteristics

Job No.	Job characteristic					Working time (hr)	Due-date (hr)	Machining Method	Technician	Technician's characteristic				
	Quickness	Difficulty	Complexity	Thoroughness	Reliability					Quickness	Difficulty	Complexity	Thoroughness	Reliability
1	2	2	4	2	3	30	75	Turning	ID01	2	3	4	4	3
2	2	4	4	2	3	45	65	Milling		5	4	3	2	2
3	3	3	4	2	4	15	40	Turning	ID02	3	2	2	3	5
4	4	3	3	2	4	50	115	Milling		2	3	5	5	2
5	3	2	4	3	3	35	60	Turning	ID03	2	5	4	3	4
6	3	2	3	4	4	35	75	Turning		3	2	4	3	4
7	3	4	2	2	3	50	85	Milling	ID04	3	2	4	3	4
8	2	4	4	3	2	50	90	Milling		2	5	4	3	4
9	3	4	2	3	3	40	80	Turning	ID05	3	2	4	3	4
10	4	4	2	3	3	30	60	Turning		2	4	3	4	4

**Table 3.** Reducing of FMC resources for simulation experiments

Experiment	CNC Milling (%)	CNC Lathe (%)	MC (%)	TC (%)	Total average (%)
1	75.00	75.00	50.00	41.67	60.42
2	50.00	50.00	50.00	41.67	47.92
3	50.00	50.00	44.44	41.67	46.53
4	41.67	50.00	44.44	41.67	44.45
5	41.67	41.67	44.44	41.67	42.36
6	41.67	41.67	38.89	41.67	40.98
7	33.33	41.67	38.89	41.67	38.89
8	33.33	41.67	38.89	37.50	37.85
9	33.33	33.33	38.89	37.50	35.76

The AMS scheduling implementation is performed with an example of one jobbing, consisting of ten jobs, and six technicians. The details of the operable tools of each technician, which show the difference of operable number of machine types and models, are summarized in Table 1. All drawings of jobs are based on considerations of the FMC leader, concerning of determining the most suitable machining process and working time. However, job characteristics are considered by a skill-based thought model, previously mentioned in the pre-process scheduling. The results of job characteristics, working time, job due-date and technician's characteristics consideration are shown in Table 2. These data are used in scheduling and simulation experiments.

For the simulation experiments, the number of operable machines of technicians in each machine type is randomly reduced. Table 3 shows the simulation experiments, consisting of nine experiments with maximum and minimum number of resources equal 75% and 33.33%, respectively. For instance, the experiment 1 in Table 3 represents the average number of resources, formerly presented in Table 1. The simulation as 16 runs algorithms is defined as standard sample size, calculated from 50 populations with 95% of confidence level and 20% of confidence interval [16]. Each experiment will be simulated until the accumulation runs of algorithms, which are under a limited iteration, equal to stand sample size. The minimum resources determination is obtained from analysis numbers of run over limited iteration of GA solution convergence. The parameters, used in our algorithms are as follows: limited iteration = 10000, population size = 40, crossover probability rate = 0.7, mutation probability rate = 0.1. The algorithm is developed using MATLAB 7.1; the scheduling and simulation experiments are performed on a PC Pentium IV 3.0 GHz with 1.0 GB RAM.

## 4 Results

The technician-work gantt-chart illustrates the scheduling results from the post-process in AMS

scheduling. The gantt-chart shows job handlings, machine tool usages, skill levels and working time of each technician. As shown in Figure 4, the gantt-charts that have the same make-span are different in man-hour of each technician and technician’s skill level on each job. Our proposed new method evaluates the performance of scheduling results in order to choose the best one among same make-span of scheduling results, using three scheduling performance indexes including a total skill level index (TSI), a total earliness index (TEI) and a man-hour index (MHI). TSI is sum of the technician’s skill level of each job in one lot of jobs. Thus, the TSI can be defined by equation (1).

$$TSI = \sum_{Job=1}^{Job=n} \text{Technician's skill level} \tag{1}$$

TEI, a total earliness of all jobs, can be calculated by equation (2).

$$TEI = \sum_{Job=1}^{Job=n} (\text{Due - date} - \text{Working finished time}) \tag{2}$$

MHI is the difference between the maximum man-hour and the minimum man-hour of the technicians. This index is determined by equation (3).

$$MHI = (\text{Man-hour})_{\max} - (\text{Man-hour})_{\min} \tag{3}$$

Table 4 shows the analysis of scheduling performance results. All forty scheduling results are classified into three groups, consisting of group  $\beta$ ,  $\kappa$  and  $\pi$  at TEI equals to 240, 235, and 205, respectively. Each group consists of different number of levels. The numbers of scheduling results of group  $\beta$  shows the highest possible occurrence and performs less mean iterations among the other groups. In this group, the  $\beta_4$  level presents maximum possibility to obtain the results with 2188 mean iteration of computing. However, the best answer of scheduling with the minimum MHI = 50, the maximum TEI = 240 and TSI = 38 (Figure 4) is similar to the result of  $\beta_1$  in Table 4. The maximum TEI affects the manufacturing starting time of a new lot to achieve the quantity of FMC production. The maximum TSI also responds to the output quality of production. Since, it is difficult to allocate similar man-hours for each technician in one lot of job, thus the minimum MHI resulting from this approach makes an easier scheduling for a new lot of job and satisfies

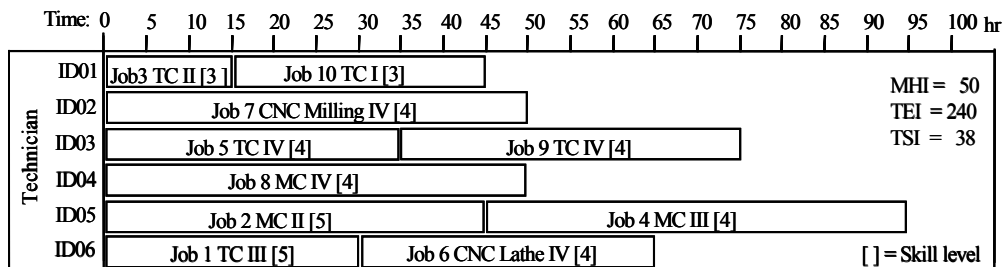


Fig. 4. Technician-working gantt-chart, represented the best scheduling result

workers. However, an implementation of the skill-based manufacturing thought model, using semantic differential, to other varying lots may result in different number of classified groups and the value of each index. These groups and value of indexes depend on job and technicians' characteristics, due-date and working time of each job.

**Table 4.** Analysis scheduling performance

Group	Level	TEI	TSI	MHI	Number of result	Total number of result	Mean iteration
$\beta$	$\beta 1$	240	40	50	1	2	1757
		240	40	65	1		
	$\beta 2$	240	39	60	2	4	1580
		240	39	65	2		
	$\beta 3$	240	38	50	1	4	1628
		240	38	55	1		
		240	38	60	2		
	$\beta 4$	240	37	50	4	10	2188
		240	37	55	2		
		240	37	65	4		
	$\beta 5$	240	36	50	1	9	2027
		240	36	55	1		
		240	36	60	2		
		240	36	65	5		
$\beta 6$	240	35	50	1	5	885	
	240	35	60	1			
	240	35	65	3			
$\beta 7$	240	34	55	1	2	583	
	240	34	65	1			
$\kappa$	$\kappa 1$	235	39	65	1	1	2693
	$\kappa 2$	235	36	65	1	1	2745
$\pi$	$\pi 1$	205	38	95	2	2	8241

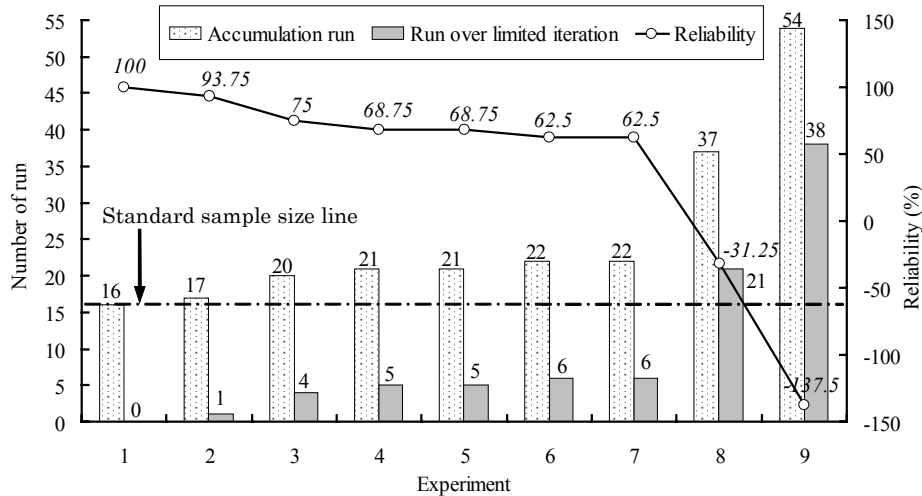


Fig. 5. Results of simulation experiments

Figure 5 shows the results of simulation experiments. The numbers of accumulate run and run over limited iteration are used to analyze the reliability of each experiment. The analysis results show that unreliability of computing the scheduling performance of the experiment 8 and 9 are -31.25% and -137.5%, respectively. According to the number of run over limited iteration is higher than the standard sample size, the experiment 7 is a critical point for reducing the number of operable machine with 38.89% of total average resources. This result ensures that the scheduling can be performed in the case of many machines breaking down or the technicians suddenly losing control on some machines.

## 5 Concluding Remarks

The AMS scheduling method, presented in this paper, consists of both pre-process and post-process. The pre-process, a skill-based manufacturing thought model, is used to estimate the degree of job and technicians' characteristics. The technician's skill level evaluation, the matching of job and technicians' characteristics, and GA algorithms are developed for skill-based scheduling in the post-process.

The scheduling results are shown through the technician-working gantt-chart. The optimal scheduling results are reached within approximately 2200 mean iterations of computation. The best scheduling result performs both a maximum TEI and TSI with a minimum MHI on the gantt-chart. The maximum TEI and TSI affects to the highest quantity and quality of shop floor productivity for a lot of jobbing, respectively; and the minimum MHI eases of a next production lots scheduling. The simulation experiment shows that the developed algorithm can be performed with

limited number of manufacturing cell resources with approximately 40% of total average resources.

According to the solving process of complex scheduling problems as a 3M (man, machine, and manufacturing) scheduling, several popular scheduling methods generally concern with only the allocation of manufacturing jobs and machines over time. However, this proposed scheduling method concerns not only manufacturing jobs and machines, but also men who handle the jobs. This could be the advantage of the proposed method to help enterprises and shop floors to allocate the technician and the operable machine tool to a suitable job resulting in an efficiency increase of manufacturing cell. Furthermore, the proposed method also schedules within the limited manufacturing cell resources (3M), using scheduling algorithms with a few computational time and high reliability of solution convergent. The proposed scheduling method, however, still has an ambiguity in determination of the degree of job and technicians' characteristics in the pre-process of scheduling; because the FMC leader possibly makes different decisions on the same type of job. Thus, further work will be addressed this problem for more efficiency of AMS scheduling.

## References

1. Mattfeld, D.C.: *Evolutionary search and the job shop, Investigations on genetic algorithms for production scheduling*. Physica-Verlag Heidelberg, Germany (1996)
2. Asokan, P., Prabhakaran, G., Satheesh, Kumar G.: Machine-cell grouping in cellular manufacturing systems using non-traditional optimization techniques-A comparative study. *The International Journal of Advanced Manufacturing Technology* **18** (2001) 140-147
3. Filip, F.G., Donciulescu, D.A., Fillip, Cr.I.: Towards intelligent real-time decision support systems for industrial milieu. *Studies in Information and Control* **11** (2002) 303-311
4. Mak, K.L., Wong, Y.S., Wang, X. X.: An adaptive genetic algorithm for manufacturing cell formation. *International Journal of Advance Manufacturing Technology* **16** (2000) 491-497
5. Morton, T.E. , Pentico, D.W.: *Heuristic scheduling systems with applications to production systems and project management*. John Wiley&Sons, New York (1993)
6. Ihara, T., Matsumura, T., Ito, Y.: Computer Aided determination system for engineer's thought model regarding process planning – case of operation planning. *Journal of the Japan Society for Precision Engineering* **64** (1998) 111-115
7. Ihara, T., Gu, H., Ito, Y.: A proposal of modeling method for thought process and its application-concept design of machining cell using thought object. *Journal of the Japan Society for Precision Engineering* **64** (1994) 1174-1178
8. Sugimura, N., Suyoto, Tanaka, T.: Study on process planning system for holonic manufacturing (1 st report, System architecture and recognition of machining feature). *Transactions of the Japan Society of Mechanical Engineers, Series C* **65** (1999) 395-400
9. Barthélemy, J. P., Bisdorff, R., Coppin, G.: Human centered processes and decision support systems. *European Journal of Operational Research* **136** (2002) 232-252
10. Coppin, G., Skrzyniarz, A.: Human centered processes: individual and decision support. *IEEE Intelligent Systems and Their Applications* **18** (2003) 27-33
11. Mondragón, S., Company, P., Vergara, M.: Semantic differential applied to the evaluation of machine tool design. *International Journal of Industrial Ergonomics* **35** (2005) 1021-1029

12. Nagamachi, M.: Kansei engineering; the implication and applications to product development. *IEEE International conference on Systems, Man, and Cybernetics* 6 (1999) 273-278
13. Hirsch-Kreinsen, H., Köhler, C., Moldaschl, M., Schultz-Wild, R.: *Design principles of worker organization and skilled labour in a computer-integrated manufacturing environment*. in Ito, Y.: *Human-intelligence-based manufacturing*. Springer-verlag, London (1993)
14. Goldberg, D.E.: *Genetic algorithms in search, Optimization and machine learning*. Addison-Wesley, MA (1989)
15. Gen, M., Cheng, R.: *Genetic algorithms and engineering design*, John Wiley&Sons, New York (1997)
16. Sample size calculator: <http://www.surveysystem.com/sscalc.htm>

## **Constructive and Tabu Search Algorithms for Hybrid Flow Shop Problems with Unrelated Parallel Machines and Setup Times**

Jitti Jungwattanakit<sup>1</sup>, Manop Reodecha<sup>1\*</sup>, Paveena Chaovalitwongse<sup>1</sup>, and Frank Werner<sup>2</sup>

<sup>1</sup> Department of Industrial Engineering, Faculty of Engineering,  
Chulalongkorn University, Bangkok 10330 Thailand  
jitti.j@student.chula.ac.th

<sup>2</sup> Faculty of Mathematics, Otto-von-Guericke-University,  
P.O. Box 4120, D-39016 Magdeburg, Germany  
frank.werner@mathematik.uni-magdeburg.de

**Abstract.** This paper investigates scheduling heuristics to seek the minimum of a positively weighted convex sum of makespan and the number of tardy jobs in a hybrid flow shop environment where at least one production stage is made up of unrelated parallel machines. Sequence- and machine-dependent setup times are considered. The problem is a combinatorial optimization problem which is difficult to be solved optimally, and hence heuristics are used to obtain good solutions in a reasonable time. Some dispatching rules and flow shop makespan heuristics are developed. Then this solution may be improved by fast polynomial heuristic improvement algorithms based on shift moves and pairwise interchanges. In addition, a metaheuristic tabu search algorithm is proposed. Some tabu search parameters are briefly discussed. The performance of the heuristics is compared relative to each other on a set of test problems with up to 50 jobs and 20 stages.

**Keywords:** Hybrid flow shop scheduling, Unrelated parallel machines, Setup times, Constructive algorithms, Improvement heuristics, Tabu Search algorithm.

### **1 Introduction**

This article is concerned with an industrial scheduling problem, which belongs to the class of NP-hard combinatorial optimization problems. Therefore, the search for efficient methods providing a

---

\* Corresponding Author. Tel: +662-218-6815-6, Fax: +662-251-3969, Email: manop@eng.chula.ac.th.

good feasible solution continues to be a challenge. The heuristics concerned can be classified into two types: constructive and iterative heuristic algorithms. A constructive algorithm generates a single solution. Alternatively, one may also generate several solutions in parallel from which the best one is chosen as the heuristic solution. The interest in iterative algorithms results from the difficulty of solving real large-size problems. This study deals with one of the most popular iterative algorithms known as a tabu search (TS) algorithm, originally proposed by [1]. It has been often used for the approximate solution of combinatorial problems [2] and has been successfully applied to problems in many different areas [3].

In this paper, heuristics are used to solve the problem of scheduling a given set of  $n$  jobs at  $k$  stages on  $m$  unrelated parallel machines. Such a problem occurs in real world problems such as e.g. in the textile industry, the production unit of which is characterized by a multi-stage manufacturing process with multiple production units per stage, called the hybrid flow shop (HFS) problem. Most textile companies are ageing while the technology changes rapidly. It is common to find more modern machines running side by side with older machines. Hence, these companies own machines of different ages, performing the same operations as the newer ones, but would generally require a longer operating time for the same operation. In addition, sequence-dependent setup times incur when machines often have to be reconfigured between jobs. If the length of the setup depends on the job just completed and on the one about to be started, then the setup times are sequence-dependent.

The hybrid flow shop scheduling problem has attracted many researchers due to two main reasons [4]. Firstly, it is a category of problems which is difficult to solve [5]. Secondly, it finds many real-world applications. Although such a problem has been widely studied, most studies concentrate on problems with identical processors [6] – [9]. In this paper, the hybrid flow shop problem with unrelated parallel machines and sequence-dependent setup times is considered. The rest of this paper is organized as follows. The problem is described in Section 2. Some heuristic algorithms are proposed in Section 3. A TS algorithm is presented in Section 4. Computational results and conclusions are shown in Section 5 and Section 6, respectively.

## 2 Statement of the Problem

The hybrid flow shop system is defined as follows: a set of jobs  $J = \{1, \dots, j, \dots, n\}$  has to be sequenced in a flow shop environment with  $k$  stages. For each stage  $t$ , a set  $M^t = \{1, \dots, i, \dots, m^t\}$  of  $m^t$  unrelated machines is considered. Each job  $j$  has its release date  $r_j \geq 0$  and a due date  $d_j \geq 0$ . It has its fixed standard processing time for every stage  $t$ . Preemption is not permitted. Each job is processed by at most one machine at each stage without overlapping between the stages. The processing time  $p'_{ij}$  of job  $j$  on machine  $i$  at stage  $t$  is equal to  $ps'_j / v'_{ij}$ , where  $ps'_j$  is the standard processing time of job  $j$  at stage  $t$ , and  $v'_{ij}$  is the relative speed of job  $j$  which is processed by machine  $i$  at stage  $t$ . In addition, the setup times considered are classified into two types, namely a machine-dependent setup time and a sequence-dependent setup time. A setup time of a job is ma-

chine-dependent if it depends on the machine to which the job is assigned. It is assumed to occur only when a job is the first job assigned to this machine.  $ch_{ij}^t$  denotes the length of the machine-dependent setup time of job  $j$  if job  $j$  is the first job assigned to machine  $i$  at stage  $t$ . A sequence-dependent setup time depends on the job just completed on that machine.  $s'_{lj}$  denotes the time needed to change over from job  $l$  to job  $j$  at stage  $t$ , where job  $l$  is processed directly before job  $j$  on the same machine.

All data are assumed to be known and constant. The scheduling problem has dual objectives, namely minimizing the makespan and minimizing the number of tardy jobs. The objective function to be minimized is:

$$\lambda C_{max} + (1 - \lambda)\eta_T \quad (1)$$

where  $C_{max}$  is the makespan, which corresponds to the completion time of the last job to leave the system,  $\eta_T$  is the total number of tardy jobs in the schedule, and  $\lambda$  is the weight (or relative importance) given to  $C_{max}$  and  $\eta_T$ , ( $0 \leq \lambda \leq 1$ ).

### 3 Heuristic Algorithms

Since the hybrid flow shop scheduling problem is NP-hard, algorithms for finding an optimal solution in polynomial time are unlikely to exist. Thus, heuristic methods are studied to find approximate solutions. Most researchers develop existing heuristics for the classical hybrid flow shop problem with identical machines by using a particular sequencing rule for the first stage [10].

Firstly, a job sequence is determined according to a particular sequencing rule, see the next section. Secondly, jobs are assigned as soon as possible to the machines at every stage using the job sequence determined for the first stage. There are two approaches for this subproblem. The first way is to order the jobs for the other stages, i.e. from stage two to stage  $k$ , according to their completion times at the previous stage, called the FIFO (First-In-First-Out) rule. The second way is to sequence the jobs for the other stages by using the same job sequence as for the first stage, which is called the permutation rule.

#### 3.1 Constructive Heuristics

In order to determine the job sequence for the first stage, we remind that the processing and setup times for every job are dependent on the machine and the previous job, respectively. This means that they are not fixed, until an assignment of jobs to machines for the corresponding stage has been done. Thus, for applying an algorithm for fixing the job sequence for stage one, an algorithm for finding the representatives of the machine speeds and the setup times is necessary.

The representatives of machine speed  $v'_{ij}$  and setup time  $s'_{lj}$  for stage  $t$ ,  $t=1, \dots, k$ , use the minimum, maximum and average values of the data. Thus, the representative of the operating time of

job  $j$  at stage  $t$  is the sum of the processing time  $ps_j^t/v_{ij}^t$  plus the representative of the setup time  $s_{ij}^t$ . Nine combinations of relative speeds and setup times will be used in our algorithms. The job sequence for the first stage is then fixed as the job sequence with the best function value obtained by all combinations of the nine different relative speeds and setup times. For determining the job sequence for the first stage, we adapt and develop several basic dispatching rules and constructive algorithms for the flow shop makespan scheduling problem.

The Shortest Processing Time (SPT) rule is a simple dispatching rule, in which the jobs are sequenced in non-decreasing order of the processing times, whereas the Longest Processing Time (LPT) rule orders the jobs in non-increasing order of their processing times. The Earliest Release Date (ERD) rule is equivalent to the well-known FIFO rule. The Earliest Due Date (EDD) rule schedules the jobs according to non-decreasing due dates of the jobs. The Minimum Slack Time (MST) rule concerns the remaining slack of each job, defined as its due date minus the processing time required to process it. The Slack time per Processing time (S/P) is similar to the MST rule, but its slack time is divided by the processing time required [11].

Palmer's heuristic [12] is a makespan heuristic denoted by PAL in an effort to use Johnson's rule by proposing a *slope order index* to sequence the jobs. The idea is to give priority to jobs that have a tendency of progressing from short times to long times as they move through the stages. Campbell, Dudek, and Smith [13] develop one of the most significant heuristic methods for the makespan problem known as CDS algorithm. In so doing,  $k - 1$  sub-problems are created and Johnson's rule is applied to each of the sub-problems. Gupta [14] provides an algorithm denoted by GUP, in a similar manner as algorithm PAL by using a different slope index and scheduling the jobs according to the slope order. Dannenbring [15] develops a method, denoted by DAN. Furthermore, the CDS and PAL algorithms are also exhibited. Nawaz, Ensore and Ham [16] develop a method, called the NEH algorithm, which is based on the idea that a job with a high total operating time on the machines should be placed first at an appropriate relative order in the sequence. Thus, jobs are sorted in non-increasing order of their total operating time requirements. The final sequence is built in a constructive way, adding a new job at each step and finding the best partial solution.

### 3.2 Improvement Heuristics

Unlike constructive algorithms, improvement heuristics start with an already built schedule and try to improve it by some given procedures. Their use is necessary since the constructive algorithms (especially some algorithms that are adapted from pure makespan heuristics and some dispatching rules such as SPT and LPT) do not consider due dates. In this section, some fast improvement heuristics will be investigated to improve the overall function value by concerning mainly the due date criterion.

The iterative algorithms described in the following and in Section 4 are based on the shift move (SM) and the pairwise interchange (PI) neighborhoods.

The SM neighborhood repositions a chosen job  $\pi_r$  at position  $r$ , which is shifted to position  $i$ , while leaving all other relative job orders unchanged. The PI neighborhood exchanges a pair of chosen jobs  $\pi_r$  and  $\pi_i$ , while leaving all other jobs in the original positions. In order to find a satisfactory solution of the due date problem, we apply fast polynomial heuristics by applying either the shift move (SM) algorithm as an improvement mechanism based on the idea that we will consider the jobs that are tardy and move them left and right or the pairwise interchange (PI) algorithm, where tardy jobs are swapped to different job positions left and right, either to randomly determined two positions (denoted by the number “2”) or to all other positions (denoted by the letter “A”). The best schedule among the generated neighbors is then taken as the result.

## 4 Tabu Search Algorithm

A TS algorithm is an iterative improvement approach designed to avoid terminating prematurely at a local optimum for combinatorial optimization problems. The TS algorithm is based on the idea of exploring the solution space of a problem by moving from one region of the search space to another in order to look for a better solution. To escape from a local optimum, the TS algorithm allows the search to move to the best solution among a set of candidate moves as defined by the neighborhood structure, although it can move to a neighbor with a worse objective function value. Nevertheless, subsequent iterations may cause the search to move repeatedly back to the same local optimum. To prevent cycling back to recently visited solutions, it should be forbidden or declared tabu for a certain number of iterations, called the size (or length) of the list. Its size is a key control parameter of the TS algorithm. This is accomplished by keeping the attributes of the forbidden moves in a list, called the tabu list.

### 4.1 Choice of an Initial Solution

To improve the quality of the solution finally obtained, we also investigated the influence of the choice of an appropriate initial solution by using particular constructive algorithms. We used as an initial solution those obtained from the constructive algorithms and the fast improvement (SM, PI) heuristics.

## 5 Computational Results

Firstly, the constructive algorithms and different fast improvement heuristics are studied. The constructive algorithms are the simple dispatching rules (i.e., the SPT, LPT, ERD, EDD, MST and S/P rules) and the flow shop makespan heuristics (i.e., the PAL, CDS, GUP, DAN and NEH rules). Then, we applied the fast polynomial improvement heuristics based on the four variants discussed

in Section 3.2. We used problems with 10jobs×5stages, 30jobs×10stages, and 50jobs×20 stages and  $\lambda \in \{0, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1\}$  in the objective function. Ten different instances for each problem size have been run. An experiment was conducted to test with the following data: The standard processing times are generated uniformly from the interval [10,100]. The number of unrelated parallel for each stage is generated uniformly from the interval [1,5], but at least one stage is made up of parallel machines. The relative speeds are distributed uniformly in the interval [0.7,1.3]. Sequence- and machine-dependent setup times are generated uniformly from the interval [0,50]. The release dates are generated uniformly from the interval between 0 and half of their total standard processing time mean. The due date of a job is set in a way that is similar to the approach presented by [17] and is as follows:

$$d_j = r_j + \sum_{i=1}^k ps_j^i + \text{total of mean setup time of a job on all stages} + (n-1) \times (\text{mean processing time of a job on one machine}) \times U(0,1) \quad (2)$$

First, we have observed in our tests that the improvement heuristics from Section 3.2 improve the quality of the constructive algorithms by about 60 – 70 percent. In general, the A-PI algorithm should be selected as the improvement algorithm. Hence, we use in the following only the A-PI improvement heuristics.

Next, we tested the constructive algorithms that are separated into four main groups. The first heuristic group includes the simple dispatching rules, and the second heuristic group includes the flow shop makespan heuristics adapted. The third and fourth heuristic groups are generated from the first two groups of heuristics where the solutions are improved by the selected polynomial improvement algorithm based on the A-PI improvement heuristics (hereafter, these algorithms are denoted by the first letter “I” in front of the letters describing the heuristics of the first two groups). Among the simple dispatching rules (heuristic Group I), the SPT, LPT and ERD rules are good dispatching rules. However, in general the SPT rule outperforms the other dispatching rules for  $\lambda < 0.01$ , and the LPT rule is better than the other rules otherwise. Among the adapted flow shop makespan heuristics in the heuristic Group II, the NEH algorithm is clearly the best algorithm among all studied constructive heuristics (but in fact, this algorithm takes the convex combination of both criteria into account when selecting partial sequences). The CDS algorithm is certainly the algorithm on the second rank (but it is substantially worse than the NEH algorithm even if the makespan portion in the objective function value is dominant, i.e. for large  $\lambda$  values). These results are similar to the conclusions of [18], where the results for small problem sizes are compared with the optimal solutions.

Thirdly, we studied the TS algorithm with a random initial solution. The favorable TS parameters, i.e., the number of neighbors (10 through 50, in step of 10), the neighborhood structure (PI and SM), and the size of tabu list (5, 10, 15, and 20) were investigated. From the preliminary tests, we set the time limit equal to one second for the problems with ten jobs, ten seconds for the problems with 30 jobs, and 30 seconds for the problems with 50 jobs. In Table 1, we give the average

**Table 1.** The effect of the tabu search parameters

$\lambda$	Problem size	Number of Neighbors					Neighborhood Structures		Size of Tabu List			
		10	20	30	40	50	PI	SM	5	10	15	20
0	10×5	0.029 <sup>a</sup>	<b>0.017</b>	0.033	0.050	0.083	<b>0.033</b>	0.052	0.057	<b>0.030</b>	0.040	0.043
	30×10	0.400	<b>0.242</b>	0.313	0.392	0.463	<b>0.337</b>	0.387	0.380	0.367	<b>0.347</b>	0.353
	50×20	<b>0.050</b>	0.146	0.346	0.533	0.625	<b>0.163</b>	0.517	0.380	0.347	<b>0.287</b>	0.347
	Sum	0.479	<b>0.404</b>	0.692	0.975	1.171	<b>0.533</b>	0.955	0.817	0.743	<b>0.673</b>	0.743
0.001	10×5	0.954 <sup>b</sup>	0.989	<b>0.943</b>	1.477	3.281	<b>0.911</b>	2.146	2.254	1.152	<b>0.924</b>	1.786
	30×10	7.912	5.236	<b>4.940</b>	5.640	6.250	<b>5.923</b>	6.068	6.046	5.912	6.412	<b>5.612</b>
	50×20	0.981	<b>0.945</b>	2.040	3.409	4.250	<b>2.050</b>	2.600	2.511	<b>2.221</b>	2.339	2.228
	Sum	9.847	<b>7.170</b>	7.923	10.526	13.781	<b>8.884</b>	10.814	10.811	<b>9.285</b>	9.675	9.626
0.005	10×5	1.136	0.648	<b>0.618</b>	0.799	1.282	<b>0.826</b>	0.967	1.036	<b>0.707</b>	0.894	0.949
	30×10	6.057	3.942	<b>3.611</b>	4.134	4.195	4.650	<b>4.125</b>	<b>4.325</b>	4.354	4.341	4.532
	50×20	1.875	<b>1.663</b>	2.623	3.493	4.099	<b>2.635</b>	2.867	2.837	2.746	2.711	<b>2.710</b>
	Sum	9.068	<b>6.253</b>	6.852	8.426	9.576	8.111	<b>7.959</b>	8.198	<b>7.807</b>	7.946	8.191
0.01	10×5	0.781	<b>0.419</b>	0.474	0.730	1.099	<b>0.667</b>	0.735	0.855	<b>0.534</b>	0.552	0.863
	30×10	5.264	3.653	<b>3.549</b>	3.931	4.390	4.413	<b>3.903</b>	<b>4.097</b>	4.165	4.238	4.131
	50×20	2.171	<b>1.807</b>	2.783	3.744	4.161	<b>2.759</b>	3.108	2.933	3.134	<b>2.687</b>	2.979
	Sum	8.216	<b>5.879</b>	6.806	8.405	9.650	7.839	<b>7.746</b>	7.885	7.833	<b>7.477</b>	7.973
0.05	10×5	0.535	0.176	<b>0.166</b>	0.191	0.332	0.379	<b>0.181</b>	0.261	<b>0.239</b>	0.257	0.364
	30×10	4.585	3.727	<b>3.632</b>	3.777	4.119	4.298	<b>3.638</b>	3.939	4.019	3.989	<b>3.926</b>
	50×20	2.410	<b>1.734</b>	2.793	3.338	3.905	2.839	<b>2.834</b>	<b>2.667</b>	2.905	2.903	2.869
	Sum	7.530	<b>5.637</b>	6.591	7.306	8.356	7.516	<b>6.653</b>	<b>6.867</b>	7.163	7.149	7.159
0.1	10×5	0.381	<b>0.119</b>	0.158	0.154	0.344	0.324	<b>0.138</b>	0.278	<b>0.150</b>	0.230	0.267
	30×10	4.067	3.542	<b>3.458</b>	3.773	3.714	4.004	<b>3.418</b>	3.684	3.769	<b>3.682</b>	3.708
	50×20	2.174	<b>1.491</b>	2.313	2.925	3.555	<b>2.407</b>	2.576	2.513	2.444	<b>2.427</b>	2.582
	Sum	6.622	<b>5.152</b>	5.929	6.851	7.613	6.735	<b>6.132</b>	6.475	6.363	<b>6.339</b>	6.557
0.5	10×5	0.331	0.164	<b>0.108</b>	0.228	0.282	0.283	<b>0.162</b>	0.219	<b>0.190</b>	0.217	0.264
	30×10	3.705	<b>2.962</b>	3.168	3.182	3.569	3.561	<b>3.073</b>	3.379	<b>3.219</b>	3.375	3.295
	50×20	2.008	<b>1.304</b>	2.098	2.860	3.510	<b>2.310</b>	2.402	2.369	2.340	2.431	<b>2.283</b>
	Sum	6.044	<b>4.430</b>	5.374	6.269	7.360	6.154	<b>5.637</b>	5.967	<b>5.749</b>	6.023	5.842
1.0	10×5	0.358	<b>0.127</b>	0.152	0.218	0.327	0.290	<b>0.183</b>	0.290	<b>0.167</b>	0.207	0.283
	30×10	3.523	<b>2.805</b>	2.877	3.169	3.299	3.341	<b>2.928</b>	3.171	3.105	<b>3.085</b>	3.177
	50×20	2.129	<b>1.415</b>	2.321	2.861	3.551	<b>2.345</b>	2.566	2.434	2.451	2.521	<b>2.416</b>
	Sum	6.011	<b>4.347</b>	5.351	6.249	7.177	5.976	<b>5.677</b>	5.895	<b>5.723</b>	5.813	5.876

<sup>a</sup> average absolute deviation for  $\lambda = 0$ , <sup>b</sup> average percentage deviation for  $\lambda > 0$ .

(absolute for  $\lambda = 0$  resp. percentage for  $\lambda > 0$ ) deviation of a particular algorithm from the best solution in these tests. From the full factorial experiment, we analyzed our results by means of a multi-factor Analysis of Variance technique using a 5% significant level. We have found that for all TS parameters, there are significant differences. For the number of neighbors, 20 and 30 non-tabu neighbors are good parameters, but generating 20 nontabu neighbors is the best variant. The PI moves are better than the SM for  $\lambda < 0.005$ , whereas for  $\lambda = 0.005$  and the problem sizes 10

jobs  $\times$  5 stages as well as 50 jobs  $\times$  20 stages, there are not statistically significantly differences in both neighborhood structures, but they are statistically significant for the problem size 30 jobs  $\times$  10 stages. For the problem size 50 jobs  $\times$  20 stages and  $\lambda \geq 0.1$ , although the average main effect of the PI moves is better than that of SM, it has been found that there is a statistically significant interaction between the neighborhood structure and the number of neighbors, that is, for 20 non-

**Table 2.** Average performance of tabu search algorithms with biased initial solutions

$\lambda$	Problem size	Group I			Group II			Group III			Group IV		
		SPTS	LPTS	ERDTS	PALTS	CDSTS	NEHTS	ISPTS	ILPTS	IERDTS	IPALTS	ICDSTS	INEHTS
0	10 $\times$ 5	0.020 <sup>a</sup>	0.000	0.020	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.020	0.000
	30 $\times$ 10	0.220	0.220	0.260	0.260	0.340	0.180	0.280	0.220	0.220	0.220	0.300	0.160
	50 $\times$ 20	0.060	0.060	0.100	0.120	0.020	0.000	0.100	0.120	0.080	0.140	0.040	0.000
	Sum	0.300	0.280	0.380	0.380	0.360	0.180	0.380	0.340	0.300	0.360	0.360	0.160
0.001	10 $\times$ 5	0.014 <sup>b</sup>	0.090	0.071	0.163	0.100	0.166	0.531	0.575	0.044	1.024	1.056	0.534
	30 $\times$ 10	3.674	4.649	4.377	4.142	3.715	3.413	4.107	3.829	4.898	4.915	4.934	3.421
	50 $\times$ 20	0.667	0.938	1.415	0.648	0.544	0.343	0.847	1.014	0.797	1.087	0.380	0.350
	Sum	4.355	5.677	5.863	4.953	4.359	3.922	5.485	5.418	5.739	7.026	6.370	4.305
0.005	10 $\times$ 5	0.306	0.984	0.912	0.615	1.179	1.076	0.573	1.279	1.044	0.832	1.103	0.721
	30 $\times$ 10	2.892	3.177	2.847	2.776	2.967	3.353	2.906	3.068	3.140	3.182	3.329	3.461
	50 $\times$ 20	1.363	1.223	1.491	1.395	0.876	0.540	0.932	1.745	2.025	2.089	0.676	0.543
	Sum	4.561	5.384	5.250	4.786	5.022	4.969	4.411	6.092	6.209	6.103	5.108	4.725
0.01	10 $\times$ 5	0.598	0.551	0.356	0.420	0.779	0.664	0.468	0.481	0.477	0.628	0.387	0.706
	30 $\times$ 10	3.063	3.562	2.373	3.490	2.851	3.068	3.824	3.346	2.963	3.274	3.109	2.994
	50 $\times$ 20	1.068	0.851	1.180	0.970	1.371	0.539	1.241	1.775	1.635	1.782	1.124	0.539
	Sum	4.729	4.964	3.909	4.880	5.001	4.271	5.533	5.602	5.075	5.684	4.620	4.239
0.05	10 $\times$ 5	0.039	0.033	0.029	0.039	0.029	0.042	0.076	0.052	0.057	0.059	0.054	0.029
	30 $\times$ 10	3.252	3.513	2.357	3.036	3.030	2.128	3.094	2.692	2.743	3.512	3.238	2.233
	50 $\times$ 20	1.064	0.959	1.146	1.083	1.421	0.491	1.027	1.146	1.700	1.531	0.891	0.491
	Sum	4.355	4.505	3.532	4.158	4.480	2.661	4.196	3.891	4.500	5.101	4.183	2.753
0.1	10 $\times$ 5	0.039	0.057	0.009	0.017	0.025	0.030	0.010	0.035	0.031	0.009	0.021	0.012
	30 $\times$ 10	2.698	2.633	1.947	2.515	2.295	1.839	2.569	2.628	2.107	2.682	2.356	1.832
	50 $\times$ 20	0.947	1.183	1.166	0.822	1.182	0.415	0.946	1.304	1.302	1.266	1.036	0.416
	Sum	3.685	3.872	3.122	3.354	3.502	2.285	3.524	3.966	3.440	3.957	3.413	2.260
0.5	10 $\times$ 5	0.061	0.015	0.055	0.006	0.039	0.039	0.027	0.035	0.061	0.019	0.041	0.039
	30 $\times$ 10	2.307	2.396	1.872	2.549	2.426	1.407	2.179	2.256	2.004	1.999	2.288	1.456
	50 $\times$ 20	0.938	1.047	1.228	0.904	1.306	0.327	0.927	1.249	1.172	1.495	0.876	0.314
	Sum	3.306	3.458	3.155	3.458	3.771	1.773	3.133	3.540	3.237	3.514	3.205	1.809
1.0	10 $\times$ 5	0.096	0.059	0.054	0.042	0.069	0.047	0.037	0.065	0.066	0.016	0.024	0.035
	30 $\times$ 10	2.264	2.517	1.808	2.787	2.468	1.741	2.366	2.433	2.082	2.654	2.181	1.747
	50 $\times$ 20	0.878	0.981	1.107	1.172	1.356	0.362	0.972	1.226	1.172	1.328	1.033	0.362
	Sum	3.237	3.557	2.969	4.000	3.893	2.150	3.375	3.724	3.320	3.998	3.237	2.143

<sup>a</sup> average absolute deviation for  $\lambda = 0$ , <sup>b</sup> average percentage deviation for  $\lambda > 0$ .

tabu SM neighbors become better than PI moves. Hence, in general the SM should be selected as the neighborhood structure for  $\lambda \geq 0.005$ . For the size of the tabu list, it has been found that a size of 10 and 15 works best, but a size 10 of the tabu list is recommended.

Finally, we used the recommended TS parameters to test the choice of an appropriate initial solution (some results are given in Table 2). The letters before TS denote the heuristic rule as an initial solution for the TS algorithm. For example, SPTTS means that the SPT rule is used as an initial solution for the TS algorithm. From the experiment, we have found that there are no statistically significant differences in the different initial solutions for the problem sizes 10jobs×5stages and 30jobs×10stages, but it became statistically significantly different for the problem size 50jobs×20stages, in particular for  $\lambda \geq 0.05$  we have found that algorithms INEHTS and NEHTS are better than the others. In general, algorithms INEHTS and NEHTS are good choices for the TS algorithm when using a biased initial solution. These results are consistent with previous studies in which the NEH algorithm is usually applied to provide the initial solution for an iterative algorithm such as a tabu search [19].

## 6 Conclusions

In this paper, first some constructive algorithms have been investigated for minimizing a convex combination of makespan and the number of tardy jobs for the hybrid flow shop problem with unrelated parallel machines and setup times, which is often occurring in real world problems. All algorithms are based on the list scheduling principle by developing job sequences for the first stage and assigning and sequencing the remaining stages by both the permutation and FIFO approaches.

The constructive algorithms are compared to the best known solutions. We have found that among the simple dispatching rules the SPT, LPT and ERD rules are good algorithms whereas among the flow shop makespan heuristics, the NEH algorithm is clearly superior to the other constructive algorithms. When applying the polynomial improvement algorithm, we have found that the all-pairwise interchange algorithm is a good improvement algorithm. Next, we used TS-based algorithms as metaheuristic algorithms. Before we studied the influence of the initial solution on the performance of the TS algorithm, we tested the TS parameters, i.e., the number of neighbors, the neighborhood structure, and the size of tabu list. We have found that a constant number of 20 neighbors works best. The neighborhood structures should be based on shift moves for  $\lambda \geq 0.05$  and on pairwise interchanges of jobs otherwise. The size of the tabu list should be selected as 10. For the recommended TS parameters, we have investigated the influence of the starting solution by using several constructive and improvement algorithms. The variants INEHTS and NEHTS can both be recommended in general.

Further research can be done to use other metaheuristic algorithms such as genetic or ant colony algorithms. The choice of good parameters for them should be tested. The influence of the starting solution should be investigated. Moreover, hybrid algorithms should be developed by

using a tabu search algorithm as a local search algorithm within a genetic algorithm or the other algorithms.

## Acknowledgements

This work was supported in part by the Department of Industrial Engineering, Chulalongkorn University, and by INTAS (project 03-51-5501).

## References

1. Glover, F.: Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research* **13** (1986) 533–549
2. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys* **35** (2003) 268–308
3. Glover, F., Laguna, M.: Tabu search. In C.R.Reeves (ed), *Modern Heuristic Techniques for Combinatorial Problems (UK: Blackwell Scientific Publications)* Oxford, chapter 3 (1993) 70-150
4. Wang, H.: Flexible Flow Shop Scheduling: Optimum, heuristics, and artificial intelligence solution. *Expert Systems* **22** (2005) 78 – 85
5. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the theory of NP-completeness* CA. W.H. Freeman and company, San Francisco (1979)
6. Gupta, J.N.D., Krüger, K., Lauff, V., Werner, F., Sotskov, Y.N.: Heuristics for hybrid flow shops with controllable processing times and assignable due dates. *Computers & Operations Research* **29** (2002) 1417-1439
7. Alisantoso, D., Khoo, L.P., Jiang, P.Y.: An immune algorithm approach to the scheduling of a flexible PCB flow shop. *The International Journal of Advanced Manufacturing Technology* **22** (2003) 819-827
8. Lin, Hung-Tso., Liao, Ching-Jong.: A case study in a two-stage hybrid flow shop with setup time and dedicated machines. *International Journal of Production Economics* **86** (2003) 133-143
9. Wang, W., Hunsucker, J.L.: An evaluation of the CDS heuristic in flow shops with multiple processors. *Journal of the Chinese Institute of Industrial Engineers* **20** (2003) 295-304
10. Santos, D.L., Hunsucker, J.L., Deal, D.E.: An evaluation of sequencing heuristics in flow shops with multiple processors. *Computers & Industrial Engineering* **30** (1996) 681-691
11. Pinedo, M., and Chao, X.: *Operations scheduling with applications in manufacturing and services*. Irwin/McGraw-Hill, New York (1999)
12. Palmer, D.S.: Sequencing jobs through a multi-stage process in the minimum total time--a quick method of obtaining a near optimum. *Operations Research Quarterly* **16** (1965) 101–107
13. Campbell, H.G., Dudek, R.A., Smith, M.L.: A Heuristic algorithm for the n-Job m-Machine sequencing problem. *Management Science* **16**(1970) B630–B637
14. Gupta, J.N.D.: A functional heuristic algorithm for the flowshop scheduling problem. *Operations Research Quarterly* **22** (1971) 39-47
15. Dannenbring, D.G.: An evaluation of flow shop sequencing heuristics. *Management Science* **23** (1977) 1174-1182
16. Nawaz, M., Ensco, Jr. E.E., Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* **11**(1983) 91–95

17. Rajendran, C., Ziegler, H.: Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times. *European Journal of Operational Research* **149** (2003) 513–522
18. Jungwattanakit, J., Reodecha, M., Chaovalitwongse, P., Werner, F.: (2006) Sequencing heuristics for flexible flow shop scheduling problems with unrelated parallel machines and setup times. *Proceedings of the 2006 IE Network National Conference* Bangkok, Thailand (2006) session F53 pp 1-8
19. Grabowski, J., Wodecki, M.: A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Computers & Operations Research* **31**(2004) 1891–1909

## Single Machine Scheduling for Jobs with Individual Due Dates and a Learning effect: Genetic Algorithm Approach

Pei Chann Chang<sup>1</sup>, Shih-Hsin Chen<sup>2</sup>, V.Mani<sup>3</sup>

<sup>1</sup> Department of Information Management, Yuan Ze University, 135, Yuan-Dong Road, Tao-Yuan, Taiwan, 32026, R.O.C.

{iepchang}@saturn.yzu.edu.tw

<sup>2</sup> Department of Industrial Engineering and Management, Yuan Ze University, Tao-Yuan 32026, Taiwan, R.O.C.

{s939506}@mail.yzu.edu.tw

<sup>3</sup> Department of Aerospace Engineering, Indian Institute of Science, Bangalore India 560-012.

{mani}@aero.iisc.ernet.in

**Abstract.** In this paper, we present a genetic algorithm approach that considers the single machine scheduling problem. There are  $n$  jobs, in which each job  $j$  ( $j = 1, 2, \dots, n$ ) has a normal processing time  $p_j$ , a due date  $d_j$ , earliness penalty  $\alpha_j$ , and tardiness penalty  $\beta_j$ . The objective is to find the sequence of jobs that minimizes the weighted sum of earliness and tardiness penalty costs, with a learning effect. The machine idle times are not considered. The worker involved in doing the same operations on a machine, learns the task and the worker become efficient in that job. Thus, in scheduling problems the job processing time depends on the position of the job in a sequence. This is the learning effect. The problem of finding the optimal sequence of jobs is a difficult combinatorial optimization problem. It can be easily seen that there are  $n!$  sequences are possible for this problem. Because of the difficult nature of this problem, we use genetic algorithm to obtain the best/optimal sequence of jobs. Various issues related to genetic algorithm such as solution representation, selection methods, and genetic operators are presented. We show via numerical examples (test problems) that the genetic algorithm approach takes only small computation time to solve fairly large problems (50 jobs).

**Keywords:** scheduling, single-machine, learning effect, genetic algorithms.

---

\* Corresponding Author. Tel: 886-3-4636165, Fax: 886-3-4635319. Email: iepchang@saturn.yzu.edu.tw.

## 1 Introduction

Single-machine scheduling problem is one of the well known problems studied by many researchers in the past fifty years. In most of these studies, the processing time of a job is assumed to be a constant. Or in other words, the processing time of a job is independent of its position in the sequence. In practical situations, this assumption may not be true. This is due to the fact that workers ability to learn when they are processing similar tasks. Because of the learning effect, the processing time of a job depends on its position in the sequence. The concept of “learning effect” is known in management literature.

The learning effect can arise in scheduling of jobs due to the fact that workers are processing same type of jobs on the same machine. So it is easy for workers to improve their performance and the processing time of a job will reduce because of the learning. The learning effect, for the case of single machine scheduling problem is first considered in [1]. In this study, the objective is to minimize the deviation from a common due date, in presence of learning effect. Another objective considered is the minimization of flow time [2]. In both the studies [1] and [2], the optimal sequence of jobs is obtained by solving an assignment problem. The learning effect in a two machine flowshop scheduling is presented in [3]. The objective is to find a sequence of jobs that minimize the total completion time in presence of learning effect. A branch and bound technique is used for the solution. To speed up the computation several dominance properties are derived. A heuristic algorithm is also introduced in [3], to improve the efficiency of the branch and bound technique. In these studies [1,2,3], the importance of considering the learning effect in production scheduling is discussed.

The learning effect for the problem of flow time minimization on parallel identical machines, is considered in [4], and a polynomial time solution is presented. The problem of flowshop scheduling with a learning effect is also studied in [5], and is shown that the classical Johnson’s rule is not the optimal solution to minimize makespan. Single-machine scheduling problem with a learning effect to minimize the number of tardy jobs is discussed in [6]. Various issues related to scheduling problems with a learning effect, is also studied in [7,8,9].

It is shown in [10] that the problem of minimizing the total tardiness on one machine is NP-hard. Another related research [11] that consider the case in which the processing times decrease in a piecewise linear fashion, which is a good approximation to study the learning effect. A survey of scheduling with time dependent processing times presented in [12], provide a framework to show how the time-dependent processing time problems have been generalized from the classical scheduling theory.

In this paper, we consider the single machine scheduling problem, in which, there are  $n$  jobs, and each job  $j$  ( $j = 1, 2, \dots, n$ ) has a normal processing time  $p_j$ , a due date  $d_j$ , earliness penalty  $\alpha_j$ , and tardiness penalty  $\beta_j$ . The objective is to find the sequence of jobs that minimizes the weighted sum of earliness and tardiness penalty costs, with a learning effect. The machine idle times are not considered. This is a difficult combinatorial optimization problem [10]. It can be easily seen that there are  $n!$  sequences are possible for this problem. Because of the difficult na-

ture of this problem, we present a genetic algorithm approach to obtain the optimal/best sequence. Genetic Algorithm (GA) is the most well-known technique for solving combinatorial optimization problems. Genetic algorithms developed in [13] attempts to design artificial systems based upon the adaptive process of natural systems. A complete description about genetic algorithms is available in [14,15,16]. A review of application of genetic algorithms in production and operations management is given in [17].

This paper is organized as follows. In the next section, we describe the scheduling problem considered in our study with a learning effect. In Section.3, we present a brief description of genetic algorithm to this scheduling problem. In Section.4, we present the numerical results obtained for the problems, using a genetic algorithm. A conclusion is presented in the last section.

## 2 Single-machine scheduling with a learning effect

We consider the single machine scheduling problem, in which, there are  $n$  jobs, and each job  $j$  ( $j=1,2,\dots,n$ ) has a normal processing time  $p_j$ , a due date  $d_j$ , earliness penalty  $\alpha_j$ , and tardiness penalty  $\beta_j$ . This type of scheduling problems has been studied in [18,19,20,21]. The objective is to find the sequence of jobs that minimizes the weighted sum of earliness and tardiness penalty costs, with a learning effect. The machine idle times are not considered. The sequence is the order in which the jobs are processed in the machine. All the  $n$  jobs available for processing at time  $t=0$ . Because of the learning effect, the processing time of a job depends on its position in the sequence. Hence, the processing time of the jobs are given as

$$p_{jr} = p_j r^a \quad (1)$$

Here  $p_{jr}$  is the processing time of job  $j$ , if it is in position  $r$  of the sequence, and  $a$  is the learning index ( $a < 0$ ). This above equation is introduced in [1] to include the learning effect. We can see from the above equation, the processing time of a job decreases as function of the position in the sequence. The value of  $a$  is obtained from learning curves. An analysis of learning curves in machine shops is discussed in [22]. The objective is to find the sequence of jobs that minimizes the weighted sum of earliness and tardiness penalty costs, with a learning effect. Hence, the objective function for our problem is

$$\text{Minimize } \sum_{j=1}^n \{\alpha_j E_j + \beta_j T_j\} \quad (2)$$

Let  $C_j$  is the completion time of job  $j$ . The earliness and tardiness of job  $j$  are given as

$$E_j = \text{Max}\{0, (d_j - C_j)\} \quad (3)$$

$$T_j = \text{Max}\{0, (C_j - d_j)\}. \quad (4)$$

In this paper, we consider this above problem with a learning effect and present a genetic algorithm to obtain the optimal/best sequence.

### 3 Genetic algorithm for Single-machine scheduling with a learning effect

Genetic Algorithms (GA) is a search algorithm based on the mechanism of natural selection that transforms a population (a set of individuals) into a new population (i.e., next generation) using genetic operators such as crossover, mutation and reproduction. A survival of fittest strategy is adopted to identify the best strings and subsequently genetic operators are used to create a new population for next generation. More details about how genetic algorithm works for a given problem can be found in literature [14,15,16].

In genetic algorithms, the search space contains all search nodes for a given combinatorial optimization problem. GA starts with an initial population of search nodes from the search space. Each search node in the population is evaluated using the objective function and a fitness value is assigned to each search node. New search nodes are generated for next generation based on fitness value and applying genetic operators to the current search nodes. This process is continued for generation after generation until the algorithm converges. To apply genetic algorithm to our scheduling problem with learning effect, we need to address the following factors.

- string representation of search nodes
- population initialization
- selection function and genetic operators
- fitness function
- termination criterion

**String Representation:** The string representation is the process of encoding a sequence for the scheduling problem. The string representation scheme depends on the structure of the problem in GA and also depends on the genetic operators used. For our single-machine scheduling problem, the sequence is a string of integers. The integers corresponds to job numbers. For example, the string  $\{4\ 3\ 5\ 1\ 2\}$ , in our problem represents the sequence in which the jobs are processed, when there are 5 jobs. In general, the length of the string is equal to the number of jobs ( $n$ ). We know that for  $n$  jobs, there are  $n!$  job sequences. We can see that in this string representation all the possible ( $n!$ ) sequences are represented and they are unique.

**Population Initialization:** In genetic algorithm based solution approach, initial sequences (population) are generated using random generation procedure. The population size, and the method of obtaining initial solutions will affect the convergence of the problem. Genetic algorithms iteratively improve the sequences, hence if the initial sequences are good sequences the convergence of the problem will be faster. The population size is problem-dependent and has to be determined through numerical simulation.

**Genetic Operators:** In genetic algorithms, the basic search mechanism is done by genetic operators. Genetic operators are used to create new sequences based on existing sequences in the population. Crossover, mutation and reproduction are the commonly used genetic operators.

**Crossover Operation:** Crossover operation uses two sequences (search nodes) to produce two new sequences. During crossover operation the parents exchange parts of their solutions. The idea is to combine the good parts of solutions of the parents to produce offsprings. A number of crossovers for combinatorial optimization problems is given in [16]. They are single point crossover, two point crossover, uniform crossover and partially mapped crossover. In our study, we use two-point crossover.

**Mutation Operation:** Mutation operation works on a single sequence to produce a single new sequence. This operation is needed to ensure diversity in the population and to avoid the premature convergence and local minima. In our study, we have used swap mutation presented in [16].

**Fitness Function:** Fitness is the driving force in genetic algorithms. In our scheduling problem, fitness function assigns a fitness value to each of the sequences in a generation. Fitness function must be capable of evaluating every sequence in the search space. For our problem the search space contains all  $n!$  possible sequences. Genetic algorithm does not know anything about the problem domain or fitness function. The only information used in the execution of genetic algorithm is the observed value of fitness for the sequences present in the population. Genetic algorithm is guided by the fitness value to search for the most efficient sequence for the problem. The fitness function for our problem is the objective function, and is minimization of the sum of the deviations. The objective function for our problem is given in (2). The genetic operators will try to maximize the fitness function for a maximization problem. Our scheduling problem is a minimization problem and so needs a transformation of the fitness value. For this purpose, we refine our selection criterion which selects a solution with lower fitness value (objective function value). The selection method used in our scheduling problem is explained below.

**Selection function and Elitism Strategy:** In genetic algorithm, the selection of a sequence from the existing population (sequences) plays an important role. The idea is that better sequence (sequence with better fitness value) should have a better chance of being selected for genetic operations to produce new sequences. In genetic algorithms there are several selection methods such as roulette wheel selection and its extensions, scaling techniques, elitist models and ranking methods are presented. In our study, we have used binary tournament selection method. The criterion of binary tournament selection is to randomly select two sequences (solutions) and compare their fitness values. The solution with better fitness is selected for maximization problems. Since, the scheduling problem is a minimization problem, solution with a lower fitness is selected. The elitism strategy is to select a number of elite solutions from external archive. The number of elite solutions depend on the population size and elitism rate.

**Termination Function:** In a genetic algorithm, in each generation, sequences are selected on the basis of their fitness and subject to genetic operators to produce new sequences for the next generation. The evolution process of successive generations continues until a termination criterion is satisfied. The most frequently used stopping criterion are population convergence criteria and a

specified maximum number of generations. Population convergence criteria for our problem is that all the sequences in the population are the same in two successive generations. This sequence is the best or optimal sequence to the problem. Another stopping criteria is to stop the evolution process when the maximum number of generations is reached. The best sequence is the one in the population with maximum fitness value.

## 4 Simulation Results

The genetic algorithm to obtain the sequence for single-machine scheduling problem with a learning effect is tested with known problems, and with standard test problems from literature.

It is known that the solution obtained from genetic algorithms, for combinatorial optimization problems, can not be guaranteed to be optimal; i.e., no formal proof of optimality. But, for a large class of difficult combinatorial optimization problems, it has been shown that the genetic algorithm produces solution that are close to the optimal or among the best available solutions [16]. So in our studies, we call the sequence obtained from genetic algorithm as “best” sequence instead of optimal sequence. The genetic algorithm to obtain the sequence, for single-machine scheduling problem with a learning effect is implemented in Java on a Pentium-IV machine. The genetic algorithm used the following parameters given in Table.1 in all our simulations.

**Table 1.** Parameters used in simulation

Parameters	Description	Value
$P_m$	Mutation probability	0.30
$P_c$	Crossover probability	0.50
$M$	Maximum number of generations	100
$N$	Population size	100

The following steps are carried out in a genetic algorithm for single-machine scheduling problem with a learning effect.

- STEP.1:** Initialization: An initial population ( $N$ ) sequences are randomly generated.
- STEP.2:** Evaluation: The fitness for each of the sequence in the population is calculated according to the fitness function.
- STEP.3:** Perform selection function using binary tournament selection method and elitism strategy, to select sequences for genetic operations.
- STEP.4:** Genetic Operations: Perform crossover and mutation operations based on probability of crossover and mutation. Here we may get more sequences than the population size ( $N$ ). Perform reproduction using elitist model to obtain  $N$  best sequences.
- STEP.5:** Now, we have the best  $N$  sequences from previous step. Repeat steps 2, 3, and 4, until the algorithm converges.

The advantage with genetic algorithm is that it starts with a random sequences and modify the sequences in successive generations, and the best sequence is obtained. The only information used is the fitness value.

For our problem, this genetic algorithm is used to obtain the sequence of jobs that minimizes the weighted sum of earliness and tardiness costs, with a learning effect. Machine idle times are not considered. For our problem, we have considered three values of  $\alpha = -0.152, -0.322,$  and  $-0.515$ , which corresponds to 90%, 80% and 70% learning curves respectively.

For the single machine scheduling problem is considered in this study, many test problems are provided with job dependent earliness and tardiness penalties in [23]. These test problems are generated in the following manner and are available in the internet. For a given value of  $n$  (number of jobs), the processing times are generated randomly from the uniform distribution  $U = [10, 100]$ . The due dates  $d_j$  are generated from  $U = [d_{min}, d_{min} + \rho P]$ , where  $d_{min} = \text{Max}\{0, P(\tau - \rho/2)\}$  and  $P = \sum_{j=1}^n P_j$ . The two parameters  $\tau$  and  $\rho$  are tardiness and range parameters. These test problems are available in [23] for  $n \in \{20, 30, 40, 50\}$ ,  $\tau \in \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$  and for the value of  $\rho \in \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ . The earliness ( $\alpha_j$ ) and tardiness ( $\beta_j$ ) for all jobs are generated randomly from the uniform distribution  $U = [1, 5]$ .

We have considered the situation when the number of jobs is 20, 30, 40 and 50, with a learning effect. These problems are taken from [23], and the problem numbers are *sks222*, *sks322*, *sks422* and *sks522*. Genetic algorithm is able to obtain the "best" sequence in a very short time. The objective function value and the computation time required by our genetic algorithm is given below in Table.2 for learning rates  $\alpha = -0.515$ ,  $\alpha = -0.322$ , and  $\alpha = -0.152$ . In our simulations, for each of these problems, the genetic algorithm was run 20 times and the average computation time is given in Table.2. Also, the minimum, mean and maximum value of the objective function value obtained from these 20 runs are given in Table.2. From this Table.2, we see that our genetic algorithm approach, takes only a small computation time to obtain the best sequence. The source code for the genetic algorithm and the results obtained are given in the website [24].

The parameters used in genetic algorithms  $P_m$ ,  $P_c$ , and  $N$  are usually determined by trial-and-error. We have tried with different values of crossover probability ( $P_c$ ), and different values of population size ( $N$ ). We obtain the same optimal sequence in our simulation and the change in computation time is not significant.

We have also used branch and bound technique for this problem, and noticed that the computational time depends on the value of  $\rho$  and  $\tau$ . This is because of the number of nodes in the branch and bound technique. For example, when  $\tau=0.2$  and  $\rho=0.2$ , the branch and bound computation time for 50 jobs is 48.38, and the computation time for genetic algorithm is 3.5 seconds. It is important to note that the branch and bound technique obtains the optimal sequence, but the sequence obtained from genetic algorithm can not be guaranteed to be optimal; i.e., no formal proof of optimality. But, for a large class of combinatorial optimization problems, it is shown that the genetic algorithm produces solution that are optimal or close to optimal solution.

**Table 2.** Objective function and computation time for different Learning rates

Learning rate	number of jobs	Minimum	Objective function value Mean	Maximum	Computation time
$\alpha = -0.515$	20	743.16	753.14	792.43	0.99223
$\alpha = -0.515$	30	1260.90	1317.40	1369.90	1.58460
$\alpha = -0.515$	40	1351.50	1358.00	1379.00	2.37970
$\alpha = -0.515$	50	2081.70	2115.50	2140.30	3.41820
$\alpha = -0.322$	20	1748.80	1750.90	1767.20	0.98490
$\alpha = -0.322$	30	3400.60	3458.50	3626.20	1.57800
$\alpha = -0.322$	40	5602.10	5672.20	5794.90	2.38380
$\alpha = -0.322$	50	4454.00	4525.60	4675.80	3.41620
$\alpha = -0.152$	20	3287.10	3291.10	3319.20	0.98477
$\alpha = -0.152$	30	6823.00	6827.30	6862.80	1.56870
$\alpha = -0.152$	40	13726	13786	13890	2.35200
$\alpha = -0.152$	50	13966	14000	14145	3.40210

## 5 Conclusions

In this paper, we consider The single machine scheduling problem, in which, each job has an individual due-date, earliness and tardiness penalties is considered in presence of a learning effect. Because of the learning effect, the processing time of a job depends on its position in the sequence. The objective is to find the sequence of jobs that minimizes the weighted sum of earliness and tardiness costs, without considering machine idle times with a learning effect. Since, this is a difficult combinatorial optimization problem, we have used genetic algorithm approach to obtain the best/optimal sequence. We shown via numerical examples that the genetic algorithm approach takes only small computation time to solve fairly large problems (50 jobs).

## References

1. Biskup, D.: Single-machine scheduling with learning considerations. *European Journal of Operational Research* **115** (1999) 173-178
2. Mosheiov, G.: Scheduling problems with a learning effect. *European Journal of Operational Research* **132** (2001) 687-693
3. Lee, W.C., Wu, C.C.: Minimizing total completion time in a two-machine flowshop with a learning effect. *International Journal of Production Economics* **88** (2004) 85-93
4. Mosheiov, G.: Parallel machine scheduling with a learning effect. *Journal of the Operational Research Society* **52** (2001) 1165-1169
5. Wang, J.B., Xia, Z.Q.: Flow-shop scheduling with a learning effect. *Journal of the Operational Research Society* **56** (2005) 1325-1330
6. Mosheiov, G., Sidney J.B.: Note on scheduling with general learning curves to minimize the number of tardy jobs. *Journal of the Operational Research Society* **56** (2005) 110-112

7. Biskup, D., Simons, D.: Common due date scheduling with autonomous and induced learning. *European Journal of Operational Research* **159** (2004) 606-616
8. Bachman, A., Janiak, A.: Scheduling jobs with position-dependent processing times. *Journal of the Operational Research Society* **55** (2004) 257-264
9. Kuo, W.H., Yang, D.L.: Minimizing the makespan in a single machine scheduling problem with a time-based learning effect. *Information Processing Letters* **97** (2006) 64-67
10. Du, J., Leung, J.Y.T.: Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research* **15** (1990) 483-495
11. Cheng, T.C.E., Ding, Q., Kovalyov, M.Y., Bachman, A., Janiak, A.: Scheduling jobs with linearly decreasing processing times. *Naval Research Logistics* **50** (2003) 531-555
12. Cheng, T.C.E., Ding, Q., Lin, B.M.T.: A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research* **152** (2004) 1-13
13. Holland, H.J.: *Adaption in natural and artificial systems*. University of Michigan Press, Ann Arbor, USA (1975)
14. Goldberg, D.E.: *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, New York, USA (1989)
15. David, L.: *Handbook of genetic algorithms*. Van Nostrand Reingold, New York, USA (1991)
16. Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*. AI Series Springer-Verlag, New York, USA (1994)
17. Chaudhry, S.S., Luo, W.: Application of genetic algorithms in production and operations management: a review. *International Journal of Production Research* **19** (2005) 4083-4101
18. Ow, P.S., Morton, E.T.: The single machine early/tardy problem. *Management Science* **35** (1989) 171-191
19. Azizoglu, M., Kondakci, S., Krica, O.: Bicriterion scheduling problems involving total tardiness and total earliness penalties. *International Journal of Production Economics* **23** (1991) 17-24
20. Su, L.H., Chang, P.C.: A heuristic to minimize a quadratic function of job lateness on a single machine. *International Journal of Production Economics* **55** (1998) 169-175
21. Su, L.H., Chang, P.C.: Scheduling n jobs on one machine to minimize the maximum lateness with a minimum number of tardy jobs. *Computers and Industrial Engineering* **40** (2001) 349-360
22. Nadler, G., Smith, W.D.: Manufacturing progress functions for types of processes. *International Journal of Production Research* **2** (1963) 115-135
23. Sourd, F.: <http://www-poleia.lip6.fr/~sourd/>
24. Chang, P.C.: <http://ppc.iem.yzu.edu.tw/publication/sourceCode/SingleMachineLearningEffect/>