An Evaluation of Sequencing Heuristics for Flexible Flowshop Scheduling Problems with Unrelated Parallel Machines and Dual Criteria

Jitti Jungwattanakit[a], Manop Reodecha[a], Paveena Chaovalitwongse[a], Frank Werner[b,*]

[a]Department of Industrial Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok 10330 Thailand
[b]Faculty of Mathematics, Otto-von-Guericke-University, P.O. Box 4120, D-39016 Magdeburg, Germany

Abstract

This paper deals with the heuristic solution of flexible flowshop scheduling problems with unrelated parallel machines. A setup time is necessary before starting the processing of a job, where the setup time depends on the previous job. No preemption of jobs is allowed. As objective function, this paper considers the minimization of the positively weighted convex sum of makespan and the number of tardy jobs. This paper develops some well-known constructive heuristics for the pure flowshop scheduling problems such as the algorithms given by Palmer (1965), Campbell, Dudek, and Smith (1970), Gupta (1971), and Dannenbring (1977) as well as the insertion heuristic by Nawaz, Enscore and Ham (1983) to the flexible flowshop environment. By using one of these heuristics, the first stage sequence is generated. This sequence will then be used, in conjunction with either FIFO or permutation rules, to construct a schedule for the overall problem. The final solution is then the best schedule obtained by one of the two rules. Furthermore, some iterative heuristics such as a genetic algorithm and a simulated annealing approach are proposed to increase the quality of the constructive solution. Detailed computational results are presented to evaluate the efficiency of the heuristic algorithms. Detailed computational results with the algorithms are presented.

Keywords: flexible flowshop; heuristics; NEH; genetic algorithm; simulated annealing

1. Introduction

Production scheduling can be defined as the allocation of available production resources over time to perform a collection of tasks (Baker, 1974). It is an important decision making process in the operation level. In a modern manufacturing environment, many scheduling problems occur. Most of the scheduling problems are very significant and hard to solve owing to the complex nature of the problems.

This paper is primarily concerned with industrial scheduling problems, where one first has to assign scarce resources to jobs and then to sequence the assigned jobs on each resource over time. It is mainly concerned with process industries like e.g. the glass-container (Paul, 1979), rubber (Yanney and Kuo, 1989), photographic film (Tsubone, Ohba, Takamuki, and Miyake, 1993), steel (Finke and Medeiros, 2002), textile, and food industries.

* Corresponding author Fax +49 391 67 11171
E-mail address: Frank.Werner@Mathematik.Uni-Magdeburg.DE

In these industries, production facilities are established as multistage production flowshop facilities, where a production stage may be made up of parallel production lines, machines or any other production facility. At some stages, the facilities (machines, lines, etc.) are duplicated in parallel in order to increase the overall capacities of the shop floor, or in order to balance the capacities of the stages, or either to eliminate or reduce the impact of bottleneck stages on the overall shop floor capacities known as flexible flowshop, multiprocessor flowshop, or hybrid flowshop environment.

A flexible flowshop environment is a generalization of the classical flowshop model. There are *k* stages and some stages may have only one machine, but at least one stage must have multiple machines. The jobs have to visit the stages in the same order string from stage one through stage *k*. A machine can process at most one job at a time and a job can be processed by at most one machine at a time. Preemption of processing is not allowed. The problem consists of assigning jobs to machines at each stage and sequencing the jobs assigned to the same machine so that some optimality criteria are minimized.

Flexible flowshop models differ in the type and number of machines at the stages. The processing time of a job on the different machines of every stage may be identical, uniform, or unrelated. These three possibilities correspond to the three classical parallel machine models with similar notations.

Let p_{ij}^t denote the processing time of job *j* on machine *i* at stage *t*, ps_j^t be the standard processing time of job *j* at stage t, and v_{ij}^t is the relative speed of machine *i* for job *j* at stage *t*. In general, one can distinguish between the following three cases:

1. identical parallel machines: $p_{ij}^t = ps_j^t$ for all *i* and *j*;
2. uniform parallel machines: $p_{ij}^t = \frac{ps_j^t}{v_i^t}$ for all *i* and *j*, where v_i^t is the relative speed of machine *i* at stage *t*, and $v_i^t = v_{ij}^t$ for all *j*;
3. unrelated parallel machines: $p_{ij}^t = \frac{ps_j^t}{v_{ij}^t}$ for all *i* and *j*.

For the past three decades, the flexible flowshop scheduling problem has attracted many researchers. Numerous research articles have been published on this topic (see e.g. Wang, 2005). There are two main reasons for this, among many others. First, a flexible flowshop environment is a category of machine scheduling problems which is difficult to solve (Garey and Johnson, 1979; Gupta, 1988; Pinedo, 1995). The flexible flowshop problem which has two stages, with one stage having at least two machines, has already been proved to be NP-hard (Hoogeveen, Lenstra, and Veltman., 1996). Thus, it is unlikely that polynomial time algorithms exist for the exact solution of the general problem. Second, the machine scheduling problem can find many real-world applications.

Although the flexible flowshop problem has been widely studied in the literature, most of the studies related to flexible flowshop problems are concentrated on problems with identical processors, see for instance, Gupta, Krüger, Lauff, Werner and Sotskov (2002), Alisantoso, Khoo, and Jiang (2003), Lin and Liao (2003) and Wang and Hunsucker (2003). In the real world situation, it is common to find newer or more modern machines running side by side with older and less efficient machines. Even though the older machines are less efficient, they may be kept in the production lines because of their high replacement costs. The older machines may perform the same operations as the newer ones, but would generally require longer operating time for the same operation. In this paper, the flexible flowshop problem with unrelated parallel machines is however considered, that is, there are different parallel machines at every stage, and speeds of the machines are dependent on the jobs. Moreover, several industries encounter sequence-dependent setup times which result in even more difficult scheduling problems. Therefore in this paper, sequence-dependent setup time restrictions are taken into account as well.

The purpose of this paper is to present and compare several constructive and iterative heuristics to solve the flexible flowshop scheduling problem with unrelated parallel machines and sequence-dependent setup times. In addition, while many papers have studied only the makespan criterion, we consider this scheduling problem with two optimization criteria. One reason for this consideration is the increasing pressure of high competition while customers expect ordered goods to be delivered on time. The remainder of this paper is as follows. In Section 2, the problem considered in this paper is described and the notations needed are introduced. Section 3 presents an algorithm for the heuristic construction of a schedule for the unrelated parallel machine problem provided that a job sequence for the first processing stage is known. Section 4 gives constructive heuristics for the determination of the first-stage job sequence for the flexible flowshop problem while Section 5 discusses some iterative heuristics. A comparison of all heuristics is presented in Section 6, and some conclusions are given in Section 7.

2. Problem Description

We consider the problem of scheduling a flexible flowshop environment with unrelated parallel machines taking sequence-dependent setup times into account as well. The addressed scheduling problem can be described as follows. Consider a main queue of incoming n jobs (j = 1, 2, …, n), a k stage (t = 1, 2, …, k) flowshop scheduling problem, each stage of which has m^t unrelated parallel machines in the system. Each job can advance to any one of the m^1 machines at stage one. In general, each job can be routed to any one of the m^t machines at stage t. When the job has been processed by the last stage k, using one of the m^k machines, it is completed, and at that time, the job can leave the system. The machines are continuously available from time zero onwards and can handle at most one job at a time. No preemption is allowed. Each job, ready at time zero, needs to be scheduled exactly on one machine of each of the k stages.

Let the completion time of job j be C_j, then

$$C_{max} = \max_{j\in\{1..n\}} \{ C_j \}.$$

Associated with each job j is a due date $d_j \geq 0$. Let $U_j = 1$ if due date for job j is smaller than the completion time C_j of job j, otherwise $U_j = 0$. The total number of tardy jobs (η_T) is defined as

$$\eta_T = \sum_{j=1}^{n} U_j$$

The objective is to seek a schedule that minimizes a positively weighted convex sum of makespan and the number of tardy jobs. Thus, the objective function value for this research is defined by

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

where $0 \leq \lambda \leq 1$. λ denotes the weight (or relative importance) given to C_{max} and η_T.

Before proceeding to the next section, some notations used throughout this paper are defined as follows:

Indices:

t	stage index, t = 1, 2, 3, . . . , k
i	machine index, i = 1, 2, 3, . . . , m^t
j, l	job index, j, l = 1, 2, 3, . . . , n

Parameters:

m^t	number of parallel machines at stage t
d_j	due date of job j
s^t_{lj}	setup time from job l *to* job j at stage t
ps^t_j	standard processing time of job j at stage t
v^t_{ij}	relative speed of machine i at stage t for job j
p^t_{ij}	processing time of job j on machine i at stage t; where $p^t_{ij} = \dfrac{ps^t_j}{v^t_{ij}}$
O^t_j	operating time of job j at stage t

Variables:

C_j^t	completion time of job *j* at stage *t*
C_{max}	the makespan, defined as max (C_1 , … , C_n), is equivalent to the completion time of the last job to leave the system
U_j	a Boolean variable equal to 1 if job *j* is tardy, and 0 otherwise
η_T	the total number of tardy jobs in the schedule

3. Heuristic construction of a schedule

Since the flexible flowshop scheduling problem is *NP-hard* (Garey and Johnson, 1979; Gupta, 1988; Pinedo, 1995), algorithms for finding an optimal solution in polynomial time are therefore unlikely to exist. Thus, heuristic methods are studied to find approximate solutions. Most researchers develop existing heuristics for the classical flexible flowshop problem with identical machines by using a particular sequencing rule for the first stage. They follow the same scheme (see Santos, Hunsucker and Deal, 1996). Firstly, a job sequence is determined according to a particular sequencing rule using modified flowshop algorithms. This problem will be considered in Section 4. 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 basically two approaches for this subproblem. The first way is that for the other stages, i.e. from stage two to stage *k*, jobs are ordered according to their completion times at the previous stage. This means that the FIFO (First in First out) rule is used to find the job sequence for the next stage by means of the job sequence of the previous stage. The second way is to sequence the jobs for the other stages by using the same job sequence as for the first stage. In other words, the permutation flexible flowshop sequencing problem determines the order of processing jobs on all machines. We will make use of both procedures (see Algorithm 2) and discuss the modifications for the problem under consideration.

In the rest of this section, we always assume that a job sequence for the first stage has already been determined. This section deals with the problem of scheduling *n* jobs on unrelated parallel machines with sequence-dependent setup times using this given job sequence for the first stage. First, we present a greedy algorithm which constructs a schedule for the *n* jobs at a particular stage provided that a certain job sequence for this stage is given (see Algorithm 1). The objective is to minimize the flowtime and minimize the idle time of the machines. In the following algorithm, the dot refers to an arbitrary stage.

Algorithm 1

Input: job sequence π=(π[1], π[2], …, π[*n*]) for some stage.

Step 1: For every machine *i,* set the available time of each machine to be zero: *av*[*i*] = 0, $i \in \{1, \ldots, m\}$. Let *u* = 1.

Step 2: For $j = \pi(u)$ do the following:

Step 2.1: Let $ps_j^\bullet$ be the standard possessing time of job *j*, $r_j^\bullet$ be the available time (i.e. release date of job *j* for the particular stage), $v_{ij}^\bullet$ be the relative speed of the machine *i* for job *j*, and $s_{\bullet j}^\bullet$ be the setup time on a particular machine provided that the previous job on this machine is changed to job *j*.

Step 2.2: For every machine *i,* determine the completion time $C_{ij}^/$ of job *j* on machine *i* by using the following equation:

$$C_{ij}^/ = \max\{r_j^\bullet, av[i]\} + \frac{ps_j^\bullet}{v_{ij}^\bullet} + s_{\bullet j}^\bullet ;$$

Step 2.3 If there is only machine i^* that gives the minimum completion time $C^* = C_{i^*j}^/$ of job *j*, then assign job *j* to machine i^* and proceed to Step 2.6.

Step 2.4 For every machine *i* that gives the minimum completion time C^* of job *j*, calculate the idle time w_{ij} of this machine if job *j* is assigned to it by using the following formula:

$$w_{ij} = C_{ij}^/ - av[i] - \frac{ps_j^\bullet}{v_{ij}^\bullet} - s_{\bullet j}^\bullet$$

```
    Step 2.5  Select a machine $i^*$ with smallest idle time $w_{i^*j}$ of job $j$.
    Step 2.6  Update the available time of machine $i^*$: $av[i^*] = C^{/}_{i^*j}$.
    Step 2.7  Store the completion time $C_j$ of job $j$ be equal to $C^{/}_{i^*j}$.
    Step 2.8  $u = u + 1$. If $u \leq n$, go to Step 2. Otherwise, stop.
```

The idea of the heuristic is to assign the jobs to the machines evenly in order to balance the workload as much as possible heuristically.

The following Algorithm 2 constructs a for a given job sequence ω^1 for the first stage a complete schedule for the problem under consideration. It considers both the FIFO and the permutation rules to fix the job sequences for stages *2, 3, …, k* by means of the job sequence for the first stage.

Algorithm 2

```
Input: job sequence $\omega^1 = (\omega^1[1], \omega^1[2], \ldots, \omega^1[n])$ for the first stage.
Step 1: Use the FIFO rule to construct a schedule.
    Step 1.1: Set $t = 1$.
    Step 1.2: For stage $t$, set $\pi = \omega^t$
    Step 1.3: Set $(ps^\bullet_j, r^\bullet_j, v^\bullet_{ij}, s^\bullet_{\bullet j}) = (ps^t_j, r^t_j, v^t_{ij}, s^t_{\bullet j})$.
    Step 1.4: Use Algorithm 1 to find the schedule for stage $t$.
    Step 1.5: Set $r^{t+1}_j = C_j$ ($C_j$ according to Step 2.7 in Algorithm 1).
    Step 1.6: Determine $\omega^{t+1}$ such that completion time of job $\omega^{t+1}[j] \leq$ completion time of job $\omega^{t+1}[j+1]$,
              $j \in \{1, .., n-1\}$.
    Step 1.7: If $t < k$ ($k$ is the number of stages), then $t = t+1$ and go to Step 1.2, otherwise go to Step 2.
Step 2: Use the permutation rule to construct a schedule.
    Step 2.1: Set $\omega^1 = \omega^2 = \ldots = \omega^k = (\omega^1[1], \omega^1[2], \ldots, \omega^1[n])$ and $t = 1$.
    Step 2.2: For stage $t$, set $\pi = \omega^t$.
    Step 2.3: Set $(ps^\bullet_j, r^\bullet_j, v^\bullet_{ij}, s^\bullet_{\bullet j}) = (ps^t_j, r^t_j, v^t_{ij}, s^t_{\bullet j})$.
    Step 2.4: Use Algorithm 1 to find the schedule for stage $t$.
    Step 2.5: Set $r^{t+1}_j = C_j$ ($C_j$ according to Step 2.7 in Algorithm 1).
    Step 2.6: If $t < k$, then $t = t+1$ and go to Step 2.2, otherwise go to Step 3.
Step 3: Return the best solution.
```

Therefore, only the job sequence used for the first stage is important for this approach. 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. This algorithm is as follows.

The representatives of machine speed $v^{/t}_{ij}$ and setup time $s^{/t}_{lj}$ use the minimum, maximum and average values of the data. Thus, the representative of operating time $O^{/t}_j$ of job *j* at stage *t* is the sum of the processing time $ps^t_j / v^{/t}_{ij}$ plus the representative of the setup time $s^{/t}_{lj}$. Nine combinations of relative speeds and setup times will be used in the suggested algorithm. 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 (see Algorithm 3).

Algorithm 3

```
Input: Relative speeds and setup times.
Step 1: Determine the representatives of relative speeds and setup times for $t=1,\ldots,k$.
```

Step 1.1: a. If *speed* =1, then $v_{ij}^{/t} = \min\{ v_{ij}^{t}; i \in \{1,...,m^t\}, j \in \{1,...,n\} \}$

b. If *speed* = 2, then $v_{ij}^{/t} = \max\{ v_{ij}^{t}; i \in \{1,...,m^t\}, j \in \{1,...,n\} \}$

c. If *speed* = 3, then $v_{ij}^{/t} = \text{average}\{ v_{ij}^{t}; i \in \{1,...,m^t\}, j \in \{1,...,n\} \}$

Step 1.2: a. If *setup* = 1, then $s_{lj}^{/t} = \min\{ s_{lj}^{t}; l \in \{1,...,n\}, l \neq j \}$

b. If *setup* = 2, then $s_{lj}^{/t} = \max\{ s_{lj}^{t}; l \in \{1,...,n\}, l \neq j \}$

c. If *setup* = 3, then $s_{lj}^{/t} = \text{average}\{ s_{lj}^{t}; l \in \{1,...,n\}, l \neq j \}$

Step 2: For every speed, determine the first-stage job sequence and a complete schedule.

Step 2.1: Set *speed* = 1.

Step 2.2: Set *setup* = 1.

Step 2.3: For every stage *t* (*t=1,…,k*) and job *j* (*j=1,…,n*), determine the representative of the operating time $O_j^{/t}$ by using the following equation:

$$O_j^{/t} = \frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t}$$

Step 2.4: Find the sequence ω^1 of the first stage, by using the representatives of the operating times from Step 2.3 to find the sequence of the first stage with some modified flowshop heuristic.

Step 2.5: Assign all jobs at stage *t* (*t=1,…,k*) to some machines by using ***Algorithm 2***.

Step 2.6: If *setup* < 3, then *setup* = *setup* + 1 and go to Step 2.3.

Step 2.7: If *speed < 3,* then *speed* = *speed* + 1 and go to Step 2.2 else go to Step 3.

Step 3: Return best solution.

It remains to discuss how the job sequence for the first stage is found in Step 2.4. This is done in Section 4 in detail.

4. Constructive algorithms

In this section we present some constructive algorithms for determining the job sequence for the first stage for the problem considered. This extends existing algorithms mostly developed for the flexible flowshop problem with identical parallel machines without setup time considerations. Using then Algorithms 1 – 3 from Section 3, a heuristic schedule for the problem under consideration is found.

We adapt and develop several constructive algorithms for the pure flowshop scheduling problem. As flowshop heuristics, we use the algorithms given by Palmer (1965), Campbell, Dudek, and Smith (1970), Gupta (1971), and Dannenbring (1977) and the insertion heuristic by Nawaz, Enscore, and Ham (1983). Next, we discuss the generalizations of these heuristics to the problem under consideration.

4.1 Palmer

A heuristic developed by Palmer (1965), in an effort to use Johnson's rule, is built around the notion of a slope index. The slope index gives a large value to jobs that have a tendency of progressing from small to large operating times as they move through the stages. The sequence of operations is given by priority to jobs having the strongest tendency to progress from short times to long times. This means that the job sequence can be generated based upon an non-increasing order of the slope indices.

Let *S* (*j*) be the slope index for job *j* and O_j^t be the operating time of job *j* at stage *t.* Palmer's slope index is calculated as follows:

$$S(j) = -\sum_{t=1}^{k} \left\{ [k - (2t-1)] O_j^t \right\}$$

To illustrate Palmer's method for use in the flowshop environment, the example given in Table 1 will be utilized.

Table 1
Standard processing times and due date for every job of a three-stage flowshop problem

Job j	1	2	3	4	5
ps_j^1	11	13	20	9	11
ps_j^2	12	15	10	12	18
ps_j^3	20	18	12	20	15
d_j	40	40	75	45	60

Since there is only one machine for every stage, the operating times for every job are equal to its standard processing times ($O_j^t = ps_j^t$). The slope indices for the five jobs are now calculated as follows:

$$S(j) \quad = - [2O_j^1 + 0O_j^2 - 2O_j^3], \text{ i.e.}$$

1. $S(1) = -2(11) - 0(12) + 2(20) = 18;$
2. $S(2) = -2(13) - 0(15) + 2(18) = 10;$
3. $S(3) = -2(20) - 0(10) + 2(12) = -16;$
4. $S(4) = -2(9) - 0(12) + 2(20) = 22;$
5. $S(5) = -2(11) - 0(18) + 2(15) = 8.$

Palmer's heuristic sequences the jobs in non-increasing order of the slope indices. For the job set of Table 1, this heuristic yields the sequence $\pi = (4, 1, 2, 5, 3)$ for the first stage. Palmer's heuristic yields a makespan value of 106. Using a 0-1 mixed integer programming formulation, one can confirm that 106 is the optimal makespan value as well.

Now, Palmer's method for the flexible flowshop problem with unrelated parallel machines and sequence-dependent setup times is developed as follows. Let $S(j, v_{ij}^{/t}, s_{lj}^{/t})$ be the slope index for job j at relative speed $v_{ij}^{/t}$ and setup time $s_{lj}^{/t}$, ps_j^t be the standard processing time of job j at stage t, $v_{ij}^{/t}$ be the representative of the relative speed on machine i at stage t for job j, and $s_{lj}^{/t}$ be the representative of the setup time from job l to job j at stage t. Palmer's slope index for the flexible flowshop problem with unrelated parallel machines and setup times is calculated as follows:

$$S(j, v_{ij}^{/t}, s_{lj}^{/t}) = -\sum_{t=1}^{k}\left\{[k-(2t-1)]\left(\frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t}\right)\right\}$$

Since the processing times and the setup times for every job are dependent on the machine and the previous job, respectively, the representative of the operating time $O_j^{/t}$ of job j at stage t in Palmer's method adaptation is the sum of the processing time $ps_j^t / v_{ij}^{/t}$ plus the setup time $s_{lj}^{/t}$. To construct a schedule for the overall problem, we set $v_{ij}^{/t}$ to be the minimum, maximum, and average relative speeds, and $s_{lj}^{/t}$ to be the minimum, maximum, and average setup times regardless for the machine and the previous job. All these nine combinations of relative speeds and setup times will be used as has been described in Algorithm 3 and finally, the solution with the best objective function value obtained by all different relative speeds and setup times is taken.

4.2 Campbell, Dudek, and Smith

Campbell, Dudek, and Smith (1970) develop one of the most significant heuristic methods for flowshop problems with makespan criterion, in the following denoted by CDS. Its strength lies in two properties: (1) it uses Johnson's rule in a heuristic fashion, and (2) it generally creates several schedules from which a "best" schedule can be chosen. In so doing, $k - 1$ sub-problems are created and Johnson's rule is applied to each of the sub-problems. Thus, $k - 1$ job sequences are generated. Since Johnson's algorithm is a two-stage algorithm, a k-stage problem must be collapsed into a two-stage problem. Let g be a counter for the $k - 1$ sub-problems, the operating times for the "first" stage are denoted as $a(j, g)$, where j denotes the job and g denotes the g-th sub-problem. Similarly, $b(j, g)$ denotes the "second" stage operating times of job j and sub-problem g. Given these notations, the operating times are calculated by the following two formulas:

$$a(j,g) = \sum_{t=1}^{g} O_j^t$$

and

$$b(j,g) = \sum_{t=k-g+1}^{k} O_j^t$$

For each of the sub-problems, Johnson's algorithm provides a job sequence using the values $a(j, g)$ and $b(j, g)$. Once Johnson's sequence is created, the problem is then returned to the consideration of all k stages.

Under the CDS adaptation rule, the operating times for the flexible flowshop problem with unrelated parallel machines and setup times are calculated as follows:

$$a(j,g,v_{ij}^{/t},s_{lj}^{/t}) = \sum_{t=1}^{g} \left(\frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t}\right)$$

and

$$b(j,g,v_{ij}^{/t},s_{lj}^{/t}) = \sum_{t=k-g+1}^{k} \left(\frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t}\right)$$

To generate the first stage, Johnson's ordering is created, and the problem is then returned to the consideration of k stages by calling Algorithm 2 for all nine combinations of relative speeds and setup times considered in Algorithm 3.

4.3 Gupta

Gupta (1971) provides an algorithm which, in a similar manner as Palmer's algorithm, uses a slope index. Denote $G(j)$ as the slope index generated by Gupta's method for job j. Then $G(j)$ is calculated as follows:

$$G(j) = \frac{e_j}{\min\limits_{1 \le g \le k-1} \{O_j^g + O_j^{g+1}\}};$$

where

$$e_j = \begin{cases} 1 & \text{if} \quad O_j^1 > O_j^k \\ -1 & \text{if} \quad O_j^1 \le O_j^k \end{cases}.$$

After calculating $G(j)$ for all jobs, the jobs are subsequently ranked in a non-decreasing order of the slope indices.

Under the Gupta adaptation rule, let $G(j,v_{ij}^{/t},s_{lj}^{/t})$ be the slope index for job j at relative speed $v_{ij}^{/t}$ and setup time $s_{lj}^{/t}$. Gupta's slope index for the flexible flowshop with unrelated parallel machines and setup times is then calculated as follows:

$$G(j,v_{ij}^{/t},s_{lj}^{/t}) = \frac{e_j}{\min\limits_{1 \le g \le k-1} \left\{\left(\frac{ps_j^g}{v_{ij}^{/g}} + s_{lj}^{/g}\right) + \left(\frac{ps_j^{g+1}}{v_{ij}^{/g+1}} + s_{lj}^{/g+1}\right)\right\}}$$

with $$e_j = \begin{cases} 1 & if \quad (\frac{ps_j^1}{v_{ij}^{/1}} + s_{lj}^{/1}) > (\frac{ps_j^k}{v_{ij}^{/k}} + s_{lj}^{/k}) \\ -1 & if \quad (\frac{ps_j^1}{v_{ij}^{/1}} + s_{lj}^{/1}) \le (\frac{ps_j^k}{v_{ij}^{/k}} + s_{lj}^{/k}) \end{cases}$$

4.4 Dannenbring

Like Palmer's rule, Dannenbring (1977) develops a method by using Johnson's algorithm as a foundation. Furthermore, algorithms of CDS and Palmer are also exhibited. Dannenbring constructs only one two-stage problem, but the operating times for the jobs reflect the behavior of Palmer's slope index. Denote *a*(*j*) and *b*(*j*) as the operating times for the constructed two-stage problem. The calculations of *a*(*j*) and *b*(*j*) are as follows:

$$a(j) = \sum_{t=1}^{k}(k-t+1)(O_j^t)$$

and

$$b(j) = \sum_{t=1}^{k} t \times O_j^t$$

After calculating *a*(*j*) and *b*(*j*) for all jobs, the jobs are subsequently ranked by applying Johnson's algorithm to generate the job sequence for stage one.

Under the Dannenbring adaptation rule, the operating times for the flexible flowshop problem with unrelated parallel machines and setup times are calculated as follows:

$$a(j, v_{ij}^{/t}, s_{lj}^{/t}) = \sum_{t=1}^{k}\left\{(k-t+1)(\frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t})\right\}$$

and

$$b(j, v_{ij}^{/t}, s_{lj}^{/t}) = \sum_{t=1}^{k}\left\{t \times (\frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t})\right\}.$$

4.5 NEH Heuristic

Nawaz, Enscore and Ham (1983) develop the probably best constructive heuristic method for the permutation flowshop problem, called NEH algorithm. It 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.

1. The total operating times for job *j* is calculated using the formula
   $$P_j = \sum_{t=1}^{k} O_j^t, \; j = 1, \ldots, n.$$
2. The *n* jobs are sorted in non-increasing order of their total operating time P_j on the machines. Then the first two jobs (those with largest P_j) are taken and the two possible schedules containing them are evaluated. The sequence with better objective function value is taken for further consideration.
3. Take every remaining job in the sorted list calculated in Step 2 and find the best schedule by placing it at all possible positions in the sequence of jobs that are already scheduled. For example, if (*j*1, *j*2, *j*3) is the current sequence of scheduled jobs and job *r* is the remaining job with largest P_r in the sorted list, then job *r* could be placed at four positions: (*r*, *j*1, *j*2, *j*3), (*j*1, *r*, *j*2, *j*3), (*j*1, *j*2, *r*, *j*3) or (*j*1, *j*2, *j*3, *r*). The sequence with best objective function value among the four considered is selected for further extension.

To apply the NEH algorithm to the flexible flowshop problem with unrelated parallel machines, the total operating times for calculating the job sequence for the first stage are calculated according to Step 2.3 of Algorithm 3 for the nine combinations of relative speeds of machines and setup times.

4.6 Illustrative Example

In order to illustrate the application of the proposed procedures, let $n=5$, $k=3$ and $m^1=m^2=m^3=3$. Table 2 shows the standard processing times and the due date of every job. Tables 3a -3c and Tables 4a - 4c show the matrix of setup time for every stage and the relative speed for every machine.

Table 2
Standard processing times and due date of every job of the three-stage flexible flowshop

Job j	1	2	3	4	5
ps_j^1	16	17	10	18	15
ps_j^2	16	15	9	18	14
ps_j^3	16	15	17	20	8
d_j	40	40	75	45	60

Table 3a
The matrix of setup times from job j to job l at stage 1

to from	1	2	3	4	5
0	0	0	0	0	0
1	-	0	2	1	0
2	2	-	2	0	4
3	4	0	-	3	2
4	1	1	2	-	1
5	2	2	0	2	-

Table 3b
The matrix of setup times from job j to job l at stage 2

to from	1	2	3	4	5
0	0	0	0	0	0
1	-	4	0	1	3
2	5	-	5	2	4
3	4	2	-	0	3
4	3	2	4	-	2
5	0	1	2	1	-

Table 3c
The matrix of setup times from job j to job l at stage 3

to from	1	2	3	4	5
0	0	0	0	0	0
1	-	2	0	4	2
2	3	-	1	3	2
3	2	1	-	3	4
4	2	3	1	-	2
5	1	0	2	1	-

Table 4a
The relative speed of every machine (v_{ij}^1) at stage 1

job *j* / machine *i*	1	2	3	4	5
1	0.85	0.83	1.02	1.03	1.1
2	1.09	0.86	0.94	0.94	1.09
3	1.04	0.92	1.15	0.86	1.05

Table 4b
The relative speed of every machine (v_{ij}^2) at stage 2

job *j* / machine *I*	1	2	3	4	5
1	1.12	1	1.04	1.07	1.03
2	1.1	0.81	0.91	1	1.01
3	0.87	1	0.84	0.81	1

Table 4c
The relative speed of every machine (v_{ij}^3) at stage 3

job *j* / machine *i*	1	2	3	4	5
1	1.02	0.96	0.97	1.05	1.04
2	0.84	0.81	0.92	1.08	1.03
3	0.84	1.08	1	0.87	1.1

For this example, the optimal solutions for every $\lambda \in \{0, 0.5, 1\}$ by using e.g. a 0-1 mixed integer programming code are shown in Table 5. All proposed heuristics generate an optimal schedule for the λ values considered.

Table 5
Results of the calculation of an optimal schedule

λ	Schedule	Value	*Cmax*	η_T
0	stage 1: 4 – 2 // 1 // 3 – 5 stage 2: 3 – 5 – 1 // 4 // 2 stage 3: 1 // – // 3 – 5 – 2	3	93.84	3
0.5	stage 1: 4 // 1 – 5 // 3 – 2 stage 2: 3 – 4 // 1 – 5 // 2 stage 3: 1 – 5 // 4 // 3 – 2	30.03	57.06	3
1	stage 1: 4 // 1 – 5 // 3 – 2 stage 2: 1 – 5 // 3 – 4 // 2 stage 3: 1 – 5 // 4 // – 3 – 2	57.06	57.06	3

5. Iterative algorithms

In this section, we consider two iterative algorithms for the heuristic solution of the problem considered, namely a genetic algorithm and a simulated annnealing algorithm. All these iterative algorithms work with the job sequences for the first stage. If a new job sequence has been generated, the schedule construction procedure described in Section 3 is applied and the objective function value of this schedule is used for evaluating the job sequence for the first stage.

5.1 Genetic Algorithm

A genetic algorithm is an iterative heuristic based on Darwin's evolutionary theory about "survival of the fittest and natural selection". It belongs to the evolutionary class of artificial intelligent (AI) techniques. Holland (1975) proposes some basic principles of natural evolution as a methodology to solve decision-making problems. For the classical flow shop problem, the first genetic algorithm has been given by Werner (1984).

A genetic algorithm is characterized by a parallel search of the state space in contrast to a point-by-point search by conventional optimization techniques. The parallel search is achieved by keeping a set of possible solutions under consideration, called a *population*. An individual in the population is a string of symbols and it is an abstract representation of a solution. The algorithm starts with an initial generation of artificial individuals which are created randomly. Each symbol is called a *gene* and each string of genes is termed as a *chromosome*. The individuals in the population are evaluated by some *fitness* measure to describe quantitatively how well the individual masters its task. The population of chromosomes evolves from some generation to the next through the use of two types of genetic operators: (1) unary operators such as *mutation* and *inversion* which change the genetic structure of a single chromosome, and (2) a higher-order operator, referred to as *crossover* which consists of obtaining new individual(s) by combining the genetic material from two selected parent chromosomes. When applying crossover, two individuals (*parents*) are *selected* from the population and new solution(s), called *offspring*, is(are) created. Mutation creates a new solution by a random change on a selected individual. The genetic operators (crossover and mutation) are applied to randomly selected parents to generate new offspring. Then the new population is selected out of the individuals of the current population and the new generated chromosomes (Gen and Cheng, 1997). A basic GA is shown below.

```
Genetic Algorithm
  Begin
    $N_p$ = Population size.
    $P_c$ = Crossover probability.
    $P_m$ = Mutation probability.
    $P \leftarrow$ Construct ($N_p$)          // Construct the initial population

    For $j$ = 1 to $N_p$
      Evaluate Fitness($p[j]$);
    EndFor

    Repeat
      For $j$ = 1  to $P_c \times N_p$
        ($x$, $y$) $\leftarrow$ Select_Cross ($P$);  // Select two parents proportional to their fitness using roulette wheel.
        Crossed[j] $\leftarrow$ Cross_Over ($x$, $y$);
        Evaluate Fitness(Crossed[$j$]);
      EndFor

      For $j$ = 1  to $P_m \times N_p$
        ($x$) $\leftarrow$ Select_Mutate ($P$);  // Select one parent proportional to its fitness using roulette wheel.
        Mutated[$j$] $\leftarrow$ Mutation ($x$);
        Evaluate Fitness(Mutated[$j$]);
      EndFor

      $P \leftarrow$ Select_for_Next_Gen        // Select from offspring and parents the new parents for the next
                                                  generation.

    Until (Stop-Criterion)
    Return (individual with highest fitness value)
  End
```

The application of a genetic algorithm to any problem requires the representation of a solution for the problem as a string of symbols, a choice of genetic operators, an evaluation function, a selection mechanism, and the determination of genetic parameters (population size and probabilities of crossover and mutation).

Encoding scheme

The string coding is used for making up a chromosome directly. Thus, the digits of the chromosome will be equal to the numbers of jobs (*n* digits). The gene codes are positive integer numbers representing the numbers of jobs. Thus, the method is named as genetic algorithm based on the *job code*. When encoding the solutions, the concept of *random key encoding* which is the most prominent and successful example of such an encoding proposed by Bean (1994) is used. The sequence of the first stage is represented by the numbers of genes from the left side to the right side as shown in Figure 1.

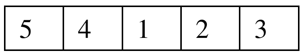

Fig. 1. Illustration of a chromosome.

Crossover

A *subsequence preservation* crossover operator is used since the offspring inherit a subsequence from the first parent will try to convert the solution. Moreover, the advantage of this method guarantees the generation of feasible sequences after the crossover. This method randomly takes two parents from the population and creates two offspring. The first offspring is created by duplicating a subsequence from the first parent into an offspring and then completing the offspring by inserting the remaining jobs derived from the second parent by making a left-to-right scan. The second offspring is created by fixing a sequence of jobs from the second parent whose jobs are the same as the random subsequence of the first parent and inserting the remaining jobs derived from the first parent by making a left-to-right scan. The crossover operator is depicted in Figure 2.

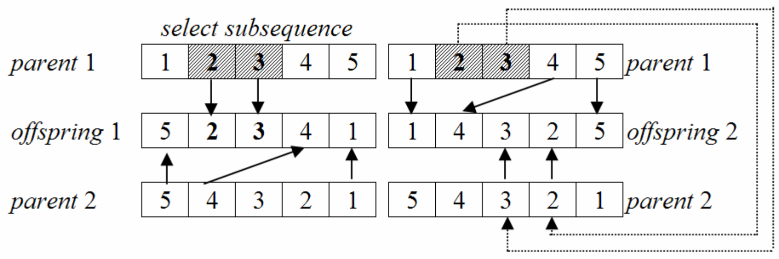

Fig. 2. Illustration of the subsequence crossover operator.

Mutation

This operator randomly chooses a chromosome and two of its genes as a candidate to be mutated, and then swaps the genes on such a chromosome. It is called reciprocal exchange mutation as illustrated in Figure 3.

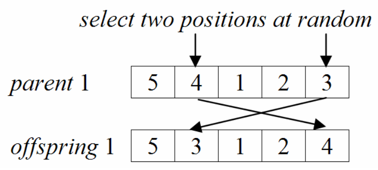

Fig. 3. Illustration of the reciprocal exchange mutation.

Evaluation policy

During each generation, chromosomes are evaluated using some measure of fitness. In most optimization applications, the fitness function is constructed based on the original objective function. The fitness value of each chromosome is a key measure to guide the direction of search in GA. Due to the minimization problem, the fitness value must be in inverse proportion to the objective function value so that a fitter chromosome has a larger fitness value.

$$fitness\,(v_z) \;=\; \frac{1}{f\,(v_z)} \qquad , z = 1,2,\ldots, population\_size\,;$$

where *fitness*(v_z) is the fitness value and *f*(v_z) is the objective function value of the *z*-th chromosome for the complete schedule generated from the corresponding job sequence for the first stage using Algorithms 1-2.

According to this research objective, the objective is to minimize a positively weighted convex sum of makespan and number of tardy jobs. Thus, the fitness value of a chromosome, *fitness*(v_z) is given by

$$fitness\,(v_z) = \frac{1}{\lambda C_{\max}(v_z) + (1-\lambda)\sum_{j=1}^{n} U_j(v_z) + 1} \qquad , z = 1,2,\ldots, population\_size\,;$$

where $C_{max}(v_z)$ is the makespan of the *z*-th chromosome (resp. of the resulting complete schedule) ,$U_j(v_z)$ is a Boolean variable for job *j* of the *z*-th chromosome which is equal to 1 if job *j* is tardy, and 0 otherwise, and λ denotes the weight (or relative importance) given to makespan and number of tardy jobs. The largest value of the fitness function is the lowest value of the positively weighted convex sum of makespan and number of tardy jobs. In the denominator value one is added in order to prevent a division by zero when the weight λ and the number of tardy jobs are zero.

Selection policy

An elitist policy and enlarged sampling space technique are used. Both parents and the offspring have the same chance of completing for survival. Figure 4 illustrates the selection based on an enlarged sampling space. Then Holland's *proportionate selection* or *roulette wheel selection* is employed to reproduce the next generation based on the current enlarged population. The idea is to determine a selection probability (also called survival probability) for each chromosome proportional to its fitness value. For chromosome v_z with fitness *fitness*(v_z), its selection probability *prob*(v_z) is calculated as follows:

$$prob(v_z) \;=\; \frac{fitness(v_z)}{\sum_{z=1}^{population\_size\,+\,offspring\_size} fitness(v_z)}$$

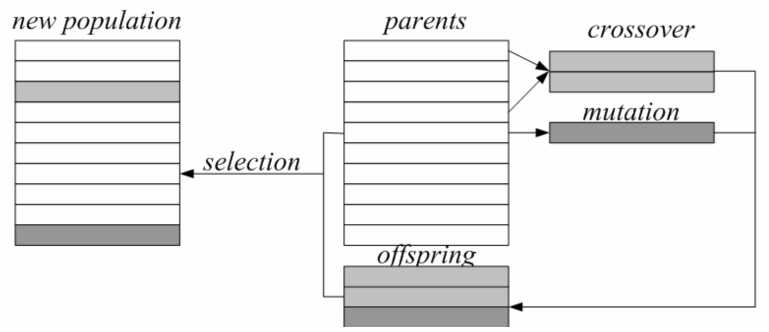

Fig. 4. Illustration of the selection performed on an enlarged sampling space.

Termination condition

In the implementation of GA, the search procedure is terminated when the best objective function value found so far is not updated for a predetermined number of generations. It can also be terminated when the number of generation exceeds the predetermined number of generations.

We apply GA with a crossover rate and a mutation rate both equal to 0.2. The maximum number of generations is set equal to 1000 in advance. Moreover, GA is terminated when within 100 generations no improvement of the best function value has been obtained. For the problem data given in Tables 2-4, the genetic algorithm also produced optimal solutions for all values of λ considerd.

5.2 Simulated Annealing

Simulated annealing (SA) is an enhanced version of local optimization or an iterative search method, in which an initial solution is repeatedly improved by making small local alterations until no such alteration yields a better solution. Annealing refers to the process which occurs when physical substances, such as metals, are raised to a high energy level (melted) and then gradually cooled until some solid state is reached. The goal of this process is to reach the lowest energy state. In this process, physical substances usually move from higher energy states to lower ones if the cooling process is sufficiently slow, so a minimization naturally occurs. Due to natural variability, however, there is some probability at each stage of the cooling process that a transition to a higher energy state will occur. As the energy state naturally declines, the probability of moving to a higher energy state decreases. Then the simulated annealing algorithm is as follows, where RANDOM(0,1) denotes a random number taken from a random variable uniformly distributed in the inerval (0,1).

As mentioned earlier, SA attempts to avoid an entrapment in a local optimum by sometimes accepting a move that deteriorates the value of the objective function f. Starting from an initial solution randomly generated, SA generates a new solution S' in the neighborhood of the initial solution S. This new point's objective function value is then compared to the initial point's value (remind that the objective function value of the full schedule generated from the job sequence for the first stage is taken). The change in the objective function value, $\Delta = f(S') - f(S)$, is calculated in order to determine whether the new value is smaller. For the case of minimization, if the objective function value decreases ($\Delta < 0$), it is automatically accepted and it becomes the point from which the search will continue. The algorithm will then proceed with another iteration. If the objective function value increases ($\Delta \geq 0$), then higher values of the objective function may also be accepted with a probability, usually determined by a function, exp ($-\Delta/T$), where T is a control parameter called the *temperature*. This temperature, which is simply a positive number, is periodically reduced by a temperature scheme, so that it moves gradually from a relatively high value to near zero as the method progresses. Thus, at the start of SA most deteriorating moves are accepted, but at the end only ameliorating ones are likely to be accepted. The method converges to a local optimum as the temperature approaches zero, but by occasionally accepting points with higher values of the objective function, the SA algorithm is able to escape local optima. During the algorithm, T is reduced every NT iterations according to $T_{i+1} = T_i - RT \times T_0$, where NT is a preset parameter which establishes the number of iterations between temperature reductions, called the *epoch length*, index i refers to the ith epoch and RT is the temperature reduction factor. We decided to take a linear reduction scheme (Winston and Venkataramanan, 2003) which, for a constant number of epochs, reduces the temperature slower than e.g. a geometric cooling scheme.

Neighborhood search

In search algorithms, the *neighborhood* of a solution describes which solutions can be generated from the current one by a specific alteration. As an example, an insertion or pairwise interchange neighborhood is commonly used when solutions are described by permutations. For a job sequence for the first stage, a job at some position is selected and reinserted at some other position in the insertion neighborhood, while two jobs are selected and interchanged in the latter neighborhood. We use the pairwise interchange neighborhood, where a job sequence has $\frac{1}{2}n(n-1)$ neighbors.

Simulated annealing algorithm

```
Begin
    $T = T_0$;                    // $T_0$ is the initial temperature
    $S = S_0$;                    // $S_0$ is the initial solution
    $BestS = S_0$;                // $BestS$ is the best solution seen so far
    $BestF = f(S_0)$              // $BestF$ is the best objective function value seen so far
    $NT$                          // Number of iterations with constant temperature
    Repeat
        For $l = 1$ to $NT$ do
            $S'$ = PERTURB ($S$); {generate a neighbor $S'$ of $S$}
            If $f(S') < BestF$ then   $BestF = f(S')$; $BestS = S'$;
            $\Delta = f(S') - f(S)$;
            If $\Delta < 0$ then $S = S'$;
            Else if (exp ($-\Delta/T$) > RANDOM(0,1)) then
                $S = S'$;
            Endif
            If no update of the best solution within the last 30 iterations then $l = NT$;
        EndFor;
        UPDATE ($T$);
    Until (Stop-Criterion)
    Return (job sequence with smallest function value)
End
```

Termination condition

In SA, the temperature is reduced to a smaller temperature when the best objective function value found so far is not updated for a predetermined number of iterations and it is also reduced when *NT* iterations have been performed (i.e. at the end of an epoch length). The procedure is terminated when the temperature becomes equal to or less than zero.

We apply SA with an initial temperature equal to T_0=2000. The number of iterations between temperature reductions is NT =100 and the temperature reduction factor RT=0.1. If no update of the best solution is made, the temperature is reduced after 0.3 *x* $NT=30$ iterations. For the problem data given in Tables 2-4, the simulated annealing algorithm also produced optimal solutions for all values of λ considerd.

6. Experiments

6.1 Small Size Problems

All methods presented above produce an optimal solution for the example problem. In practice, however, this will not always be the case. This section provides experiments designed to study the effectiveness of the constructive algorithms such as Palmer (PAL), CDS, Gupta (GUP), Dannnenbring (DAN), and the NEH adaptation, and the iterative algorithms such as the genetic algorithm (GA) and simulated annealing (SA). The small sample consists of 180 flexible flowshop problems. The number of jobs ranges from three to five, the number of stages ranges form two to five, and the number of unrelated parallel machines at each stage ranges from two to five. The standard processing times of the operations are uniformly distributed integer values from one to fifty time units, while the relative speed for every machine at each stage for every job varies between 0.7 and 1.3, and the setup times from the previous job to the next job at each stage range from zero to ten.

In order to evaluate the performance of each of the heuristics, we use the relative error for λ>0 and the absolute error for λ=0. If the positively weighted convex sum of the objectives obtained by a particular heuristic is denoted by f_{heu} and the optimal positively weighted convex sum of the objectives is denoted by f_{opt}, then the error E is calculated as:

$$E = \begin{cases} \dfrac{f_{heu} - f_{opt}}{f_{opt}} & ;when \quad \lambda > 0 \\ f_{heu} - f_{opt} & ;otherwise \end{cases}$$

Table 6 shows the mean error *E* for every heuristic. The results show that the genetic algorithm is the best one for every λ. Table 7 shows the percentages of the particular values of tardy jobs obtained for every heuristic. Table 8 shows the percentage of instances for which the error is in the corresponding interval. The genetic algorithm finds the optimal value in about 45% of all problems, simulated annealing in about 40%, and the NEH algorithm in about 30%, while the other heuristics in about 25% of the problems.

Table 6
Mean error *E* for every heuristic

	Mean error *E* (job units)	Mean error *E* (%)	
	λ = 0	λ = 0.5	λ = 1
PAL	0.16	2.86	2.86
CDS	0.12	2.72	2.66
GUP	0.14	2.86	2.61
DAN	0.12	2.72	2.71
NEH	0.08	2.27	2.04
GA	0.03	1.06	1.02
SA	0.03	1.49	1.47

Table 7
Percentage of number of tardy jobs for every heuristic

η_T	PAL	CDS	GUP	DAN	NEH	GA	SA
0	85.00	88.89	86.11	83.33	92.22	97.22	96.67
1	13.89	10.56	13.33	11.11	7.78	2.78	3.33
2	1.11	0.56	0.56	0.56	0	0	0

6.2 Large Size Problems

For a comparison of the proposed heuristics, some test problems are randomly generated the details of which are as follows.

(1) A manufacturing environment is defined as a number of jobs × number of stages scenario in which the number of jobs can be 5, 10, 30, or 50, and the number of stages can be 5, 10, or 20. Thus, twelve manufacturing environments are considered for the experiments.
(2) For the objective function, the weight λ for each problem can be 0, 0.25, 0.5, 0.75, or 1. Thus, the total number of scenarios to be tested is equal to sixty different problem types.
(3) The number of unrelated parallel machines at each stage is randomly set between one and three, but at least one stage must have multiple machines.
(4) The standard processing times of a job are integers uniformly distributed in the interval (1, 100). The relative speeds vary between 0.7 and 1.3, and setup times vary between zero and ten time units.

The computational experiments are conducted for 50 test problems in each scenario by using an Intel Pentium 4 2.00GHz CPU. While an optimal solution for the small problem set is obtainable using e.g. 0-1 mixed integer programming, for the large size problems optimal solutions are very difficult to find. Therefore, this paper compares the performance of all proposed heuristics by using the mean (relative or absolute) deviation (*MD*) which is calculated as follows:

$$MD = \begin{cases} \frac{1}{50}\left(\frac{f_{heu} - f_{best}}{f_{best}}\right) & ;when \quad \lambda > 0, \\ \frac{1}{50}(f_{heu} - f_{best}) & ;otherwise \end{cases}$$

where f_{heu} is the function value obtained by a particular of the five constructive or two iterative algorithms, and f_{best} is the best function value obtained by all heuristics.

The results for all algorithms given in Table 9 through Table 13 indicate that, among the constructive algorithms the NEH algorithm can obtain 279, 242, 249, 254, and 245 best solutions for $\lambda \in \{0.00, 0.25, 0.50, 0.75 \text{ and } 1.00\}$, respectively, and the total mean relative deviations (ΣMD and Σ *%MD*, respectively) from the best function value are 20.82 units as well as 9.85%, 8.65%, 7.72%, and 8.34%, respectively. The other constructive algorithms obtain a number of best solutions substantially smaller than the NEH algorithm, and have total mean deviations from the best function value considerably higher than the NEH algorithm. We can conclude that the NEH algorithm clearly outperforms the other constructive algorithms. Disregarding the NEH algorithm, the results show that the performance of the CDS algorithm is better than the performance of the algorithms by Palmer, Gupta and Dannenbring. Comparing the performance of constructive and iterative algorithms reveals that the performance of the iterative algorithms is substantially better than the performance of constructive algorithms but the CPU times of the constructive algorithms are faster than the CPU times of the iterative algorithms. Among the iterative algorithms, the performance of the genetic algorithm is slightly better than the performance of simulated annealing. The computational times of the genetic algorithm and the simulated annealing algorithm are comparable. The results for the different values of λ are similar, but the superiority of the genetic algorithm over simulated annealing and the NEH algorithm is significant in particular for λ=0 (see Table 9).

7. Conclusions

This paper develops and compares some constructive and iterative heuristics for flexible flowshop scheduling problems with unrelated parallel machines, where a sequence-dependent setup time is necessary before starting the processing of a job. As objective function, this paper considers the minimization of the positively weighted convex sum of makespan and the number of tardy jobs. All the heuristics suggested in this paper generate a job sequence for the first stage by using generalized constructive heuristics for pure flowshop scheduling problems such as Palmer (1965), Campbell, Dudek, and Smith (1970), Gupta (1971), and Dannenbring (1977) as well as the insertion heuristic by Nawaz, Enscore and Ham (1983) to the flexible flowshop environment or iterative algorithms such as a genetic or a simulated annealing algorithm.

The results show that among the constructive algorithms, a job insertion based algorithm outperforms the other constructive algorithms, while among the iterative algorithms the genetic algorithm outperforms the simulated annealing algorithm.

In future research, heuristics that use different stage sequencing orders will be studied to increase the solution quality further. In addition, the procedure for fixing the processing and setup times as well as the relative speeds in the constructive algorithms can be refined further. The use of advanced techniques, e.g. the development of a tabu search, ant colony or an iterated local search algorithm, can also be worthwhile.

Acknowledgement

The fourth author has been supported by INTAS (project 03-51-5501).

References

Alisantoso, D., Khoo, L. P., and Jiang, P. Y. 2003. An immune algorithm approach to the scheduling of a flexible PCB flow shop. **The International Journal of Advanced Manufacturing Technology** 22(11-12): 819-827.

Baker, K. R. 1974. **Introduction to Sequencing and Scheduling**. 1st ed. Canada: John Wiley & Sons.

Bean, J. C. 1994. Genetic algorithms and random keys for sequencing and optimization. **INFORMS Journal on Computing**, 6(2):154-160

Campbell, H. G, Dudek, R. A., and Smith, M. L. 1970. A heuristic algorithm for the n-job m-machine sequencing problem. **Management Science** 16(10): 630-637.

Dannenbring, D. G. 1977. An evaluation of flow shop sequencing heuristics. **Management Science** 23(11):1174-1182.

Finke, D. A., and Medeiros, D. J. 2002. Shop scheduling using tabu search and simulation. **Proceedings of the 2002 Winter Simulation Conference** : 1013-1017.

Garey, M. R., and Johnson, D. S. 1979. **Computers and Intractability: A Guide to the Theory of NP-Completeness**, San Francisco, CA: Freeman.

Gen, M., and Cheng, R. 1997. **Genetic Algorithms and Engineering Design**. 1st ed. New York: John Wiley&Son.

Gupta, J. N. D. 1971. A functional heuristic algorithm for the flow-shop scheduling problem. **Operations Research Quarterly** 22(1): 39-47.

Gupta, J. N. D. 1988. Two-stage, hybrid flow shop scheduling problem. **Journal of the Operational Research Society** 39(4): 359-364.

Gupta, J. N. D., Krüger, K., Lauff, V., Werner, F., and Sotskov, Y. N. 2002. Heuristics for hybrid flow shops with controllable processing times and assignable due dates. **Computers & Operations Research** 29(10): 1417-1439.

Holland, J. A. 1975. **Adaptation in natural and artificial systems**. Ann Arbor. University of Michigan.

Hoogeveen, J. A., Lenstra, J. K., and Veltman, B. 1996. Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard. **European Journal of Operational Research**, 89 (1–2): 172–175.

Lin, Hung-Tso., and Liao, Ching-Jong. 2003. A case study in a two-stage hybrid flow shop with setup time and dedicated machines. **International Journal of Production Economics** 86(2): 133-143.

Nawaz, M, Enscore, Jr. E., and Ham, I. 1983. A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem. **OMEGA** 11(1): 91-95.

Palmer, D. S. 1965. Sequencing jobs through a multi-stage process in the minimum total time--a quick method of obtaining a near optimum. **Operational Research Quarterly** 16(1):101-107.

Paul, R. J. 1979. A production scheduling problem in the glass-container industry. **Operations Research** 27(2): 290–302.

Pinedo, M. 1995. **Scheduling: theory, algorithms, and systems**. 1st ed. New Jersey: Prentice-Hall.

Santos, D. L., Hunsucker, J. L., and Deal, D. E. 1996. An evaluation of sequencing heuristics in flow shops with multiple processors. **Computers & Industrial Engineering** 30(4): 681-691.

Tsubone, H., Ohba, M., Takamuki, H., and Miyake, Y. 1993. Production scheduling system in the hybrid flow shop. **OMEGA**. 21(2): 205-214.

Wang, W. 2005. Flexible flow shop scheduling: Optimum, heuristics, and artificial intelligence solutions. **Expert Systems**. 22(2): 78 – 85.

Wang, W., and Hunsucker, J. L. 2003. An evaluation of the CDS heuristic in flow shops with multiple processors. **Journal of the Chinese Institute of Industrial Engineers** 20(3): 295-304.

Winson, W. L., and Venkataramanan. 2003. **Introduction to Mathematical Programming**. 4th.ed. Canada: Duxbury Press.

Werner, F. 1984. On the solution of special sequencing problems, PhD Thesis, TU Magdeburg (in German).

Yanney, J. D., and Kuo, W. 1989. A practical approach to scheduling a multistage, multiprocessor flow-shop problem. **International Journal of Production Research** 27(10) : 1733–1742.

Table 8
Percentage of numbers how often a particular error *E* has been obtained by every heuristic

E (%)	PAL		CDS		GUP		DAN		NEH		GA		SA	
	λ=0.5	λ=1.0	λ=0.5	λ=1.0	λ=0.5	λ=1.0	λ=0.5	λ=1.0	λ=0.5	λ=1.0	λ=0.5	λ=1.0	λ=0.5	λ=1.0
$E = 0$	23.89	22.78	26.11	25.00	28.33	26.11	25.56	24.44	33.89	32.78	47.22	45.00	40.56	38.89
$0 < E \leq 1$	12.22	12.22	11.11	10.00	13.89	13.33	13.89	13.89	14.44	16.11	18.89	18.89	18.33	16.67
$1 < E \leq 2$	13.33	12.78	11.67	14.44	10.00	11.67	11.11	11.37	11.67	13.89	13.89	17.78	13.33	16.67
$2 < E \leq 3$	9.44	10.56	11.44	12.22	10.56	11.67	12.22	12.22	10.56	10.56	6.67	6.11	8.33	8.33
$3 < E \leq 4$	10.00	11.67	10.00	12.22	8.89	10.00	8.33	8.33	10.56	8.33	5.00	5.00	6.11	5.00
$4 < E \leq 5$	10.56	8.89	9.44	9.44	9.44	8.89	10.00	10.00	3.33	5.00	5.00	5.00	6.11	7.22
$5 < E \leq 6$	3.33	3.89	3.33	3.33	3.33	8.44	5.00	3.89	3.33	3.89	1.67	2.22	2.22	3.89
$6 < E \leq 7$	7.22	7.78	5.00	6.11	5.56	6.67	6.67	8.89	3.33	4.44	1.11	0.00	2.75	2.22
$7 < E \leq 8$	3.33	4.44	2.78	1.67	3.89	2.22	1.67	2.22	3.33	2.22	0.56	0.00	1.11	0.56
$8 < E \leq 9$	5.00	2.22	1.11	1.11	2.78	1.67	1.67	1.11	2.22	1.67	0.00	0.00	0.00	0.56
$9 < E \leq 10$	0.56	1.11	2.78	1.67	1.67	1.67	1.67	1.11	1.11	0.00	0.00	0.00	1.11	0.00
$10 < E \leq 11$	0.56	1.11	1.67	1.67	1.11	1.67	1.11	1.11	0.00	0.00	0.00	0.00	0.00	0.00
$11 < E \leq 12$	0.00	0.56	0.00	0.56	0.00	0.00	0.56	0.00	0.56	0.00	0.00	0.00	0.00	0.00
$12 < E \leq 13$	0.56	0.00	0.00	0.00	0.56	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$13 < E \leq 14$	0.00	0.00	0.56	0.00	0.00	0.00	0.00	0.56	0.00	0.56	0.00	0.00	0.00	0.00
$14 < E \leq 15$	0.00	0.00	0.00	0.56	0.00	0.00	0.56	0.00	0.56	0.00	0.00	0.00	0.00	0.00
$15 < E \leq 16$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.56	0.00	0.00	0.00	0.00	0.00	0.00
$16 < E \leq 17$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.11	0.00	0.00	0.00	0.00	0.00
$17 < E \leq 18$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.56	0.00	0.00	0.00	0.00

Table 9
Comparison of the heuristics for λ = 0.00

Size	Mean absolute deviation (No. of best solutions)							CPU times in seconds						
	PAL	CDS	GUP	DAN	NEH	GA	SA	PAL	CDS	GUP	DAN	NEH	GA	SA
5 × 5	0.32 (35)	0.26(38)	0.34(33)	0.32(35)	0.28(38)	0.00(50)	0.06(47)	0.0000	0.0003	0.0000	0.0003	0.0024	0.0082	0.0128
5 × 10	0.32(35)	0.26(37)	0.36(34)	0.32(36)	0.22(40)	0.06(47)	0.02(49)	0.0000	0.0045	0.0010	0.0000	0.0100	0.0156	0.0113
5 × 20	0.26(37)	0.18(42)	0.24(39)	0.36(32)	0.12(44)	0.00(50)	0.00(50)	0.0012	0.0208	0.0010	0.0019	0.0228	0.0522	0.0341
10 × 5	1.20(16)	0.72(25)	1.10(15)	1.04(20)	0.82(23)	0.06(49)	0.20(41)	0.0003	0.0033	0.0000	0.0000	0.0221	0.0831	0.0635
10 × 10	1.12(17)	0.56(26)	1.02(18)	0.98(18)	0.84(24)	0.00(50)	0.16(42)	0.0009	0.0101	0.0022	0.0028	0.0630	0.1493	0.1423
10 × 20	1.08(17)	0.30(36)	1.06(19)	0.94(16)	0.44((33)	0.00(50)	0.06(47)	0.0033	0.0492	0.0041	0.0053	0.1302	0.2222	0.3138
30 × 5	2.98(9)	2.12(14)	2.76(11)	2.52(11)	1.38(23)	0.08(48)	0.50(41)	0.0024	0.0084	0.0022	0.0029	0.7709	1.1249	0.9654
30 × 10	2.94(8)	1.80(10)	2.86(10)	2.70(9)	1.10(19)	0.06(48)	0.42(41)	0.0039	0.0482	0.0078	0.0098	1.7106	2.8794	2.0719
30 × 20	2.86(2)	1.84(7)	3.22(1)	2.82(4)	1.46(16)	0.10(48)	0.34(42)	0.0139	0.3191	0.0103	0.0167	4.9231	6.6178	6.8850
50 × 5	9.76(2)	8.26(5)	9.78(1)	9.44(1)	4.50(6)	0.00(50)	0.68(43)	0.0033	0.0180	0.0059	0.0122	3.9762	10.8948	7.6950
50 × 10	10.80(0)	8.30(1)	10.06(0)	10.08(0)	5.18(7)	0.00(50)	0.36(47)	0.0108	0.1501	0.0119	0.0147	10.3445	33.2564	25.0312
50 × 20	8.34(0)	5.68(2)	8.82(0)	7.58(0)	4.48(6)	0.02(49)	0.40(47)	0.0353	0.5399	0.0290	0.0372	32.2563	53.2611	57.7071
Σ*MD*	41.98(178)	30.28(243)	41.62(181)	39.10(182)	20.82(279)	0.38(589)	3.2(537)							

Table 10
Comparison of the heuristics for λ = 0.25

Size	% Mean relative deviation (No. of best solutions)							CPU times in seconds						
	PAL	CDS	GUP	DAN	NEH	GA	SA	PAL	CDS	GUP	DAN	NEH	GA	SA
5 × 5	2.80(13)	2.96(14)	2.75(14)	2.94(14)	1.58(29)	0.01(48)	0.17(46)	0.0000	0.0015	0.0000	0.0000	0.0043	0.0173	0.0190
5 × 10	2.02(13)	1.32(20)	2.83(12)	2.51(13)	1.23(23)	0.02(48)	0.11(47)	0.0003	0.0045	0.0003	0.0022	0.0084	0.0519	0.0434
5 × 20	1.79(9)	0.71(28)	2.38(10)	2.52(6)	1.10(25)	0.01(48)	0.05(46)	0.0012	0.0233	0.0003	0.0022	0.0188	0.0878	0.0605
10 × 5	4.12(5)	3.32(6)	4.50(3)	4.66(4)	1.15(29)	0.06(47)	0.20(42)	0.0003	0.0030	0.0006	0.0003	0.0265	0.0828	0.0641
10 × 10	2.85(5)	2.04(10)	3.40(5)	3.21(5)	0.93(28)	0.06(44)	0.12(45)	0.0006	0.0102	0.0006	0.0025	0.0538	0.1308	0.1479
10 × 20	3.07(5)	1.40(10)	3.82(4)	3.18(5)	0.94(16)	0.09(44)	0.19(35)	0.0033	0.0732	0.0035	0.0037	0.1072	0.3166	0.3828
30 × 5	7.89(3)	6.33(5)	7.77(3)	6.55(4)	0.44(23)	0.08(46)	0.19(42)	0.0015	0.0077	0.0022	0.0047	0.6578	0.9269	0.9829
30 × 10	7.76(3)	5.59(5)	8.43(3)	7.50(3)	0.26(24)	0.02(46)	0.12(41)	0.0048	0.0714	0.0075	0.0097	1.6582	2.6850	2.0709
30 × 20	7.14(1)	4.42(2)	8.47(1)	6.73(4)	0.40(18)	0.01(46)	0.11(41)	0.0134	0.2874	0.0135	0.0184	4.9094	9.2293	7.0693
50 × 5	9.32(1)	8.06(3)	9.85(0)	8.45(2)	0.32(15)	0.00(48)	0.10(41)	0.0030	0.0186	0.0063	0.0082	3.9278	10.5672	7.7562
50 × 10	9.49(0)	7.81(1)	11.55(0)	8.77(0)	0.84(5)	0.00(49)	0.09(46)	0.0114	0.1710	0.0132	0.0147	10.1396	38.4227	25.5632
50 × 20	8.27(0)	5.92(1)	11.22(1)	7.45(0)	0.66(7)	0.03(49)	0.18(40)	0.0375	0.5005	0.0287	0.0387	32.2606	57.3872	58.5294
Σ *%MD*	66.52(58)	49.88(105)	76.97(56)	64.47(60)	9.85(242)	0.39(562)	1.63(512)							

Table 11
Comparison of the heuristics for λ = 0.50

Size	% Mean relative deviation (No. of best solutions)							CPU times in seconds						
	PAL	CDS	GUP	DAN	NEH	GA	SA	PAL	CDS	GUP	DAN	NEH	GA	SA
5 × 5	2.34(14)	2.39(15)	2.21(15)	2.58(12)	0.93(32)	0.00(50)	0.05(47)	0.0000	0.0006	0.0000	0.0000	0.0016	0.0122	0.0176
5 × 10	1.92(11)	1.19(19)	2.77(11)	2.38(11)	1.03(22)	0.02(48)	0.11(47)	0.0003	0.0037	0.0003	0.0003	0.0094	0.0440	0.0357
5 × 20	1.80(9)	0.72(27)	2.41(10)	2.44(6)	1.06(23)	0.02(49)	0.01(48)	0.0015	0.0191	0.0018	0.0037	0.0153	0.0765	0.0604
10 × 5	3.87(6)	3.16(6)	4.24(3)	4.45(3)	0.92(29)	0.01(48)	0.10(42)	0.0006	0.0027	0.0016	0.0006	0.0291	0.0696	0.0625
10 × 10	2.85(4)	2.06(10)	3.39(4)	3.27(4)	0.89(28)	0.05(42)	0.18(43)	0.0015	0.0119	0.0048	0.0025	0.0602	0.1231	0.1490
10 × 20	3.04(4)	1.29(11)	3.68(4)	3.06(5)	0.82(19)	0.05(46)	0.12(40)	0.0027	0.0668	0.0054	0.0059	0.1053	0.3164	0.3834
30 × 5	7.44(5)	5.97(5)	7.41(4)	6.22(5)	0.24(27)	0.01(47)	0.08(40)	0.0021	0.0089	0.0025	0.0050	0.6591	0.9231	0.9746
30 × 10	7.59(3)	5.49(5)	8.80(3)	7.50(3)	0.31(27)	0.00(48)	0.22(39)	0.0058	0.0573	0.0078	0.0095	1.6751	2.8851	2.0224
30 × 20	7.13(1)	4.46(2)	8.37(1)	6.78(1)	0.49(14)	0.01(48)	0.18(38)	0.0144	0.2298	0.0125	0.0160	4.8544	10.1723	7.0853
50 × 5	9.00(1)	7.78(3)	9.47(0)	8.20(2)	0.32(14)	0.01(46)	0.07(43)	0.0042	0.0256	0.0087	0.0113	3.8888	7.6918	7.7784
50 × 10	9.53(0)	7.86(1)	11.67(0)	8.84(0)	0.94(7)	0.01(49)	0.09(45)	0.0100	0.1357	0.0102	0.0150	10.1481	35.0466	25.3990
50 × 20	8.26(0)	5.88(1)	11.48(0)	7.47(0)	0.70(7)	0.01(49)	0.15(42)	0.0389	0.6030	0.0272	0.0407	31.9904	59.7749	58.8381
Σ *%MD*	64.77(58)	48.25(105)	75.9(55)	63.19(52)	8.65(249)	0.20(570)	1.36(514)							

Table 12
Comparison of the heuristics for λ = 0.75

Size	% Mean relative deviation (No. of best solutions)							CPU times in seconds						
	PAL	CDS	GUP	DAN	NEH	GA	SA	PAL	CDS	GUP	DAN	NEH	GA	SA
5 × 5	2.43(10)	2.45(13)	2.30(12)	2.69(11)	1.02(29)	0.00(50)	0.09(46)	0.0000	0.0018	0.0006	0.0000	0.0032	0.0164	0.0143
5 × 10	2.04(9)	1.29(17)	2.90(9)	2.49(9)	0.87(24)	0.00(50)	0.14(44)	0.0009	0.0042	0.0006	0.0006	0.0066	0.0472	0.0337
5 × 20	1.95(8)	0.87(24)	2.57(10)	2.57(6)	0.93(27)	0.00(50)	0.01(49)	0.0003	0.0254	0.0015	0.0006	0.0162	0.0709	0.0577
10 × 5	4.21(6)	4.84(6)	4.52(4)	4.72(3)	1.00(32)	0.09(47)	0.33(43)	0.0006	0.0009	0.0000	0.0010	0.0243	0.0659	0.0640
10 × 10	3.17(5)	2.43(8)	3.62(6)	3.57(6)	0.96(24)	0.06(45)	0.25(37)	0.0012	0.0101	0.0015	0.0028	0.0608	0.1313	0.1503
10 × 20	3.03(4)	1.27(15)	3.64(4)	3.04(4)	0.80(18)	0.18(46)	0.08(41)	0.0030	0.0634	0.0046	0.0037	0.1046	0.3121	0.3859
30 × 5	7.65(8)	6.21(8)	7.70(6)	6.47(8)	0.19(28)	0.01(48)	0.06(44)	0.0021	0.0072	0.0028	0.0034	0.6654	0.8257	0.9449
30 × 10	7.71(3)	5.75(5)	8.71(3)	7.60(3)	0.27(21)	0.02(44)	0.12(39)	0.0051	0.0702	0.0078	0.0071	1.7431	2.6352	2.0460
30 × 20	7.10(1)	4.41(2)	8.54(1)	6.80(1)	0.36(21)	0.01(49)	0.17(42)	0.0147	0.2371	0.0118	0.0172	4.8419	9.4974	6.9461
50 × 5	9.32(1)	8.11(3)	9.94(0)	8.52(1)	0.21(21)	0.00(49)	0.10(41)	0.0066	0.0325	0.0075	0.0084	3.7931	7.3219	7.7980
50 × 10	9.80(0)	8.13(1)	11.97(0)	9.12(0)	0.55(4)	0.00(50)	0.05(46)	0.0133	0.1019	0.0120	0.0146	10.1288	32.8102	25.5427
50 × 20	8.10(0)	5.72(1)	11.53(0)	7.33(0)	0.56(5)	0.00(49)	0.10(43)	0.0343	0.5157	0.1277	0.0376	32.0284	61.8431	59.1258
Σ *%MD*	66.51(55)	51.48(103)	77.94(55)	64.92(52)	7.72(254)	0.37(577)	1.50(515)							

Table 13
Comparison of the heuristics for $\lambda = 1.00$

Size	% Mean relative deviation (No. of best solutions)							CPU times in seconds						
	PAL	CDS	GUP	DAN	NEH	GA	SA	PAL	CDS	GUP	DAN	NEH	GA	SA
5 × 5	2.36(11)	2.37(13)	2.25(14)	2.64(11)	0.80(32)	0.00(50)	0.03(49)	0.0000	0.0012	0.0003	0.0000	0.0050	0.0251	0.0151
5 × 10	1.87(10)	1.22(20)	2.84(7)	2.31(10)	1.04(21)	0.00(49)	0.15(44)	0.0009	0.0038	0.0000	0.0000	0.0068	0.0409	0.0332
5 × 20	1.86(8)	0.77(25)	2.47(10)	2.47(10)	1.06(24)	0.00(50)	0.05(47)	0.0003	0.0253	0.0028	0.0034	0.0166	0.0672	0.0591
10 × 5	3.83(6)	3.22(5)	4.13(4)	4.41(4)	0.99(31)	0.00(50)	0.25(40)	0.0003	0.0015	0.0009	0.0028	0.0267	0.0600	0.0651
10 × 10	2.86(3)	2.09(7)	3.30(3)	3.29(4)	0.87(25)	0.05(44)	0.14(39)	0.0012	0.0106	0.0019	0.0022	0.0513	0.1265	0.1504
10 × 20	2.98(5)	1.22(12)	3.57(3)	2.99(4)	0.77(18)	0.05(46)	0.05(43)	0.0028	0.0725	0.0043	0.0066	0.1064	0.3169	0.3866
30 × 5	7.28(3)	5.86(5)	7.29(3)	6.13(4)	0.22(28)	0.01(45)	0.08(42)	0.0012	0.0092	0.0040	0.0046	0.6653	0.9890	0.9719
30 × 10	7.47(3)	5.42(5)	8.51(3)	7.42(3)	0.22(28)	0.03(44)	0.09(42)	0.0038	0.0646	0.0092	0.0071	1.7096	2.6399	2.0483
30 × 20	7.13(1)	4.42(2)	8.32(2)	6.79(1)	0.43(16)	0.03(43)	0.14(41)	0.0199	0.2391	0.0119	0.0165	4.9141	8.6551	7.0404
50 × 5	9.10(1)	8.12(2)	9.69(1)	8.33(1)	0.37(16)	0.02(47)	0.08(44)	0.0022	0.0294	0.0072	0.0063	3.7857	7.7952	7.6505
50 × 10	9.41(0)	7.77(1)	11.59(0)	8.74(0)	0.86(4)	0.00(50)	0.09(45)	0.0121	0.0970	0.0106	0.0129	10.1159	35.4105	25.2787
50 × 20	8.26(0)	5.93(1)	11.73(0)	7.53(0)	0.71(2)	0.01(48)	0.11(43)	0.0420	0.5115	0.0336	0.0384	31.9928	63.4703	58.8691
Σ %*MD*	64.11(51)	48.41(98)	75.69(50)	63.05(52)	8.34(245)	0.20(566)	1.26(519)							