Minimizing Total Flow Time in a Two-Machine Flowshop Problem with Minimum Makespan

Jatinder N. D. Gupta
Department of Management
Ball State University
Muncie, IN 47306, USA

Venkata R. Neppalli
Department of Industrial Engineering
Mississippi State University
Mississippi State, MS 39762, USA
and
Frank Werner
Otto-von-Guericke-Universität
Fakultät für Mathematik, PSF 4120, 39016 Magdeburg, GERMANY

February 28, 2000

Abstract

This paper considers the two-machine flowshop scheduling problem where it is desired to find a minimum total flow time schedule subject to the condition that the makespan of the schedule is minimum. Based on the analysis of the problem characteristics, several existing results are extended to develop two optimization algorithms for the problem. In view of the NP-hardness of the problem, two polynomially solvable cases are identified and solved. Further, several polynomial heuristic solution algorithms are developed and empirically evaluated as to their effectiveness in finding an optimal schedule for the problem.

Keywords: flowshop scheduling, hierarchical criteria, dominance conditions, heuristic algorithms, empirical evaluation

email address: frank.werner@mathematik.uni-magdeburg.de
1 Introduction

In traditional multi-stage problems, a schedule is found to optimize only a single measure of performance (see for instance Tzafestas & Triantafyllakis (1993)). However, in many practical situations, scheduling problems generally involve multiple objectives (Gupta & Dudek (1971), Panwalkar et al. (1973)) as managers develop schedules based on multicriteria. Because of such situations, multicriteria scheduling problems are receiving much attention recently (Nagar et al. (1995a)). Thus, for example, the minimization of total throughput time (also called makespan or maximum completion time) and the minimization of the average time spent by a job in a shop are both important criteria for a scheduling problem. When all jobs are available at time zero and processing conditions are deterministic, the minimization of the average time spent by a job in a shop is equivalent to the minimization of the total flow time of all jobs defined as the sum of the completion times of all jobs.

Multicriteria scheduling problems can be modeled in three different ways. First, when the criteria are equally important, all the efficient solutions for the problem can be generated. Then by using multiattribute decision methods, tradeoffs can be made between these solutions. Second, when the criteria are weighted differently, an objective function can be defined as the sum of weighted functions and transform the problem into a single criterion scheduling problem. Finally, when there is a hierarchy of priority levels for the criteria, the problem can first be solved the for the first priority criterion, ignoring the other criteria and then solved for the second priority criterion under the constraint that the optimal solution of the first priority criterion does not change. Termed hierarchical criteria scheduling (Lee & Vairaktarakis (1993)), this procedure is continued until we solve the problem with last priority criterion as the objective and the optimal solutions of the other criteria as the constraints.

For a scheduling problem with two criteria of interest, if the first two approaches are required, we call the problem a bicriteria scheduling problem. If the second approach is required, we term the problem a hierarchical criteria problem. Using the standard three field notation (Lawler et al. (1993)), a bicriteria scheduling problem for finding all efficient solutions can be represented as \( \alpha|\beta|F(C1,C2) \), where \( \alpha \) denotes the machine environment, \( \beta \) corresponds to the deviations from standard scheduling assumptions, and \( F(C1,C2) \) indicates that efficient solutions relative to criteria \( C1 \) and \( C2 \) are being sought. If the bicriteria scheduling problem involves the sum of weighted values of two objective functions, the problem is denoted as \( \alpha|\beta|F_w(C1,C2) \) where \( F_w(C1,C2) \) represents the weighted sum of two criteria \( C1 \) and \( C2 \). Similarly, a secondary criterion
problem can be denoted as $\alpha|\beta|F_h(C2/C1)$, where $C1$ and $C2$ denote the primary criterion and secondary criterion, respectively, and the notation $F_h(C2/C1)$ represents the *hierarchical* optimization of criterion $C2$ given that criterion $C1$ is at its optimal value.

This paper considers the two-machine flowshop scheduling problem to hierarchically minimize total flow time given that the makespan is optimal. We develop polynomially bounded heuristic algorithms to solve the $F2||F_h(\sum C_i/C_{\text{max}})$ problem and perform a comparative study for them.

The rest of the paper is organized as follows. Section 2 succinctly describes the problem and reviews existing approaches to solve the $F2||F_h(\sum C_i/C_{\text{max}})$ problem. Dominance conditions for generating an optimal schedule are discussed in Section 3 where two dominance algorithms are also described to optimally solve the $F2||F_h(\sum C_i/C_{\text{max}})$ problem. These dominance conditions are further used in Section 4 to describe special polynomially solvable cases. Several polynomial algorithms for finding an optimal or near-optimal schedule for this secondary criteria problem are developed in Section 5. The efficiency and effectiveness of the proposed heuristic algorithms are empirically evaluated in Section 6. Finally, conclusions and directions for future research are outlined in Section 7.

## 2 Problem Description

To succinctly define the $F2||F_h(\sum C_i/C_{\text{max}})$ problem, let $N = \{1, 2, \ldots, n\}$ represent the set of $n$ jobs available at time zero. Each job $i \in N$ is to be processed on two machines, first on machine 1 and then on machine 2. A job once started on a machine is completed on that machine without interruption (i.e., no preemption is allowed). For each job $i \in N$, let $a_i$ and $b_i$ be the processing times at the first and second machine, respectively. The completion time of job $\sigma(i)$ at sequence position $i$ in schedule $\sigma = (\sigma(1), \ldots, \sigma(n))$ is denoted as $C_{\sigma(i)}$ and is calculated using the following recursive relationship:

$$
C_{\sigma(i)} = \max \left\{ \sum_{j=1}^{k} a_{\sigma(j)} + \sum_{j=k}^{i} b_{\sigma(j)} \mid 1 \leq k \leq i \right\}; \quad i \in N
$$

(1)

The hierarchical criteria two-machine flowshop scheduling ($F2||F_h(\sum C_i/C_{\text{max}})$) problem considered here is one of finding a schedule $\sigma = (\sigma(1), \ldots, \sigma(n))$ that first minimizes $C_{\sigma(n)}$ and then minimizes $\sum C_{\sigma(i)}$ where $C_{\sigma(i)}$ values are given by equation (1) above. This type of problems are also known as *secondary criteria* problems.

The $F2||C_{\text{max}}$ problem can be solved by in $O(n \log n)$ computational steps by using Johnson’s (1954) algorithm described as follows:
Algorithm Johnson

Input: $a_i, b_i$ for $i = 1, \ldots, n$.

Step 1. For jobs $j$ with $a_j \leq b_j$, form a partial sequence $\sigma$ such that $a_{\sigma(1)} \leq \ldots \leq a_{\sigma(n')}$, where $n'$ is the number of jobs $j$ with $a_j \leq b_j$.

Step 2. Append the jobs $j$ with $a_j > b_j$ to $\sigma$ so that $b_{\sigma(n' + 1)} \geq \ldots \geq b_{\sigma(n)}$.

While the $F_2||C_{\max}$ problem can be polynomially solved as above, the $F_2||\sum C_i$ problem is known to be NP-hard in the strong sense (Lawler et al. (1993)). Using this result, Chen & Bulfin (1994) have shown that the $F_2||F_h(\sum C_i/C_{\max})$ problem is NP-hard in the strong sense.

Early analytical attempts for solving the $F_2||F_h(\sum C_i/C_{\max})$ problems had been informal in nature and took the shape of generating all alternate minimum makespan schedules. Thus, the problem of generating all minimum makespan schedules, approached by Bagga (1969), Karla & Bansal (1979), Potts (1976), and Szwarc (1981), involved the development of dominance conditions that, if satisfied, can generate alternate minimum makespan schedules. Pandit & Subramanyam (1975) on the other hand, have developed a lexicographic search technique to find all optimal schedules for the $F_2||C_{\max}$ problem. Further, any branch and bound technique for the $F_2||C_{\max}$ problem can be used to generate all optimal schedules.

The literature on multiple criteria multi-stage scheduling problems is summarized by Nagar et al. (1995a). Daniels and Chambers (1990) developed constructive procedures including a heuristic algorithm to solve the $F||F(C_{\max}, T_{\max})$ problem. Nagar et al. (1995b) developed a branch and bound algorithm for the solution of the $F_2||F_w(C_{\max}, \sum C_i)$ problem. They also combined the branch and bound procedure with a genetic algorithm to find an optimal or near optimal solution to the $F_2||F_w(C_{\max}, T_{\max})$ problem (Nager et al. (1996)).

Chen & Vempati (1992) developed a backward branch-and-bound algorithm to optimally solve the $F_2||F_h(\sum C_i/C_{\max})$ problem. A forward branch and bound algorithm for the $F_2||F_h(\sum C_i/C_{\max})$ problem is developed by Rajendran (1993). However, both Rajendran’s, and Chen & Vempati’s algorithms cannot efficiently solve problems involving about 15 or more jobs. Rajendran (1993) developed two heuristics for the $F_2||F_h(\sum C_i/C_{\max})$ problem and tested their effectiveness in solving problems involving 25 or less jobs. Genetic algorithms and other local search approaches to improve Rajendran’s heuristic are suggested by Nepalli et al. (1996) and Gupta et al. (1999).
3 Dominance Algorithms

This section describes dominance conditions that help us eliminate non-optimal schedules. These conditions are used to develop two optimization algorithms to solve the $F_2||F_h(\sum C_i/C_{\text{max}})$ problem.

3.1 Dominance Conditions

We first describe dominance conditions derived from known results for the general $m$-machine permutation flowshop problem. To describe these conditions, let $C^j(\sigma)$ be the completion time of a partial (or complete) schedule $\sigma$ at machine $j \leq m$. Further, let $F(\sigma)$ be the total flow time of a partial (or complete) schedule $\sigma$ expressed as follows:

$$F(\sigma) = \sum_{i=1}^{n} C_{\text{\sigma}(i)}.$$  \hspace{1cm} (2)

Let $\pi$ be an arbitrary partial schedule of jobs not contained in the known initial partial schedule $\sigma$. Further, consider two individual jobs $i$ and $j$ not contained in $\sigma$ and $\pi$.

**Theorem 1** (Gupta (1972b)): For any two schedules $\sigma ij \pi$ and $\sigma ji \pi$, if

$$C^j(\sigma ij) \leq C^j(\sigma ji); \hspace{1cm} 2 \leq j \leq m$$

and

$$C^m(\sigma i) + C^m(\sigma ij) \leq C^m(\sigma j) + C^m(\sigma ji),$$

then

$$C^j(\sigma ij \pi) \leq C^j(\sigma ji \pi); \hspace{1cm} 2 \leq j \leq m$$

and

$$F(\sigma ij \pi) \leq F(\sigma ji \pi).$$

Theorem 1 above is true for any two partial schedules that differ only by the interchange of an adjacent pair of jobs. However, the above results can be extended to cover any two partial schedules that are different permutations of the same subset of jobs.

**Theorem 2** (Gupta (1972b)): For any two partial schedules $\sigma$ and $\delta$ that are different permutations of the same subset of jobs, if

$$C^j(\sigma) \leq C^j(\delta); \hspace{1cm} 2 \leq j \leq m$$
and
\[ F(\sigma) \leq F(\delta), \]

then
\[ C^j(\sigma\pi) \leq C^j(\delta\pi); \quad 2 \leq j \leq m \]

and
\[ F(\sigma\pi) \leq F(\delta\pi). \]

Using the above results, we now describe some dominance conditions that are applicable to the two-machine flowshop problem only. For this purpose, without loss of generality, let \( J = (1, \ldots, n) \) be the Johnson’s schedule with makespan \( C(J) \). Consider a given partial schedule \( \sigma \), a single job \( i \) and an arbitrary partial schedule \( \pi \) of jobs not contained in \( \sigma \). Further, let \( J_\pi \) denote Johnson’s schedule for all jobs not in \( \sigma \). Then we get the following results.

**Theorem 3** (Potts (1976), Szwarc (1981)): Consider a schedule \( \sigma i\pi \). If
\[ C^1(\sigma) + a_i + \sum_{k \in \pi} b_k > C(J), \]

then
\[ C(\sigma i\pi) > C(J). \]

As a result of Theorem 3, job \( i \) need not be augmented to partial schedule \( \sigma \) since its makespan will be more than the minimum makespan. However, Theorem 3 can be strengthened by considering the effect of augmenting partial schedule \( \pi \) to \( \sigma i \).

**Theorem 4** (Potts (1976), Szwarc (1981)): Consider a schedule \( \sigma i\pi \). If
\[ C^1(\sigma) + C(iJ_\pi) > C(J), \]

then
\[ C(\sigma i\pi) > C(J). \]

In Theorems 3 and 4 we were considering the augmentation of a single job to a known partial schedule \( \sigma \). However, we can determine if a known initial partial schedule will produce a non-optimal makespan value. For this purpose, assume that job \( i \) is included somewhere in the arbitrary partial schedule \( \pi \) of jobs not contained in \( \sigma \).
Theorem 5 (Szwarc (1981)): For any schedule \( \sigma \pi \), we have

\[ C(\sigma \pi) \geq C(\sigma J) \]

Utilizing the results in Theorems 5 above and the fact that a feasible schedule to our problem must have a minimum makespan, we can directly state the following result.

Theorem 6 For any schedule \( \sigma \pi \), if

\[ C(\sigma J) > C(J) \]

then no complete schedule with \( \sigma \) as its initial partial schedule can be an optimal solution of the \( F2||Fh(\sum C_i/C_{\text{max}}) \) problem.

We now develop conditions that show if a pairwise interchange of jobs can reduce the makespan. For this purpose, let \( \alpha = (\alpha(1), \ldots, \alpha(q), \alpha(q+1), \ldots, \alpha(n)) \) be a minimum makespan schedule with \( I(\alpha) \) as the total idle time on second machine. Further, for any partial schedule \( \alpha_u = (\alpha(1), \alpha(2), \ldots, \alpha(u)) \), define:

\[ H(\alpha_u) = \sum_{i=1}^{u} a_{\alpha(i)} - \sum_{i=1}^{u-1} b_{\alpha(i)} \]

then, it is readily seen that:

\[ I(\alpha) = \max_{1 \leq u \leq n} \{ H(\alpha_u) \} \] (4)

Theorem 7: For two schedules \( \alpha = (\alpha(1), \ldots, \alpha(q), \alpha(q+1), \ldots, \alpha(n)) \) and \( \alpha' = (\alpha(1), \ldots, \alpha(q+1), \alpha(q), \ldots, \alpha(n)) \), if

\[ \max\{ H(\alpha_q) - a_{\alpha(q)} + a_{\alpha(q+1)}; H(\alpha_{q+1}) + b_{\alpha(q)} - b_{\alpha(q+1)} \} > I(\alpha), \] (5)

then

\[ C(\alpha') > C(\alpha). \] (6)

Proof. Let \( \alpha'_q = (\alpha(1), \alpha(2), \ldots, \alpha(q-1), \alpha(q+1)) \) and \( \alpha'_{q+1} = (\alpha(1), \alpha(2), \ldots, \alpha(q-1), \alpha(q+1), \alpha(q)) \). Then, from equation (3), it follows that:

\[ H(\alpha'_q) = H(\alpha_q) - a_{\alpha(q)} + a_{\alpha(q+1)} \]

and

\[ H(\alpha'_{q+1}) = H(\alpha_{q+1}) + b_{\alpha(q)} - b_{\alpha(q+1)} \] (8)
Further, from equation (4) it follows that:

\[ I(\alpha') \geq \max \{ H(\alpha_q'), H(\alpha_{q+1}') \} \]  

(9)

Substituting from equations (7) and (8) in (9) and using the conditions (5) of Theorem 7 shows that:

\[ I(\alpha') \geq I(\alpha). \]  

(10)

Since the sum of processing times of all jobs at the second machine is constant, (6) follows from (10). □

We now state sufficient conditions for a given schedule to be an optimal solution to any \( F_2||F_h(\sum C_i/C_{\text{max}}) \) problem.

**Theorem 8** The schedule \( \rho = (1,2,\ldots,n) \) satisfying the following conditions:

\[
\begin{align*}
\min \{a_i; b_j\} &\leq \min \{a_j; b_i\}; \quad 1 \leq i \leq j \leq n-1 \\
a_i &\leq a_j; \quad 1 \leq i \leq j \leq n-1 \\
b_i &\leq b_j; \quad 1 \leq i \leq j \leq n-1
\end{align*}
\]  

(11) (12) (13)

optimally solves the \( F_2||F_h(\sum C_i/C_{\text{max}}) \) problem.

**Proof.** Let \( \sigma = (1,2,\ldots,i-1) \) and \( \pi = (j+1,\ldots,n) \). Further, let \( i \) and \( j \) be two adjacent jobs. Thus, \( \rho = \sigma ij\pi \). Now consider another schedule \( \rho' = \sigma ji\pi \). Since conditions (11) represent Johnson’s schedule, it follows that schedule \( \rho \) has a minimal makespan value. Also, in view of conditions (11) above,

\[ C^2(\sigma ij) \leq C^2(\sigma ji) \]  

(14)

Now, from conditions (12) above it follows that

\[ C^1(\sigma i) = C^1(\sigma) + a_i \leq C^1(\sigma) + a_j = C^1(\sigma j) \]  

(15)

Further, in view of conditions (13) and equation (15) above

\[
\begin{align*}
C^2(\sigma i) &= \max \{C^1(\sigma i); C^2(\sigma)\} + b_i \\
&\leq \max \{C^1(\sigma j); C^2(\sigma)\} + b_j \\
&= C^2(\sigma j)
\end{align*}
\]  

(16)
Adding inequalities (14) and (16) yields:

\[ C^2(\sigma_i) + C^2(\sigma_{ij}) \leq C^2(\sigma_j) + C^2(\sigma_{ji}). \]

Therefore, the proof of Theorem 8 follows from Theorem 1. □

Theorem 8 gives a sufficient optimality condition but an optimal schedule need not to satisfy conditions (11) through (13). However, if a given schedule does satisfy conditions (11) through (13), Theorem 8 establishes its optimality.

### 3.2 Dominance Algorithms

Using the dominance conditions in Theorem 2, the following pure dominance algorithm can optimally solve the \( F|\sum C_i/C_{\text{max}} \) problem containing \( m \) stages.

**Pure Dominance Algorithm**

**Step 0.** Let \( L = 1 \) and \( S = \{1, 2, \ldots, n\} \) be the set of \( n \) partial schedules each consisting only of one job \( i \). Compute \( C^j(\rho), 1 \leq j \leq m \), and \( F(\rho) \forall \rho \in S \). Enter step 1.

**Step 1.** For each partial schedule \( \rho \in S \), form \( n - L \) partial schedules consisting of \( L + 1 \) jobs by assigning each of the \( n - L \) unassigned jobs at sequence position \( L + 1 \). Let the set of partial schedules so formed be \( S \). Compute \( C^j(\rho), 1 \leq j \leq m \), and \( F(\rho) \forall \rho \in S \).

**Step 2.** Group the partial schedules containing the same subset of jobs together, i.e., let \( S = \{S_1, S_2, \ldots, S_k\} \), where for each \( i \leq k \), \( S_i \) contains all partial schedules containing the same subset of jobs. For each subset \( S_i \), test the following:

(a) If for two partial schedules \( \sigma, \sigma' \in S_i \), the conditions \( C^j(\sigma) \leq C^j(\sigma'), 1 \leq j \leq m \), and \( F(\sigma) \leq F(\sigma') \) hold, eliminate \( \rho \) from \( S_i \).

(b) Repeat step 2(a) for other \( \sigma, \sigma' \in S_i \) until it is no longer possible to reduce the subset \( S_i \). Enter step 3.

**Step 3.** Let \( L = L + 1 \). If \( L < n - 1 \), return to step 1. Otherwise for each \( \sigma \in S_{n-1} \), let \( \sigma_i \) be the complete schedule obtained by augmenting the job \( i \notin \sigma \). Compute \( C^j(\sigma_i), 1 \leq j \leq m \), of the schedules so obtained. Among the schedules with minimum \( C^m(\sigma_i) \), a schedule that minimizes \( F(\sigma) \) is an optimal schedule.

While the above algorithm will optimally solve the \( F|\sum C_i/C_{\text{max}} \) problem, it is, however, not likely to be an efficient algorithm. Using Theorem 6, we can improve the efficiency of the pure dominance algorithm to optimally solve the \( F|\sum C_i/C_{\text{max}} \) problem. The steps of such an improved dominance algorithm are as follows:
Improved Dominance Algorithm

Step 0. Let $L = 1$ and $S = \{1, 2, \ldots, n\}$ be the set of $n$ partial schedules each consisting only of one job $i$. Compute $C^j(\rho), 1 \leq j \leq 2$, and $F(\sigma) \forall \rho \in S$. Enter step 1.

Step 1. For each partial schedule $\rho \in S$, form $n - L$ partial schedules consisting of $L + 1$ jobs by assigning each of the $n - L$ unassigned jobs at sequence position $L + 1$. Let the set of partial schedules so formed be $S$. Enter step 2.

Step 2. Compute $C^j(\rho), 1 \leq j \leq 2$, and $\forall \rho \in S$. For any $\rho \in S$, if $C^2(\rho J) > C(J)$, eliminate $\rho$ from $S$. Enter step 3.

Step 3. For each $\rho \in S$, compute $F(\rho)$. Group the partial schedules containing the same subset of jobs together, i.e., let $S = \{S_1, S_2, \ldots, S_k\}$, where for each $i \leq k$, $S_i$ contains all partial schedules containing the same subset of jobs. For each subset $S_i \in S$, let $\rho_i$ be the schedule such that $F(\rho_i) = \min_{\sigma_i \in S_i} \{F(\sigma_i)\}$. Let $S = \{\rho(1), \ldots, \rho(k)\}$. Enter step 4.

Step 4. Let $L = L + 1$. If $L < n - 1$, return to step 1; otherwise for each $\sigma \in S_{n-1}$, let $\sigma i$ be the complete schedule obtained by augmenting the job $i \notin \sigma$. Compute $C^j(\sigma i), 1 \leq j \leq 2$, of the schedules so obtained. Among the schedules with minimum $C^2(\sigma i)$, a schedule that minimizes $F(\sigma i)$ is an optimal schedule.

While the improved dominance algorithm is relatively more efficient than the pure dominance algorithm, its computational burden is still exponential as in the best case, it will retain $\sum_{r=1}^n \binom{n}{r}$ partial schedules, where $\binom{n}{r}$ is the number of combinations of $r$ items taken out of $n$ items.

4 Polynomials Solvable Special Cases

We now describe polynomially bounded optimization algorithms for two special cases of the $F2||F_h(\sum C_i/C_{\max})$ problem.

Case 1:

We start with the ordered matrix problems considered by Smith et al. (1975) and Panwalkar & Khan (1976). For these problems, the processing times satisfy the following conditions: for each job $i \in N, a_i \leq b_i$ and for all $i$ and $j \in N, a_i \leq a_j$ implies $b_i \leq b_j$.

Theorem 9 For the $F2||F_h(\sum C_i/C_{\max})$ problem satisfying conditions of case 1, an optimal schedule is obtained by arranging jobs in non-descending order of their processing time on second machine.
**Proof.** It is readily seen that the processing times of case 1 satisfy the conditions of Theorem 8. Therefore, the proof of Theorem 9 follows from Theorem 8. □

**Case 2:**

For this case, the processing times satisfy the following conditions:

$$\min_{1 \leq i \leq n} \{a_i\} \geq \max_{1 \leq j \leq n} \{b_j\}.$$  \hspace{1cm} (17)

**Theorem 10** For the $F^2/F_h(\sum C_i/C_{\text{max}})$ problem satisfying conditions of case 2, an optimal schedule is obtained by arranging all jobs in ascending order of their processing times on the first machine with the last job selected with minimum processing time at the second stage where ties for the last job are broken in favor of a job with largest processing time on the first machine.

**Proof.** From equation (1), the makespan $C_{\text{max}}$ of an arbitrary $\sigma = (\sigma(1), \ldots, \sigma(n))$ is given by

$$C_{\text{max}} = \max_{1 \leq u \leq n} \left\{ \sum_{i=1}^{u} a_{\sigma(i)} + \sum_{i=u}^{n} b_{\sigma(i)} \right\}$$ \hspace{1cm} (18)

and in view of the special conditions (17) of case 2, equation (18) simplifies to:

$$C_{\text{max}} = \sum_{i=1}^{n} a_{\sigma(i)} + b_{\sigma(n)}$$ \hspace{1cm} (19)

Since $\sum_{i=1}^{n} a_{\sigma(i)}$ in equation (19) is a constant and is independent of the sequence of jobs, it follows that a schedule where the last job has minimum processing time on the second machine minimizes makespan. Further, except for ties for $\min_{1 \leq i \leq n} b_i$ values, $\sigma(n)$ is the unique last job in a minimum makespan schedule.

Now, using equation (1) and conditions (17) of case 2, the completion time of job $\sigma(u)$ in schedule $\sigma$, $C_{\sigma(u)}$ is given by

$$C_{\sigma(u)} = \sum_{i=1}^{u} a_{\sigma(i)} + b_{\sigma(u)}$$ \hspace{1cm} (20)

Therefore, using equations (2) and (20), the total flow time of schedule $\sigma$, $F(\sigma)$ is expressed as:

$$F(\sigma) = \sum_{u=1}^{n} C_{\sigma(u)} = \sum_{u=1}^{n} \sum_{i=1}^{u} a_{\sigma(i)} + \sum_{u=1}^{n} b_{\sigma(u)}$$ \hspace{1cm} (21)

Since $\sum_{u=1}^{n} b_{\sigma(u)}$ in equation (21) is constant and is independent of the sequence, it follows that the schedule arranged in ascending order of their processing times minimizes total
flow time. Since makespan is the primary criterion, arranging jobs in an ascending order of their processing times on the first machine with the last job selected with minimum processing time at the second stage optimally solves the $F_2||F_h(\sum C_i/C_{\text{max}})$ problem satisfying conditions of case 2. In this process, ties for the last job are broken in favor of a job with largest processing time on the first machine. □

5 Heuristic Algorithms

In this section, we describe several polynomial heuristic algorithms to find approximate solutions to the $F_2||F_h(\sum C_i/C_{\text{max}})$ problem. Solutions obtained from these heuristics can be used as upper bounds in an optimization algorithm or as an initial solution in a meta-heuristic algorithm. Moreover, when the computational effort to optimally solve the problem with an exact optimization algorithm is prohibitive, these heuristics provide good quality solutions at little computational effort.

5.1 Constructive Heuristic Algorithms

We start the development of the heuristic algorithms by using a weaker form of the conditions in Theorems 2, 5, and 6 including the fact that the augmentation of Johnson’s sequence to a partial schedule always minimizes makespan. Following the approach outlined by Gupta (1972a), we relax the requirement that partial schedules have to be different permutations of the same subset of jobs. Starting with each job at the first sequence position, we append each job to a known partial sequence and select that feasible partial sequence which has minimum total flow time. Thus, our first heuristic is as follows:

Algorithm Append1:

1. Let $J = (1, \ldots, n)$ be Johnson’s schedule of all $n$ jobs. Compute $C(J)$ and $F(J)$. Set $\alpha = J, s = i = 1$ and $\sigma = (1)$. Enter step 2.

2. Generate $n - i$ schedules by appending each job $k \notin \sigma$ to partial schedule $\sigma$ to get a partial schedule $\sigma k$. Let $\omega = \{\omega_1, \omega_2, \ldots, \omega_r\}$ be the set of all partial schedules $\sigma k$ with $C(\sigma k J_{\sigma k}) = C(J)$; If $\omega = \emptyset$, go to step 4; otherwise enter step 3.

3. Determine $\omega_q J_{\omega_q}$ with $F(\omega_q J_{\omega_q}) = \min_{1 \leq j \leq r}\{F(\omega_j J_{\omega_j})\}$. If $F(\omega_q J_{\omega_q}) < F(\alpha)$, set $\alpha = (\omega_q J_{\omega_q})$. Determine $\omega_\nu$ with $F(\omega_\nu) = \min_{1 \leq j \leq r}\{F(\omega_j)\}$. Set $\sigma = \omega_\nu$ and $i = i + 1$. If $i = n$, enter step 4; otherwise return to step 2.
4. If \( s < n \), set \( s = s + 1 \), \( \sigma = (s) \), and \( i = 1 \) and return to step 2; otherwise accept the schedule \( \alpha \) with total flow time \( F(\alpha) \) and makespan \( C(\alpha) \) as the solution of the problem.

Algorithm \textit{Append1} requires \( O(n^4) \) time and is a form of greedy heuristic. The computational requirements of Algorithm \textit{Append1} can be decreased if, instead of generating schedules for each job in the first sequence position, we first select a specific feasible pair to start the sequence. The steps of our second heuristic algorithm are as follows:

**Algorithm \textit{Append2}:**

1. Let \( J = (1, \ldots, n) \) be Johnson’s schedule of all \( n \) jobs. Compute \( C(J) \) and \( F(J) \). Set \( \alpha = J \). Enter step 2.

2. Generate \( n \times (n-1) \) job pairs \((ab)\) with \( 1 \leq a, b \leq n, a \neq b \). Let \( \omega = \{\omega_1, \omega_2, \ldots, \omega_r\} \) be the set of all job pairs \((ab)\) with \( C(abJ_{ab}) = C(J) \). Enter step 3.

3. Determine \( \omega_q J_{\sigma q} \) with \( F(\omega_q J_{\sigma q}) = \min_{1 \leq j \leq r} \{F(\omega_j J_{\sigma j})\} \). If \( F(\omega_q J_{\sigma q}) < F(\alpha) \), set \( \alpha = (\omega_q J_{\sigma q}) \). Determine \( \omega_v \) with \( F(\omega_v) = \min_{1 \leq j \leq r} \{F(\omega_j)\} \). Set \( \sigma = \omega_v \) and \( i = i + 1 \). Enter step 4.

4. Generate \( n - i \) schedules by appending each job \( k \notin \sigma \) to partial schedule \( \sigma \) to get a partial schedule \( \sigma k \). Let \( \omega = \{\omega_1, \omega_2, \ldots, \omega_r\} \) be the set of all partial schedules \( \sigma k \) with \( C(\sigma k J_{\sigma k}) = C(J) \). Enter step 5.

5. Determine \( \omega_q J_{\sigma q} \) with \( F(\omega_q J_{\sigma q}) = \min_{1 \leq j \leq r} \{F(\omega_j J_{\sigma j})\} \). If \( F(\omega_q J_{\sigma q}) < F(\alpha) \), set \( \alpha = (\omega_q J_{\sigma q}) \). Determine \( \omega_v \) with \( F(\omega_v) = \min_{1 \leq j \leq r} \{F(\omega_j)\} \). Set \( \sigma = \omega_v \) and \( i = i + 1 \). If \( i < n \), return to step 4; otherwise accept the schedule \( \alpha \) with total flow time \( F(\alpha) \) and makespan \( C(\alpha) \) as the solution of the problem.

We note that time requirement of Algorithm \textit{Append2} is \( O(n^3) \) and it is a significant reduction from that required for Algorithm \textit{Append1}. However, its effectiveness may be increased by inserting jobs at various sequence positions in a known partial schedule (as in the algorithm by Nawaz et al. (1983) in addition to appending these jobs at the end of a partial schedule. Therefore, in the next heuristic, a job is inserted at each sequence position of a known partial schedule to find feasible schedules, out of which the most promising (determined by least total flow time) partial schedule is selected for further search. The specific steps of this algorithm are as follows:
Algorithm *Insert*:

1. Let \( J = (1, \ldots, n) \) be Johnson’s schedule of all \( n \) jobs. Compute \( C(J) \) and \( F(J) \).
   Set \( \alpha = J, i = 1 \) and \( \sigma = (1) \). Further, let \( \sigma = \sigma_p \sigma_{i-p} \) for \( 0 \leq p \leq i \). Enter step 2.

2. For each of the \( n-i \) jobs \( \not\in \sigma \), generate \( (n-i)*(i+1) \) partial sequences represented by \( \sigma_p k \sigma_{i-p} \), where \( k \not\in \sigma \) and \( 0 \leq p \leq i \). Let \( \omega = \{\omega_1, \omega_2, \ldots, \omega_r\} \) be the set of all partial schedules \( \sigma_p k \sigma_{i-p} \). Enter step 3.

3. Determine \( \omega_q \) with \( F(\omega_q) = \min_{1 \leq j \leq r} \{ F(\omega_j) \} \). If \( F(\omega_q) < F(\alpha) \), set \( \alpha = (\omega_q) \). Determine \( \omega_k \) with \( F(\omega_k) = \min_{1 \leq j \leq r} \{ F(\omega_j) \} \). Set \( \sigma = \omega_k \) and \( i = i + 1 \). Enter step 4.

4. If \( i < n \), set \( i = i + 1 \) and return to step 2; otherwise accept the schedule \( \alpha \) with total flow time \( F(\alpha) \) and makespan \( C(\alpha) \) as the solution of the problem.

Algorithm *Insert* time requirement is \( O(n^4) \). However, its performance is likely to be better than the earlier two algorithms.

### 5.2 Controlled Insertion Heuristic Algorithms

In the following two heuristic procedures, we adapted a modified version of Algorithm *Insert* with an initial sequence directing the next job for consideration. The choice of Algorithm *Insert* is due to its superior performance over the other two constructive heuristics (discussed in the next section). Hence in the following heuristics, the next job to be considered for inserting into the partial sequence is guided by an initial sequence. One obvious way to do this is to use Johnson’s schedule as the initial schedule that guides the selection of next job to be inserted.

Algorithm *CI-J*:

1. Let \( J = (1, \ldots, n) \) be Johnson’s schedule of all \( n \) jobs. Compute \( C(J) \) and \( F(J) \).
   Set \( \alpha = J, i = 1 \) and \( \sigma = (1) \). Further, let \( \sigma = \sigma_p \sigma_{i-p} \) for \( 0 \leq p \leq i \). Enter step 2.

2. Generate \( n*(n-1) \) job pairs \( (ab) \) with \( 1 \leq a, b \leq n, a \neq b \). Let \( \omega = \{\omega_1, \omega_2, \ldots, \omega_r\} \) be the set of all job pairs \( (ab) \) with \( C(ab) = C(J) \). Determine \( \omega_q \) with \( F(\omega_q) = \min_{1 \leq j \leq r} \{ F(\omega_j) \} \). If \( F(\omega_q) < F(\alpha) \), set \( \alpha = (\omega_q) \). Determine \( \omega_r \) with \( F(\omega_r) = \min_{1 \leq j \leq r} \{ F(\omega_j) \} \). Set \( \sigma = \omega_r \), \( s = 0 \) and \( i = i + 1 \). Enter step 3.

3. Set \( s = s + 1 \). Enter step 4.
4. If \( s \in \sigma \), return to step 3, otherwise generate \( i + 1 \) partial sequences represented by \( \sigma_p s \sigma_{i-p}, 0 \leq p \leq i \). Let \( \omega = \{\omega_1, \omega_2, \ldots, \omega_r\} \) be the set of all \( \sigma_p s \sigma_{i-p} \) with \( C(\sigma_p s \sigma_{i-p}, J) = C(J) \). Enter step 5.

5. Determine \( \omega_q J \omega_q \) with \( F(\omega_q J \omega_q) = \min_{1 \leq j \leq r} \{F(\omega_j J \omega_j)\} \). If \( F(\omega_q J \omega_q) < F(\alpha) \), set \( \alpha = (\omega_q J \omega_q) \). Determine \( \omega_\nu \) with \( F(\omega_\nu) = \min_{1 \leq j \leq r} \{F(\omega_j)\} \). Set \( \sigma = \omega_\nu, s = 0 \) and \( i = i + 1 \). If \( i < n \), then return to step 3; otherwise accept the schedule \( \alpha \) with total flow time \( F(\alpha) \) and makespan \( C(\alpha) \) as the solution of the problem.

The time requirement of Algorithm \( CI-J \) is \( O(n^3) \). Its performance can be improved by considering an initial schedule that meets various conditions for optimality. This is done by ordering jobs according to the total number of times a job dominates other jobs. The steps of such an algorithm are as follows:

**Algorithm \( CI-JDM \):**

1. Let \( J = (1, \ldots, n) \) be Johnson’s schedule of all \( n \) jobs. Compute \( C(J) \) and \( F(J) \). Set \( \alpha = J \). For each job \( k, 1 \leq k \leq n \), set \( nn(k) = 0 \). For each job pair \( (ab) \) with \( 1 \leq a, b \leq n, a \neq b \), if \( F(ab) \leq F(ba) \), then \( nn(a) = nn(a) + 1 \), else \( nn(b) = nn(b) + 1 \). Let \( \pi = (\pi(1), \ldots, \pi(n)) \) such that \( nn(\pi(l)) \leq nn(\pi(l + 1)) \), where \( 1 \leq l \leq n - 1 \). Further, let \( \sigma = \sigma_p \sigma_{i-p} \) for \( 0 \leq p \leq i \). Enter step 2.

2. Steps 2 through 5, same as those in Algorithm \( CI-J \).

The time requirements of algorithm \( CI-J \) is \( O(n^3) \).

### 5.3 Iterative Improvement Heuristics

We now consider fast polynomial time iterative improvement procedures for finding approximate solutions to the problem. In these procedures an initial sequence is considered and perturbed based on some dominance conditions in order to improve the quality of the sequence. In these procedures, dominance relations are used in estimating the job and/or job positions to perturb. First such heuristic uses Johnson’s schedule and performs pairwise interchanges to improve this schedule. Since the computational effort of pairwise interchanges may be excessive, the maximum number of interchanges is defined as twice the number of jobs. The steps of this algorithm are as follows:

1. Let \( J = (1, \ldots, n) \) be Johnson’s schedule of all \( n \) jobs. Compute \( C(J) \) and \( F(J) \). Set \( \alpha = J \). For each job \( k, 1 \leq k \leq n \), set \( nn(k) = 0 \). For each job pair \( (ab) \) with \( 1 \leq a, b \leq n, a \neq b \), if \( F(ab) \leq F(ba) \), then \( nn(a) = nn(a) + 1 \), else \( nn(b) = nn(b) + 1 \). Let \( \pi = (\pi(1), \ldots, \pi(n)) \) such that \( nn(\pi(l)) \leq nn(\pi(l + 1)) \), where \( 1 \leq l \leq n - 1 \). Further, let \( \sigma = \sigma_p \sigma_{i-p} \) for \( 0 \leq p \leq i \). Enter step 2.

2. Steps 2 through 5, same as those in Algorithm \( CI-J \).
Algorithm STR:

1. Let $J = (1, \ldots, n)$ be Johnson’s schedule of all $n$ jobs. Compute $C(J)$ and $F(J)$. Set $\alpha = J$, $max\_iter = 2 * n$, $iter = 0$, and $r = n$. Enter step 2.

2. Swap the jobs in sequence positions $r$ and $r - 1$ to obtain the sequence $\alpha' = (\alpha(1), \ldots, \alpha(r), \alpha(r - 1), \ldots, \alpha(n))$. Compute $F(\alpha')$ and $C(\alpha')$.

3. If $C(\alpha') = C(\alpha)$ and $F(\alpha') < F(\alpha)$, then $\alpha = \alpha'$. If $r > 2$, set $r = r - 1$ and return to step 2; otherwise enter step 3.

4. If $iter < max\_iter$, set $r = n$, $iter = iter + 1$ and return to step 2; otherwise accept the schedule $\alpha$ with total flow time $F(\alpha)$ and makespan $C(\alpha)$ as the solution of the problem.

The above algorithm requires $O(n^3)$ time and does not use any dominance conditions to reduce the number of interchanges made. This can be done by using the conditions of Theorem 2. The steps of the algorithm to do so are as follows:

Algorithm DOM:

1. Let $J = (1, \ldots, n)$ be Johnson’s schedule of all $n$ jobs. Compute $C(J)$ and $F(J)$. Set $\alpha = J$ and $r = n$. Enter step 2.

2. For $\alpha_r = (\alpha(1), \ldots, \alpha(r - 1), \alpha(r))$, compute $F(\alpha_r)$ and $C(\alpha_r)$. Swap the jobs in sequence positions $r$ and $r - 1$ to obtain the sequence $\alpha' = (\alpha(1), \ldots, \alpha(r), \alpha(r - 1), \ldots, \alpha(n))$. Compute $F(\alpha'_r)$, $F(\alpha')$, $C(\alpha'_r)$ and $C(\alpha')$. Enter step 3.

3. If $C(\alpha'_r) \leq C(\alpha_r)$ and $F(\alpha'_r) \leq F(\alpha_r)$ or $C(\alpha') = C(\alpha)$ and $F(\alpha') < F(\alpha)$, set $\alpha = \alpha'$, $r = n$ and return to step 2; otherwise enter step 4.

4. If $r > 2$, then set $r = r - 1$ and return to step 2; otherwise accept the schedule $\alpha$ with total flow time $F(\alpha)$ and makespan $C(\alpha)$ as the solution of the problem.

The time requirement of Algorithm DOM is $O(n^4)$.

The last heuristic in this category is based on the one proposed by Rajendran (1993), which considers Johnson’s schedule and iteratively improves the schedule based on the total flow time of the interchanged schedule. To describe this algorithm, consider a schedule $\alpha = (\alpha(1), \ldots, \alpha(n))$ and for each $r$, $1 \leq r \leq n$, define:

$$D_r = a_{\alpha(r)} + b_{\alpha(r)} - a_{\alpha(r+1)} - b_{\alpha(r+1)}$$
\[ D'_r = 2a_\alpha(r) + b_\alpha(r) - 2a_\alpha(r+1) - b_\alpha(r+1). \]

Further, define the following:

\[ H_\alpha(u) = \sum_{j=1}^{u} a_\alpha(j) - \sum_{j=1}^{u-1} b_\alpha(j). \]

Then, the heuristic based on Rajendran’s work is as follows:

**Algorithm RAJ:**

1. Let \( J = (1, \ldots, n) \) be Johnson’s schedule of all \( n \) jobs. Compute \( C(J) \) and \( F(J) \). Set \( \alpha = J \). Let \( I(\alpha) = \max_{1 \leq u \leq n} \{ H_\alpha(u) \} \) and \( p = \min_{1 \leq u \leq n} \{ u \mid I(\alpha) = H_\alpha(u) \} \). Set \( D_{p-1} = D_p = D_n = -1 \). For each \( r \) with \( 1 \leq r \leq n \) and \( r \notin \{ p - 1, p, n \} \), compute \( D(r) \) and \( D'_r \). Enter step 2.

2. Let \( \omega \) be the set of all \( \alpha_r \) with \( 1 \leq r \leq n \) and \( D_{\alpha(r)} \geq 0 \). If \( \omega = \emptyset \), go to step 5; otherwise let \( \omega = \{ \omega(1), \ldots, \omega(s) \} \) be such that either \( D_{\omega(k)} > D_{\omega(k+1)} \) or \( D_{\omega(k)} = D_{\omega(k+1)} \) and \( D'_{\omega(k)} > D'_{\omega(k+1)} \), where \( 1 \leq k \leq s - 1 \). Set \( t = 1 \) and enter step 3.

3. Find \( q \) such that \( \alpha(q) = \omega(t) \). If \( \max \{ H_{\alpha(q)} - a_{\alpha(q)} + a_{\alpha(q+1)}; H_{\alpha(q+1)} + b_{\alpha(q)} - b_{\alpha(q+1)} \} \leq I(\alpha) \), enter step 4; otherwise go to step 5.

4. Swap the jobs in sequence positions \( q \) and \( q + 1 \) to obtain the sequence \( \alpha' = (\alpha(1), \ldots, \alpha(q+1), \alpha(q), \ldots, \alpha(n)) \). If \( F(\alpha') \geq F(\alpha) \), enter step 5; otherwise, set \( \alpha = \alpha' \) and for \( r = q - 1, q, q+1 \), compute \( D(r) \) and \( D'_r \), update \( H_{\alpha(q)} \) and \( H_{\alpha(q+1)} \), and return to step 2.

5. If \( t < s \), set \( t = t + 1 \) and return to step 3; otherwise accept the schedule \( \alpha \) with total flow time \( F(\alpha) \) and makespan \( C(\alpha) \) as the solution of the problem.

Algorithm RAJ requires \( O(n^4) \) time.

### 5.4 Combined Heuristic

The controlled insertion and iterative improvement procedures can be combined to seek a better heuristic. This is done by combining Algorithm RAJ and the controlled insertion algorithms described earlier. The steps of such a combined algorithm are as follows:
**Algorithm CI-RAJ:**

1. Let $J = (1, \ldots, n)$ be Johnson’s schedule of all $n$ jobs. Compute $C(J)$ and $F(J)$. Set $\alpha = J$. Let $\pi = (\pi(1), \ldots, \pi(n))$ be the schedule obtained by Algorithm RAJ. Further, let $\sigma = \sigma_p \sigma_{i-p}$ for $0 \leq p \leq i$. Enter step 2.

2. Steps 2 through 5, same as those in algorithm CI-J.

The time requirement of Algorithm CI-RAJ is $O(n^4)$.

6 **Computational Results**

We now describe the computational results of the empirical evaluation of the proposed algorithms. Since the effectiveness of the existing genetic algorithm (Nepalli et al. (1996)) in finding optimal or near optimal solutions to the $F2||F_h(\sum C_i/C_{\text{max}})$ problem has been shown to be inferior to the job insertion based heuristics (see also Gupta et al. (1999)), only the nine polynomially bounded heuristic algorithms described in the previous section are evaluated in terms of their relative effectiveness in solving the $F2||F_h(\sum C_i/C_{\text{max}})$ problem.

6.1 **Test Problems**

In order to analyze the effect of domination of the first and second machines the following three classes of problems are considered in this study:

- **Class 1**: Both machines are equivalent (the range of processing times on the first machine is $U(1,99)$ and on the second machine is $U(1,99)$, where $U(1,99)$ means that the processing times are uniformly distributed integers from the set $\{1, 2, \ldots, 99\}$);

- **Class 2**: the first machine dominates the second machine (the range of processing times on the first machine is $U(1,99)$ and on the second machine is $U(1,49)$); and

- **Class 3**: the second machine dominates the first machine (the range of processing times on the first machine is $U(1,49)$ and on the second machine is $U(1,99)$).

The number of jobs (problem size) varied from $n = 10$ to $n = 80$. For each problem configuration 50 test problems were generated. All the heuristic procedures were implemented in FORTRAN77 and run on a SUN4/490 machine.
6.2 Comparing Heuristics for Small Problems

Computational effort to optimally solve the problem using the dominance algorithms described in Section 3 became quite prohibited for problems containing more than 10 jobs. Therefore, this section reports results only for problems with 10 jobs. The optimal solutions to all 10-job problems were found by the improved dominance algorithm in Section 3. For each problem class and each heuristic, Table 1 reports the minimum, average and maximum percentage deviation of the solution values of a particular heuristic algorithm from the optimal solution value.

The results in Table 1 can be used to compare the effectiveness of the nine heuristics on these 10-job problems. The average percentage deviations of the heuristic solution value from its optimal value is within 2 percent for all heuristic algorithms, generally less than 1 percent for most heuristics. Further, the results in Table 1 show that the heuristic Insert yields the lowest average and maximum percentage deviation values for problem classes 2 and 3 (less than quarter of a percent). For class 1 problems, its performance is second best, not too different from the best of the remaining eight heuristics (less than three-fifth of a percent). Based on these empirical results, we conclude that the heuristic Insert is best suited to solve small sized problems.

6.3 Comparing Heuristics for Large Problems

In order to perform the comparative analysis of these nine heuristic algorithms for large sized problems, the following two performance measures were used:

- the number of times the procedure provided the minimum solution value (minimum flow time subject to obtaining the optimal makespan) among the nine heuristic algorithms; and
- the Average Relative Performance (ARP), which is the average of the deviations of the solution values of a particular heuristic algorithm from the best solution values obtained among all the nine heuristic algorithms.

The Average Relative Performance (ARP) of heuristic algorithm $i$ is computed by using the following equation:

$$ARP_i = \left\{ \frac{\sum_{j=1}^{nprobs} \{ (F_{ij} - F_j^*) / F_j^* \} * 100}{nprobs} \right\} / nprobs,$$
where $F_{ij}$ is the flow time provided by heuristic algorithm $i$ for problem $j$, $F_{ij}^*$ is the best flow time obtained for the problem $j$ among the nine heuristic algorithms and $nprobs$ is the number of tested problems.

Table 2 presents the number of times each heuristic algorithm produced the minimum solution value (minimum flow time subject to obtaining optimal makespan) among the compared nine heuristic algorithms. From the results in Table 2, it is evident that Heuristic Insert clearly dominates the rest of the heuristics. Out of the tested 1500 problems, Heuristic Insert produced minimum solution values in 866 problems which is equal to 57.73%. Furthermore, it is also evident that the performance of Heuristic Insert across all the problem classes is consistent. Heuristic CI-RAJ produced 563 minimum solutions among the tested 1500 problems and is the next best heuristic algorithm among the nine heuristic algorithms. Heuristics Append1 and Append2 performed reasonably well for class 3 problems (when the second machine dominates the first machine), however, performed poorly for class 1 and 2 problems. Heuristic STR performed better than both, Heuristic RAJ (Rajendran’s Heuristic developed for the candidate bicriteria problem) and Heuristic DOM (the heuristic algorithm developed based on the dominance relations proposed in this study). Further, Heuristic STR performed reasonably well for class 2 problems, when compared with Heuristics Insert and CI-RAJ.

--- Insert Table 2 about here ---

Table 3 presents the average relative performance of the nine heuristics. Similar conclusions to those of Table 2 can be drawn from the results in Table 3. The average relative performance of both the Heuristics Insert and CI-RAJ dominated the rest of the heuristic algorithms. The ARP of both the Heuristics Insert and CI-RAJ varied from 0.01 to 0.26 with an overall ARP of 0.08 % and 0.12 % for Heuristics Insert and CI-RAJ, respectively. The superior performance of Heuristic Insert can be attributed to the limitation set on the number of solutions searched in each iteration by Heuristic CI-RAJ when compared with Heuristic Insert. Heuristic STR performed reasonably well for solving the class 2 problems with respect to the ARP measure also. Along with Heuristic STR, Heuristics CI-JDM and DOM are appealing on class 2 problems.

--- Insert Table 3 about here ---
Finally, in Table 4 the computational requirements of each of the heuristics are presented. Except Heuristic $\text{Append1}$, the computational requirements of all the heuristics were steady across all problem classes. The computational requirements of Heuristic $\text{Append1}$ varied with the problem class, since the heuristic procedure is based on the number of feasible sequences possible when each job is considered in the first job position. From the computational requirements point of view, the superior performance of Heuristic $\text{Insert}$ over Heuristic $\text{CI-RAJ}$ is justified with its increased CPU requirements. Further, the CPU requirements of Heuristic $\text{Insert}$ are reasonable even to solve large size problems. Hence Heuristic $\text{Insert}$ is preferred over Heuristic $\text{CI-RAJ}$.

7 Conclusions

This paper considered the two-machine flowshop scheduling problem of minimizing the total flow time subject to the condition that the makespan of the schedule is minimum. In view of the NP-hardness of the problem, some polynomially solvable cases were identified and solved. Based on the dominance conditions, two optimization algorithms were developed. Several heuristic polynomial algorithms were developed and evaluated for finding approximate solutions to the problem. From the comparative analysis of the nine heuristic algorithms including the known polynomially bounded heuristic for this problem (Rajendran (1993)), it is evident that Heuristic $\text{Insert}$ which is based on the procedure of the algorithm by Nawaz et al. (1983), outperforms the rest of the heuristics with reasonable computational requirements. This is in correspondence with other scheduling problems, where job or operation insertion algorithms also yield the best results among constructive algorithms (see for instance Sotskov et al. (1999), Werner & Winkler (1995)).

The polynomially bounded algorithms discussed in this paper (in particular Heuristic $\text{Insert}$) may be used to find an initial solution in a local search algorithm, which may improve its performance. Also, since the polynomial algorithms perform rather well (see for instance Table 1), one could use several initial solutions obtained by different polynomial algorithms presented in this paper to perform a multi-start iterative algorithm to generate near-optimal solutions. However, very recent investigations (see Gupta et al. (1999)) have shown that all of the tested local search algorithms have obtained average percentage improvements over the solution generated by the insertion algorithm of clearly less than 1 % for the problem considered here (this result is in contrast to other weighted secondary criteria, where local search algorithms significantly improve the quality of the
insertion solution). The latter observations once more confirm the excellent solution quality of the insertion algorithms presented in this paper for $C2 = \sum C_i$.

Further research can be extended to evaluate the effectiveness of the proposed first three heuristics for other bicriteria problems in order to yield a general procedure for solving the bicriteria scheduling problems. Another important area of research is to investigate the means of developing efficient dominance conditions and using them in local search algorithms.

References


Table 1: The relative performance of the nine heuristics when compared with an optimal solution for the 10 job problems

<table>
<thead>
<tr>
<th>Method</th>
<th>class 1</th>
<th></th>
<th>class 2</th>
<th></th>
<th>class 3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Aver</td>
<td>Max</td>
<td>Min</td>
<td>Aver</td>
<td>Max</td>
</tr>
<tr>
<td>Append1</td>
<td>0.00</td>
<td>1.09</td>
<td>8.12</td>
<td>0.00</td>
<td>0.88</td>
<td>3.25</td>
</tr>
<tr>
<td>Append2</td>
<td>0.00</td>
<td>1.36</td>
<td>8.12</td>
<td>0.00</td>
<td>1.37</td>
<td>5.92</td>
</tr>
<tr>
<td>Insert</td>
<td>0.00</td>
<td>0.58</td>
<td>8.12</td>
<td>0.00</td>
<td>0.07</td>
<td>1.40</td>
</tr>
<tr>
<td>CI-RAJ</td>
<td>0.00</td>
<td>0.56</td>
<td>5.75</td>
<td>0.00</td>
<td>0.25</td>
<td>5.29</td>
</tr>
<tr>
<td>CI-JDM</td>
<td>0.00</td>
<td>0.97</td>
<td>5.75</td>
<td>0.00</td>
<td>0.25</td>
<td>5.29</td>
</tr>
<tr>
<td>CI-J</td>
<td>0.00</td>
<td>2.61</td>
<td>11.90</td>
<td>0.00</td>
<td>0.59</td>
<td>6.60</td>
</tr>
<tr>
<td>RAJ</td>
<td>0.00</td>
<td>3.32</td>
<td>13.40</td>
<td>0.00</td>
<td>0.56</td>
<td>6.04</td>
</tr>
<tr>
<td>DOM</td>
<td>0.00</td>
<td>2.26</td>
<td>11.85</td>
<td>0.00</td>
<td>0.39</td>
<td>3.44</td>
</tr>
<tr>
<td>STR</td>
<td>0.00</td>
<td>1.90</td>
<td>11.85</td>
<td>0.00</td>
<td>0.19</td>
<td>2.56</td>
</tr>
<tr>
<td>$n$</td>
<td>Proc. times</td>
<td>Constructive heuristics</td>
<td>Controlled insertion heuristics</td>
<td>Dominance based heuristics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-------------</td>
<td>-------------------------</td>
<td>---------------------------------</td>
<td>--------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$Append_1$</td>
<td>$Append_2$</td>
<td>$Insert$</td>
<td>$CI$-RAJ</td>
<td>$CI$-JDM</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>class1</td>
<td>26</td>
<td>24</td>
<td>36</td>
<td>28</td>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td>class2</td>
<td>10</td>
<td>7</td>
<td>43</td>
<td>39</td>
<td>39</td>
<td>33</td>
</tr>
<tr>
<td>class3</td>
<td>40</td>
<td>40</td>
<td>44</td>
<td>40</td>
<td>34</td>
<td>29</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>class1</td>
<td>14</td>
<td>13</td>
<td>26</td>
<td>20</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>class2</td>
<td>1</td>
<td>1</td>
<td>33</td>
<td>31</td>
<td>29</td>
<td>17</td>
</tr>
<tr>
<td>class3</td>
<td>33</td>
<td>33</td>
<td>41</td>
<td>34</td>
<td>27</td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>class1</td>
<td>8</td>
<td>8</td>
<td>22</td>
<td>18</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>class2</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>15</td>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>class3</td>
<td>35</td>
<td>33</td>
<td>36</td>
<td>33</td>
<td>30</td>
<td>16</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>class1</td>
<td>12</td>
<td>7</td>
<td>28</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>class2</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td>13</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>class3</td>
<td>24</td>
<td>22</td>
<td>31</td>
<td>24</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>class1</td>
<td>8</td>
<td>8</td>
<td>27</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>class2</td>
<td>0</td>
<td>0</td>
<td>28</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>class3</td>
<td>22</td>
<td>20</td>
<td>33</td>
<td>20</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>class1</td>
<td>8</td>
<td>7</td>
<td>22</td>
<td>21</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>class2</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>12</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>class3</td>
<td>17</td>
<td>16</td>
<td>30</td>
<td>15</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>class1</td>
<td>9</td>
<td>5</td>
<td>23</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>class2</td>
<td>0</td>
<td>0</td>
<td>23</td>
<td>11</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>class3</td>
<td>15</td>
<td>12</td>
<td>25</td>
<td>15</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>class1</td>
<td>5</td>
<td>3</td>
<td>24</td>
<td>18</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>class2</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>class3</td>
<td>5</td>
<td>9</td>
<td>26</td>
<td>13</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>class1</td>
<td>15</td>
<td>7</td>
<td>21</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>class2</td>
<td>0</td>
<td>0</td>
<td>31</td>
<td>24</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>class3</td>
<td>22</td>
<td>11</td>
<td>20</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>class1</td>
<td>8</td>
<td>4</td>
<td>27</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>class2</td>
<td>0</td>
<td>0</td>
<td>35</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>class3</td>
<td>12</td>
<td>11</td>
<td>27</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>class1</td>
<td>500</td>
<td>113</td>
<td>86</td>
<td>256</td>
<td>174</td>
<td>35</td>
</tr>
<tr>
<td>class2</td>
<td>500</td>
<td>11</td>
<td>8</td>
<td>297</td>
<td>178</td>
<td>113</td>
</tr>
<tr>
<td>class3</td>
<td>500</td>
<td>225</td>
<td>207</td>
<td>313</td>
<td>211</td>
<td>122</td>
</tr>
<tr>
<td>total</td>
<td>1500</td>
<td>349</td>
<td>301</td>
<td>866</td>
<td>563</td>
<td>270</td>
</tr>
</tbody>
</table>
Table 3: Average relative performance of the nine heuristics

<table>
<thead>
<tr>
<th>n</th>
<th>Proc. times</th>
<th>Constructive heuristics</th>
<th>Controlled insertion heuristics</th>
<th>Dominance based heuristics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Append1</td>
<td>Append2</td>
<td>Insert</td>
</tr>
<tr>
<td>10</td>
<td>class1</td>
<td>0.49</td>
<td>0.62</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>0.44</td>
<td>0.69</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.16</td>
<td>0.16</td>
<td>0.06</td>
</tr>
<tr>
<td>15</td>
<td>class1</td>
<td>0.70</td>
<td>0.89</td>
<td>0.20</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>0.75</td>
<td>1.07</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.02</td>
<td>0.14</td>
<td>0.07</td>
</tr>
<tr>
<td>20</td>
<td>class1</td>
<td>0.81</td>
<td>1.03</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>0.97</td>
<td>1.21</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.14</td>
<td>0.13</td>
<td>0.08</td>
</tr>
<tr>
<td>25</td>
<td>class1</td>
<td>0.78</td>
<td>1.01</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>1.07</td>
<td>1.49</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.15</td>
<td>0.16</td>
<td>0.08</td>
</tr>
<tr>
<td>30</td>
<td>class1</td>
<td>0.90</td>
<td>1.08</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>1.21</td>
<td>1.61</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.08</td>
<td>0.09</td>
<td>0.04</td>
</tr>
<tr>
<td>40</td>
<td>class1</td>
<td>0.95</td>
<td>1.17</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>1.28</td>
<td>1.60</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.13</td>
<td>0.18</td>
<td>0.06</td>
</tr>
<tr>
<td>50</td>
<td>class1</td>
<td>1.12</td>
<td>1.31</td>
<td>0.11</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>1.41</td>
<td>1.82</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.11</td>
<td>0.13</td>
<td>0.09</td>
</tr>
<tr>
<td>60</td>
<td>class1</td>
<td>0.89</td>
<td>1.09</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>1.55</td>
<td>1.90</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.12</td>
<td>0.12</td>
<td>0.05</td>
</tr>
<tr>
<td>70</td>
<td>class1</td>
<td>0.90</td>
<td>1.10</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>1.57</td>
<td>1.98</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.06</td>
<td>0.05</td>
<td>0.07</td>
</tr>
<tr>
<td>80</td>
<td>class1</td>
<td>0.87</td>
<td>1.06</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>1.49</td>
<td>1.86</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.09</td>
<td>0.09</td>
<td>0.05</td>
</tr>
<tr>
<td>ARP</td>
<td>class1</td>
<td>0.84</td>
<td>1.03</td>
<td>0.14</td>
</tr>
<tr>
<td>ARP</td>
<td>class2</td>
<td>1.17</td>
<td>1.52</td>
<td>0.03</td>
</tr>
<tr>
<td>ARP</td>
<td>class3</td>
<td>0.10</td>
<td>0.12</td>
<td>0.06</td>
</tr>
<tr>
<td>overall</td>
<td></td>
<td>0.70</td>
<td>0.89</td>
<td>0.08</td>
</tr>
</tbody>
</table>
Table 4: Computational requirements of the nine heuristics (in seconds)

<table>
<thead>
<tr>
<th>$n$</th>
<th>Proc. times</th>
<th>Constructive heuristics</th>
<th>Controlled insertion heuristics</th>
<th>Dominance based heuristics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Append1</td>
<td>Append2</td>
<td>Insert</td>
</tr>
<tr>
<td>10</td>
<td>class1</td>
<td>0.02</td>
<td>0.01</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>0.04</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.01</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>15</td>
<td>class1</td>
<td>0.09</td>
<td>0.05</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>0.24</td>
<td>0.06</td>
<td>0.11</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.02</td>
<td>0.06</td>
<td>0.09</td>
</tr>
<tr>
<td>20</td>
<td>class1</td>
<td>0.28</td>
<td>0.12</td>
<td>0.30</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>0.72</td>
<td>0.12</td>
<td>0.32</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.05</td>
<td>0.15</td>
<td>0.30</td>
</tr>
<tr>
<td>25</td>
<td>class1</td>
<td>0.60</td>
<td>0.24</td>
<td>0.70</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>1.74</td>
<td>0.24</td>
<td>0.76</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.10</td>
<td>0.30</td>
<td>0.68</td>
</tr>
<tr>
<td>30</td>
<td>class1</td>
<td>1.34</td>
<td>0.42</td>
<td>1.42</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>3.59</td>
<td>0.42</td>
<td>1.42</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.16</td>
<td>0.50</td>
<td>1.38</td>
</tr>
<tr>
<td>40</td>
<td>class1</td>
<td>4.90</td>
<td>1.00</td>
<td>4.36</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>11.48</td>
<td>1.04</td>
<td>4.46</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.46</td>
<td>1.16</td>
<td>4.28</td>
</tr>
<tr>
<td>50</td>
<td>class1</td>
<td>13.42</td>
<td>1.94</td>
<td>10.50</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>28.10</td>
<td>2.04</td>
<td>10.70</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>0.86</td>
<td>2.32</td>
<td>10.42</td>
</tr>
<tr>
<td>60</td>
<td>class1</td>
<td>23.48</td>
<td>3.38</td>
<td>22.40</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>59.02</td>
<td>3.52</td>
<td>22.40</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>1.86</td>
<td>4.08</td>
<td>21.60</td>
</tr>
<tr>
<td>70</td>
<td>class1</td>
<td>45.01</td>
<td>5.63</td>
<td>41.20</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>114.60</td>
<td>5.86</td>
<td>41.80</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>3.28</td>
<td>6.68</td>
<td>40.08</td>
</tr>
<tr>
<td>80</td>
<td>class1</td>
<td>83.12</td>
<td>8.55</td>
<td>71.40</td>
</tr>
<tr>
<td></td>
<td>class2</td>
<td>202.80</td>
<td>8.82</td>
<td>74.00</td>
</tr>
<tr>
<td></td>
<td>class3</td>
<td>4.28</td>
<td>10.22</td>
<td>68.75</td>
</tr>
</tbody>
</table>