8 Flow Shop Scheduling

Consider scheduling tasks on dedicated processors or machines. We assume that tasks belong to a set of n jobs, each of which is characterized by the same machine sequence. For convenience, let us assume that any two consecutive tasks of the same job are to be processed on different machines. The type of factory layout in the general case - handled in Chapter 10 - is the job shop; the particular case where each job is processed on a set of machines in the same order is the flow shop. The most commonly used performance measure will be makespan minimization.

8.1 Introduction

8.1.1 The Flow Shop Scheduling Problem

A flow shop consists of a set of different machines (processors) that perform tasks of jobs. All jobs have the same processing order through the machines, i.e. a job is composed of an ordered list of tasks where the i^{th} task of each job is determined by the same machine required and the processing time on it. Assume that the order of processing a set of jobs $\mathcal{J}$ on m different machines is described by the machine sequence $P_1, \cdots, P_m$. Thus job $J_j \in \mathcal{J}$ is composed of the tasks T_{1j}, $\cdots, T_{mj}$ with processing times p_{ij} for all machines P_i, $i = 1, \cdots, m$. There are several constraints on jobs and machines: (*i*) There are no precedence constraints among tasks of different jobs; (*ii*) each machine can handle only one job at a time; (*iii*) each job can be performed only on one machine at a time. While the machine sequence of all jobs is the same, the problem is to find the job sequences on the machines which minimize the makespan, i.e. the maximum of the completion times of all tasks. It is well known that - in case of practical like situations - the problem is NP-hard [GJS76].

Most of the literature on flow shop scheduling is limited to a particular case of flow shop - the *permutation flow shops*, in which each machine processes the jobs in the same order. Thus, in a permutation flow shop once the job sequence on the first machine is fixed it will be kept on all remaining machines. The resulting schedule will be called *permutation schedule*.

By a simple interchange argument we can easily see that there exists an optimal flow shop schedule with the same job order on the first two machines P_1 and P_2 as well as the same job order on the last two machines P_{m-1} and P_m. Con-

sider an optimal flow shop schedule. Among all job pairs with different processing orders on the first two machines, let J_i and J_k be two jobs such that the number of tasks scheduled between T_{1i} and T_{1k} is minimum. Suppose T_{1i} is processed before T_{1k} (while T_{2i} is processed after T_{2k}). Obviously, T_{1k} immediately follows T_{1i} and no other job is scheduled on machine P_1 in between. Hence, interchanging T_{1i} and T_{1k} has no effect on any of the remaining tasks' start times. Repetitious application of this interchange argument yields the same job order on the first two machine (and analogously for the last two machines). Consequently, any flow shop scheduling problem consisting of at most three machines has an optimal schedule which is a permutation schedule. This result cannot be extended any further as can be shown by a 2-job 4-machine example with $p_{11} = p_{41} = p_{22} = p_{32} = 4$ and $p_{21} = p_{31} = p_{12} = p_{42} = 1$. Both permutation schedules have a makespan of 14 while job orders (J_2, J_1) on P_1 and P_2 and (J_1, J_2) on P_3 and P_4 lead to a schedule with a makespan of 12. Although it is common practice to focus attention on permutation schedules, Potts et al. [PSW91] showed that this assumption can be costly in terms of the deviation of the maximum completion times, i.e. the makespans, of the optimal permutation schedule and the optimal flow shop schedule. They showed that there are instances for which the objective value of the optimal permutation schedule is much worse (in a factor more than $1/2\sqrt{m}$) than that of the optimal flow shop schedule.

Any job shop model (see Chapter 10) can be used to model the flow shop scheduling problem. We present a model basically proposed by Wagner [Wag59, Sta88] in order to describe the permutation flow shop. The following decision variables are used (for $i, j = 1, \cdots, n$; $k = 1, \cdots, m$):

$z_{ij} = \begin{cases} 1 & \text{if job } J_i \text{ is assigned to the } j^{\text{th}} \text{ position in the permutation,} \\ 0 & \text{otherwise;} \end{cases}$

x_{jk} = idle time (waiting time) on machine P_k before the start of the job in position j in the permutation of jobs;

y_{jk} = idle time (waiting time) of the job in the j^{th} position in the permutation, after finishing processing on machine P_k, while waiting for machine P_{k+1} to become free ;

C_{max} = makespan or maximum flow time of any job in the job set.

Hence we get the model:

Minimize C_{max}

subject to $\sum_{j=1}^{n} z_{ij} = 1, \qquad i = 1, \cdots, n \qquad (8.1.1)$

$\sum_{i=1}^{n} z_{ij} = 1, \qquad j = 1, \cdots, n \qquad (8.1.2)$

$$\sum_{i=1}^{n} p_{ri} z_{ij+1} + y_{j+1\,r} + x_{j+1\,r} = y_{jr} + \sum_{i=1}^{n} p_{r+1\,i} z_{ij} + x_{j+1\,r+1}, \quad (8.1.3)$$
$$j = 1,\cdots,n-1;\ r = 1,\cdots,m-1$$

$$\sum_{j=1}^{n}\sum_{i=1}^{n} p_{mi} z_{ij} + \sum_{j=1}^{n} x_{jm} = C_{max}, \quad (8.1.4)$$

$$\sum_{r=1}^{k-1}\sum_{i=1}^{n} p_{ri} z_{i1} = x_{1k}, \qquad k = 2,\cdots,m \quad (8.1.5)$$

$$y_{1k} = 0, \qquad k = 1,\cdots,m-1 \quad (8.1.6)$$

Equations (8.1.1) and (8.1.2) assign jobs and permutation positions to each other. Equations (8.1.3) provide Gantt chart accounting between all adjacent pairs of machines in the m-machine flow shop. Equation (8.1.4) determines the makespan. Equations (8.1.5) account for the machine idle time of the second and the following machines while they are waiting for the arrival of the first job. Equations (8.1.6) ensure that the first job in the permutation would always pass immediately to each successive machine.

8.1.2 Complexity

The minimum makespan problem of flow shop scheduling is a classical combinatorial optimization problem that has received considerable attention in the literature. Only a few particular cases are efficiently solvable, cf. [MRK83]:

(*i*) The two machine flow shop case is easy [Joh54]. In the same way the case of three machines is polynomially solvable under very restrictive requirements on the processing times of the intermediate machine [Bak74].

(*ii*) The two machine flow shop scheduling algorithm of Johnson can be applied to a case with three machines if the intermediate machine is no bottleneck, i.e. it can process any number of jobs at the same time, cf. [CMM67]. An easy consequence is that the two machine variant with time lags is solvable in polynomial time. That means for each job J_i there is a minimum time interval l_i between the completion of job J_i on the first machine P_1 and its starting time on the second machine P_2. The time lags can be viewed as processing times on an intermediate machine without limited capacity. Application of Johnson's algorithm to the problem with two machines P_1 and P_2, and processing times $p_{1i}+l_i$ and $p_{2i}+l_i$ on P_1 and P_2, respectively, yields an optimal schedule, cf. [Joh58, Mit58, MRK83].

(*iii*) Scheduling two jobs by the graphical method as described in [Bru88] and first introduced by Akers [Ake56]. (Actually this method also applies in the more general case of a job shop, cf. Chapter 10.)

(*iv*) Johnson's algorithm solves the preemptive two machine flow shop $F2 \mid pmtn \mid C_{max}$.

(*v*) If the definition of precedence constraints $J_i \prec J_j$ specifies that job J_i must complete its processing on each machine before job J_j may start processing on that machine then the two machine flow shop problem with tree or series-parallel precedence constraints and makespan minimization is solvable in polynomial time, cf. [Mon79, Sid79, MS79].

Slight modifications, even in the case of two machines, turn out to be difficult, see [TSS94]. For instance, $F3 \mid\mid C_{max}$ [GJS76], $F2 \mid r_j \mid C_{max}$ [LRKB77], $F2 \mid\mid L_{max}$ [LRKB77], $F2 \mid\mid \sum C_j$ [GJS76], $F2 \mid pmtn, r_j \mid C_{max}$ [CS81], $F2 \mid pmtn \mid L_{max}$ [TSS94], $F3 \mid pmtn \mid C_{max}$ [GS78], $F3 \mid pmtn \mid \sum C_j$ [LLR+93], $F2 \mid prec \mid C_{max}$ [Mon80], and $F2 \mid pmtn \mid \sum C_j$ [DL93] are strongly NP-hard.

8.2 Exact Methods

In this section we will be concerned with a couple of polynomially solvable cases of flow shop scheduling and continue to the most successful branch and bound algorithms. A survey on earlier approaches in order to schedule flow shops exactly can be found in [Bak75, KK88]. Dudek et al. [DPS92] review flow shop sequencing research since 1954.

8.2.1 The Algorithms of Johnson and Akers

An early idea of Johnson [Joh54] turned out to influence the development of solution procedures substantially. Johnson's algorithm solves the $F2 \mid\mid C_{max}$ to optimality constructing an optimal permutation schedule through the following approach:

Algorithm 8.2.1 *Johnson's algorithm for* $F2 \mid\mid C_{max}$ [Joh54].

begin

Let $\mathcal{S}_1$ contain all jobs $J_i \in \mathcal{J}$ with $p_{1i} \leq p_{2i}$ in a sequence of non-decreasing order of their processing times p_{1i};

Let $\mathcal{S}_2$ contain the remaining jobs of $\mathcal{J}$ (not in $\mathcal{S}_1$) in a sequence of non-increasing order of their processing times p_{2i};

Schedule all jobs on both machines in order of the concatenation sequence $(\mathcal{S}_1, \mathcal{S}_2)$;

end;

As Johnson's algorithm is a sorting procedure its time complexity is $O(n\log n)$. The algorithm is based on the following sufficient optimality condition.

Theorem 8.2.2 [Joh54] *Consider a permutation of n jobs where job J_i precedes job J_j if* $\min\{ p_{1i}, p_{2j} \} \le \min\{ p_{2i}, p_{1j} \}$ *for* $1 \le i, j \le n$. *Then the induced permutation schedule is optimal for* $F2 \mid\mid C_{max}$.

Proof. Let π be a permutation defining a schedule of the flow shop problem with n jobs. We may assume $\pi = (1, 2, \cdots, n)$. Then, there is an $s \in \{1, 2, \cdots, n\}$ such that the makespan $C_{max}(\pi)$ of the schedule equals

$$\sum_{i=1}^{s} p_{1i} + \sum_{i=s}^{n} p_{2i} = \sum_{i=1}^{s} p_{1i} - \sum_{i=1}^{s-1} p_{2i} + \sum_{i=1}^{n} p_{2i} .$$

Hence minimization of the makespan $\min_{\pi}\{C_{max}(\pi)\}$ is equivalent to

$$\min_{\pi}\{ \max_{1 \le s \le n} \Delta_s(\pi)\} \text{ where } \Delta_s(\pi) = \sum_{i=1}^{s} p_{1i} - \sum_{i=1}^{s-1} p_{2i} .$$

Let π' be another permutation different from π in exactly two positions j and $j+1$, i.e. the jobs' order defined by π' is $J_1, J_2, \cdots, J_{j-1}, J_{j+1}, J_j, J_{j+2}, J_{j+3}, \cdots, J_n$. As $\Delta_s(\pi) = \Delta_s(\pi')$ for $s = 1, \cdots, j-1, j+2, \cdots, n$, we get, that $\max_{1 \le s \le n} \Delta_s(\pi) \le \max_{1 \le s \le n} \Delta_s(\pi')$ holds if $\max\{\Delta_j(\pi), \Delta_{j+1}(\pi)\} \le \max\{\Delta_j(\pi'), \Delta_{j+1}(\pi')\}$. The latter is equivalent to

$$\max\{ p_{1j}, p_{1j} - p_{2j} + p_{1j+1} \} \le \max\{ p_{1j+1}, p_{1j+1} - p_{2j+1} + p_{1j} \}$$

which is equivalent to

$$p_{1j} \le p_{1j+1} \text{ and } p_{1j} - p_{2j} + p_{1j+1} \le p_{1j+1}$$

or

$$p_{1j} \le p_{1j+1} - p_{2j+1} + p_{1j} \text{ and } p_{1j} - p_{2j} + p_{1j+1} \le p_{1j+1} - p_{2j+1} + p_{1j} .$$

Thus, if $p_{1j} \le \min\{p_{1j+1}, p_{2j}\}$ or $p_{2j+1} \le \min\{p_{1j+1}, p_{2j}\}$, or equivalently, if $\min\{p_{1j}, p_{2j+1}\} \le \min\{p_{1j+1}, p_{2j}\}$ then permutation π defines a schedule at least as good as π'.

Among all permutations defining an optimal schedule, assume π is a permutation satisfying J_i precedes J_j if $\min\{p_{1i}, p_{2j}\} \le \min\{p_{2i}, p_{1j}\}$, for any two jobs J_i and J_j where one is an immediate successor of the other in the schedule. It remains to verify transitivity, i.e. if $\min\{p_{1i}, p_{2j}\} \le \min\{p_{2i}, p_{1j}\}$ implies J_i precedes J_j and $\min\{p_{1j}, p_{2k}\} \le \min\{p_{2j}, p_{1k}\}$ implies J_j precedes J_k then $\min\{p_{1i}, p_{2k}\} \le \min\{p_{2i}, p_{1k}\}$ implies J_i precedes J_k in π. There are 16 different cases to distinguish according to the relative values of the four processing time pairs p_{1i},

p_{2j}; p_{2i}, p_{1j}; p_{1j}, p_{2k} and p_{2j}, p_{1k}. Twelve of the cases are easy to verify. The remaining four cases,

(1) $p_{1i} \geq p_{2j} \leq p_{1k}$ and $p_{2i} \leq p_{1j} \leq p_{2k}$;

(2) $p_{1i} \geq p_{2j} \leq p_{1k}$ and $p_{2i} \geq p_{1j} \leq p_{2k}$;

(3) $p_{1i} \geq p_{2j} \geq p_{1k}$ and $p_{2i} \leq p_{1j} \leq p_{2k}$; and

(4) $p_{1i} \geq p_{2j} \geq p_{1k}$ and $p_{2i} \geq p_{1j} \leq p_{2k}$

imply that J_i may precede J_j or J_j may precede J_i. Hence, there is an optimal schedule satisfying the condition of the theorem for any pair of jobs. Finally, observe that this schedule is uniquely defined in case of strict inequalities $\min\{p_{1i}, p_{2i+1}\} < \min\{p_{2i}, p_{1i+1}\}$ for all pairs i, $i+1$ in π. If $\min\{p_{1i}, p_{2i+1}\} = \min\{p_{2i}, p_{1i+1}\}$ for a pair i, $i+1$ in π then an interchange of J_i and J_{i+1} will not increase the makespan. This proves that the theorem describes a sufficient optimality condition. □

Johnson's algorithm can be used as a heuristic when $m > 2$. Then the set of machines is divided into two subsets each of which defines a pseudo-machine having a processing time equal to the processing time on the real machines assigned to that subset. Johnson's algorithm can be applied to this n-job 2-pseudo-machine problem to obtain a permutation schedule. The quality of the outcome heavily depends on the splitting of the set of jobs into two subsets. If $m = 3$ an optimal schedule can be found from the two groups $\{P_1, P_2\}$ and $\{P_2, P_3\}$ if $\max_i p_{2i} \leq \min_i p_{1i}$ or $\max_i p_{2i} \leq \min_i p_{3i}$. Thus, for the pseudo machines $\{P_1, P_2\}$ and $\{P_2, P_3\}$ the processing times are defined as $p_{\{P_1,P_2\},i} = p_{1i} + p_{2i}$ and $p_{\{P_2,P_3\},i} = p_{2i} + p_{3i}$.

The problem of scheduling only two jobs on an arbitrary number of machines can be solved in polynomial time using the graphical method proposed by [Bru88] and first introduced by Akers [Ake56].

Assume to process two jobs J_1 and J_2 (not necessarily in the same order) in an m-machine flow shop. The problem can be formulated as a shortest path problem in the plane with rectangular objects as obstacles. The processing times of the tasks of J_1 (J_2) on the machines are represented as intervals on the x-axis (y-axis) which are arranged in order (next to each other) in which the corresponding tasks are to be processed. An interval I_{i1} (I_{i2}) on the x-axis (y-axis) is associated to a machine P_i on which the job J_1 (J_2) is supposed to be processed. Let x_F (y_F) denote the sum of the processing times of job J_1 (J_2) on all machines. Let $F = (x_F, y_F)$ be that point in the plane with coordinates x_F and y_F. Any rectangular $I_{i1} \times I_{i2}$ defines an obstacle in the plane. A feasible schedule corresponds to a path from the origin $O = (0,0)$ to F avoiding passing through any obstacle. Such a path consists of a couple of segments parallel to one of the axis or diagonal in the plane. A segment parallel to the x-axis (y-axis) can be interpreted in such a way

that only job J_1 (J_2) is processed on a particular machine while J_2 (J_1) is waiting for that machine, because parallel segments are only required if the path from O to F touches the border of an obstacle. An obstacle defined by some machine P_i and forcing the path from O to F to continue in parallel to one of the axis implies an avoidance of a conflict among both jobs. Hence, an obstacle means to sequence both jobs with respect to P_i. Minimization of the makespan corresponds to finding a shortest path from O to F avoiding all obstacles. The problem can be reduced to the problem of finding an unrestricted shortest path in an appropriate network $G = (\mathcal{V}, \mathcal{E})$. The set of vertices consists of O, F and all north-west and south-east corners of all rectangles. Each vertex v (except F) has at most two outgoing edges. These edges are obtained as follows: We are going from the point in the plane corresponding to vertex v diagonally until we hit the border of an obstacle or the boundary of the rectangle defined by O and F. In the latter case F is a neighbor of v. The length d_{vF} of the edge connecting v and F equals the length of the projection of the diagonal part of the v and F connecting path plus the length of the parallel to one of the axis part of this path. In other words, if v is defined in the plane by the coordinates (x_v, y_v) then $d_{vF} = \max\{ x_F - x_v, y_F - y_v \}$. If we hit the border of an obstacle, we introduce two arcs connecting the north-west corner (say vertex u defined by coordinates (x_u, y_u)) and the south-east corner (say vertex w defined by coordinates (x_w, y_w)) to v. The length of the edge connecting v to u is $d_{vu} = \max\{ x_u - x_v, y_u - y_v \}$. Correspondingly the length of the edge connecting v and w is $d_{vw} = \max\{ x_w - x_v, y_w - y_v \}$. Thus an application of a shortest path algorithm yields the minimum makespan. In our special case the complexity of the algorithm reduces to $O(m \log m)$, cf. [Bru88].

8.2.2 Dominance and Branching Rules

One of the early branch and bound procedures used to find an optimal permutation schedule is described by Ignall and Schrage [IS65] and, independently by Lomnicki [Lom65]. Associated with each node of the search tree is a partial permutation π defining a partial permutation schedule S_π on a set of jobs. Let $\mathcal{J}_\pi$ be the set of jobs from the schedule S_π. A lower bound is calculated for any completion τ of the partial permutation π to a complete permutation $(\pi\tau)$. The lower bound is obtained by considering the work remaining on each machine. The number of branches departing from a search tree node (with a minimum lower bound) equals the number of jobs not in S_π, i.e. for each job J_i with $i \notin \pi$ a branch is considered extending the partial permutation π by one additional position to a new partial permutation (πi). Moreover extensions of the algorithm use some dominance rules under which certain completions of partial permutations π can be eliminated because there exists a schedule at least as good as π among the completions of another partial permutation π'. Let $C_k(\pi)$ denote the completion

time of the last job in S_π on machine P_k, i.e. $C_k(\pi)$ is the earliest time at which some job not in $\mathcal{J}_\pi$ could begin processing on machine P_k. Then π' dominates π if for any completion τ of π there exists a completion τ' of π' such that $C_m(\pi'\tau') \leq C_m(\pi\tau)$. An immediate consequence is the following transitive *dominance criterion.*

Theorem 8.2.3 [IS65] *If* $\mathcal{J}_\pi = \mathcal{J}_{\pi'}$ *and* $C_k(\pi') \leq C_k(\pi)$ *for* $k = 1, 2, \cdots, m$, *then* π' *dominates* π. □

There are other dominance criteria reported in [McM69] and [Szw71, Szw73, Szw78] violating transitivity. In general these dominance criteria consider sets $\mathcal{J}_\pi \neq \mathcal{J}$. We can formulate

Theorem 8.2.4 *If* $C_{k-1}(\pi ji) - C_{k-1}(\pi i) \leq C_k(\pi ji) - C_k(\pi i) \leq p_{kj}$ *for* $k = 2, \cdots, m$, *then* (πji) *dominates* (πi). □

8.2.3 Lower Bounds

Next we consider different types of lower bounds that apply in order to estimate the quality of all possible completions τ of partial permutations π to a complete permutation $(\pi\tau)$.

The amount of processing time yet required on the first machine is $\sum_{j\in\tau} p_{1j}$. Suppose that a particular job J_j will be the last one in the permutation schedule. Then after completion of job J_j on P_1 an interval of at least $\sum_{k=2}^{m} p_{kj}$ must elapse before the whole schedule can be completed. In the most favorable situation the last job will be the one which minimizes the latter sum. Hence a lower bound on the makespan is

$$LB_1 = C_1(\pi) + \sum_{i\in\tau} p_{1i} + \min_{j\in\tau}\{\sum_{k=2}^{m} p_{kj}\} .$$

Similarly we obtain lower bounds (with respect to the remaining machines)

$$LB_p = C_p(\pi) + \sum_{i\in\tau} p_{pi} + \min_{j\in\tau}\{\sum_{k=p+1}^{m} p_{kj}\} , \text{ for } p = 2, \cdots, m-1.$$

And on the last machine we get

$$LB_m = C_m(\pi) + \sum_{i\in\tau} p_{mi} .$$

The lower bound proposed by Ignall and Schrage is the maximum of these m bounds.

To illustrate the procedure let us consider a 4–job, 3-machine instance from [Bak74]. The processing times p_{ij} can be found in Table 8.2.1.

p_{ij}	J_1	J_2	J_3	J_4
P_1	3	11	7	10
P_2	4	1	9	12
P_3	10	5	13	2

Table 8.2.1 *Processing times of a 4-job, 3-machine instance.*

Initially the permutation π is empty and four branches are generated from the initial search tree node. Each branch defines the next (first) position 1, 2, 3, or 4 in π. The partial permutations π, the values $C_p(\pi)$, and the lower bounds LB_p, for $p = 1, 2, 3$, and the maximum LB of the lower bounds obtained throughout the search are given in Table 8.2.2.

π	$C_1(\pi)$	$C_2(\pi)$	$C_3(\pi)$	LB_1	LB_2	LB_3	LB
1	3	7	17	37	31	37	37
2	11	12	17	45	39	42	45
3	7	16	29	37	35	46	46
4	10	22	24	37	41	52	52
1, 2	14	15	22	45	38	37	45
1, 3	10	19	32	37	34	39	39
1, 4	13	25	27	37	40	45	45

Table 8.2.2 *Search tree nodes of the Ignall / Schrage* [IS65] *branch and bound.*

Two additional branches are generated from that node associated with permutation (1, 4). These branches immediately lead to feasible solutions (1, 3, 2, 4) and (1, 3, 4, 2) with makespans equal to 45 and 39, respectively. Hence, (1, 3, 4, 2) is a permutation defining an optimal schedule.

The calculation of lower bound can be strengthened in a number of ways. On each machine P_k, except the first one, there may occur some idle time of P_k between the completion of job J_i and the start of its immediate successor J_j. The idle time arises if J_j is not ready "in time" on the previous machine P_{k-1}, in other words $C_{k-1}(\pi j) > C_k(\pi)$. Thus we can improve the aforementioned bounds if we replace the earliest start time on P_r of the next job not in J_π by

$$\bar{C}_1(\pi) = C_1(\pi) \text{ and } \bar{C}_r(\pi) = \max_{k=1,2,\cdots,r} \{C_k(\pi) + \min_{j\in\pi} \{\sum_{q=k}^{r-1} p_{qj}\}\}, \text{ for } r = 2,\cdots,m.$$

Besides the above machine based bound another job based bound can be calculated as follows: Consider a partial permutation π and let τ be an extension to a complete schedule $S_{\pi\tau}$. For any job J_j with $j\in\tau$ we can calculate a lower bound on the makespan of $S_{\pi\tau}$ as $C_1(\pi) + \sum_{k=1}^{m} p_{kj} + \sum_{J_i\in\mathcal{J}_1} p_{1i} + \sum_{J_i\in\mathcal{J}_2} p_{mi}$ where $\mathcal{J}_1$ ($\mathcal{J}_2$) are the sets of jobs processed before (after) J_j in schedule $S_{\pi\tau}$, respectively. Since $\sum_{J_i\in\mathcal{J}_1} p_{1i}$ $+ \sum_{J_i\in\mathcal{J}_2} p_{mi} \geq \sum_{\substack{i\in\tau \\ i\neq j}} \min\{p_{1i}, p_{mi}\}$ we get the following lower bounds:

$$LB_{J_j} = \max_{j\in\tau}\Big\{ \max_{1\leq r\leq s\leq m} \{ C_r(\pi) + \sum_{q=r}^{s} p_{qj} + \sum_{\substack{i\in\tau \\ i\neq j}} \min\{p_{ri}, p_{si}\}\}\Big\}$$

Let us consider the computation of lower bounds within a more general framework which can be found in [LLRK78]. The makespan of an optimal solution of any sub-problem consisting of all jobs and a subset of the set of machines defines a lower bound on the makespan of the complete problem. In general these bounds are costly to compute (the problem is NP-hard if the number of machines is at least 3) except in the case of two machines where we can use Johnson's algorithm. Therefore let us restrict ourselves to the case of any two machines P_u and P_v. That means only P_u and P_v are of limited capacity and can process only one job at a time. P_u and P_v are said to be bottleneck machines, while the remaining machines $P_1,\cdots,P_{u-1}$, $P_{u+1},\cdots,P_{v-1}$, $P_{v+1},\cdots,P_m$, the non-bottleneck machines, are available with unlimited capacity. In particular, a non-bottleneck machine may process jobs simultaneously. Since the three (at most) sequences of non-bottleneck machines $P_{1u} = P_1,\cdots,P_{u-1}$; $P_{uv} = P_{u+1},\cdots,P_{v-1}$, and $P_{vm} = P_{v+1}$, $\cdots,P_m$ can be treated as one machine each (because we can process the jobs on the non-bottleneck machines without interruption), it follows that (in our lower bound computation) each partial permutation π defines a partial schedule for a problem with at most five machines P_{1u}, P_u, P_{uv}, P_v, P_{vm}, in that order. Of course, the jobs' processing times on P_{1u}, P_{uv}, and P_{vm} have still to be defined. We define for any job J_i the processing times

$$p_{1u\,i} = \max_{r=1,2,\cdots,u-1} \{ C_r(\pi) + \sum_{k=r+1}^{u-1} p_{ki} \}; \quad p_{uv\,i} = \sum_{k=u+1}^{v-1} p_{ki}; \quad p_{vm\,i} = \sum_{k=v+1}^{m} p_{ki};$$

processing times on bottleneck machines are unchanged. Thus, the processing times $p_{1u\,i}$, $p_{uv\,i}$, and $p_{vm\,i}$ are the earliest possible start time of processing of job J_i on machine P_u, the minimum time lag between completion time of J_i on P_u and start time of J_i on P_v, and a minimum remaining flow time of J_i after com-

pletion on machine P_v, respectively. If $u = v$ we have a problem of at most three machines with only one bottleneck machine. Note, we can drop any of the machines P_{1u}, P_{uv}, or P_{vm} from the (at most) five machine modified flow shop problem through the introduction of a lower bound r_{1u}, r_{uv} on the start time of the successor machine, or a lower bound r_{vm} on the finish time of the whole schedule, respectively. In that case $r_{1u} = \min_{i \in \pi}\{p_{1u\,i}\}$, $r_{uv} = \min_{i \in \pi}\{p_{uv\,i}\}$; $r_{vm} = \min_{i \in \pi}\{p_{vm\,i}\}$. If $u = 1$, $v = u + 1$, or $v = m$ we have $r_{1u} = C_1(\pi)$, $r_{uv} = 0$, or $r_{vm} = 0$, respectively. The makespan $LB_\pi(\alpha, \beta, \gamma, \delta, \varepsilon)$ of an optimal solution for each of the resulting problems defines a lower bound on the makespan of any completion τ to a permutation schedule ($\pi\tau$). Hereby α equals P_{1u} or r_{1u} reflecting the cases whether the start times on P_u are depending on the completion on a preceding machine P_{1u} or an approximation of them, respectively. In analogy we get $\gamma \in \{P_{uv}, r_{uv}\}$ and $\varepsilon \in \{P_{vm}, r_{vm}\}$. Parameters β and δ correspond to P_u and P_v, respectively.

Let us consider the bounds in detail (neglecting symmetric cases):

(1) $LB_\pi(r_{1u}, P_u, r_{um}) = r_{1u} + \sum_{\substack{i=1 \\ i \in \pi}}^{n} p_{ui} + r_{um}$.

(2) The computation of $LB_\pi(r_{1u}, P_u, P_{um})$ amounts to minimization of the maximum completion time on machine P_{um}. The completion time of J_i on machine P_{um} equals the sum of the completion time of J_i on machine P_u and the processing time $p_{um\,i}$. Hence, minimizing maximum completion time on machine P_{um} corresponds to minimizing maximum lateness on machine P_u if the due date of job J_i is defined to be $-p_{um\,i}$. This problem can be solved optimally using the earliest due date rule, i.e. ordering the jobs according to non-decreasing due dates. In our case this amounts to ordering the jobs according to non-increasing processing times $p_{um\,i}$. Adding the value r_{1u} to the value of an optimal solution of this one-machine problem with due dates yields the lower bound $LB_\pi(r_{1u}, P_u, P_{um})$.

(3) The bound $LB_\pi(P_{1u}, P_u, r_{um})$ leads to the solution of a one-machine problem with release date $p_{1u\,i}$ for each job J_i. Ordering the jobs according to non-decreasing processing time $p_{1u\,i}$ yields an optimal solution. Once again, adding r_{um} to the value of this optimal solution gives the lower bound $LB_\pi(P_{1u}, P_u, r_{um})$.

(4) The computation of $LB_\pi(P_{1u}, P_u, P_{um})$ corresponds to minimizing maximum lateness on P_u with respect to due dates $-p_{um\,i}$ and release dates $p_{1u\,i}$. The problem is NP-hard, cf. [LRKB77]. Anyway, the problem can be solved quickly if the number of jobs is reasonable, see the one-machine lower bound on the job shop scheduling problem described in Chapter 10.

(5) Computation of $LB_\pi(r_{1u}, P_u, r_{uv}, P_v, r_{um})$ leads to the solution of a flow shop scheduling problem on two machines P_u and P_v. The order of the jobs obtained from Johnson's algorithms will not be affected if P_v is unavailable until $C_v(\pi)$. Adding r_{1u} and r_{um} to the makespan of an optimal solution of this two machine flow shop scheduling problem yields the desired bound.

(6) Computation of $LB_\pi(r_{1u}, P_u, P_{uv}, P_v, r_{um})$ leads to the solution of a 3-machine flow shop problem with a non-bottleneck machine between P_u and P_v. The same procedure as described under (5) yields the desired bound. The only difference being that Johnson's algorithm is used in order to solve a 2-machine flow shop with processing times $p_{ui}+p_{uv\,i}$ and $p_{vi}+p_{uv\,i}$ for all $i \notin \pi$.

Computation of the remaining lower bounds require to solve NP-hard problems, cf. [LRKB77] and [LLRK78].

$LB_\pi(r_{1u}, P_u, P_{uv}, P_v, r_{um})$ and $LB_\pi(P_{1u}, P_u, P_{um})$ turned out to be the strongest lower bounds. Let us consider an example taken from [LLRK78]: Let $n = m = 3$; let $p_{11} = p_{12} = 1$, $p_{13} = 3$, $p_{21} = p_{22} = p_{23} = 3$, $p_{31} = 3$, $p_{32} = 1$, $p_{33} = 2$. We have $LB_\pi(P_{1u}, P_u, P_{um}) = 12$ and $LB_\pi(r_{1u}, P_u, P_{uv}, P_v, r_{um}) = 11$. If $p_{21} = p_{22} = p_{23} = 1$ and all other processing times are kept then $LB_\pi(P_{1u}, P_u, P_{um}) = 8$ and $LB_\pi(r_{1u}, P_u, P_{uv}, P_v, r_{um}) = 9$.

In order to determine the minimum effort to calculate each bound we refer the reader to [LLRK78].

8.3 Approximation Algorithms

8.3.1 Priority Rule and Local Search Based Heuristics

Noteworthy flow shop heuristics for the makespan criterion are those of Campbell et al. [CDS70] and Dannenbring [Dan77]. Both used Johnson's algorithm, the former to solve a series of two machine approximations to obtain a complete schedule. The second method locally improved this solution by switching adjacent jobs in the sequence. Dannenbring constructed an artificial two machine flow shop problem with processing times $\sum_{j=1}^{m}(m-j+1)p_{ji}$ on the first artificial machine and processing times $\sum_{j=1}^{m} jp_{ji}$ on the second artificial machine for each job J_i, $i = 1,\cdots,n$. The weights of the processing times are based on Palmer's [Pal65] 'slope index' in order to specify a job priority. Job priorities are chosen so that jobs with processing times that tend to increase from machine to machine will receive higher priority while jobs with processing times that tend to decrease from machine to machine will receive lower priority. Hence the slope index, i.e.

the priority to choose for the next job J_i is $s_i = \sum_{j=1}^{m}(m-2j+1)p_{ji}$ for $i = 1,\cdots,n$. Then a permutation schedule is constructed using the job ordering with respect to decreasing s_i. Hundal and Rajgopal [HR88] extended Palmer's heuristic by computing two other sets of slope indices which account for machine $(m+1)/2$ when m is odd. Two more schedules are produced and the best one is selected. The two sets of slope indices are $s_i = \sum_{j=1}^{m}(m-2j+2)p_{ji}$ and $s_i = \sum_{j=1}^{m}(m-2j)p_{ji}$ for $i = 1,\cdots,n$.

Campbell et al. [CDS70] essentially generate a set of $m-1$ two machine problems by splitting the m machines into two groups. Then Johnson's two machine algorithm is applied to find the $m-1$ schedules, followed by selecting the best one. The processing times for the reduced problems are defined as $p_{1ki} = \sum_{j=1}^{k} p_{ji}$ and $p_{2ki} = \sum_{j=m-k+1}^{m} p_{ji}$ for $i = 1,\cdots,n$, where p_{1ki} (p_{2ki}) represents the processing time for job J_i on the artificial first (second) machine in the k^{th} problem, $k = 1,\cdots,m-1$.

Gupta [Gup71] recognizes that Johnson's algorithm is in fact a sorting algorithm which assigns an index to each job and sorts the jobs in ascending order by these indices. He generalized the index function to handle also cases of more than three machines. The index of job J_i is defined as

$$s_i = \lambda / \min_{1\le j\le m-1}\{p_{ji} + p_{j+1\,i}\} \quad \text{for } i = 1,\cdots,n$$

where

$$\lambda = \begin{cases} 1 & \text{if } p_{ji} \le p_{1i}, \\ -1 & \text{otherwise.} \end{cases}$$

The idea of [HC91] is the heuristical minimization of gaps between successive jobs. They compute the differences $d_{kij} = p_{k+1\,i} - p_{kj}$ for $i, j = 1,\cdots,n$; $k = 1,\cdots,m-1$ and $i \neq j$. If job J_i precedes job J_j in the schedule, then the positive value d_{kij} implies that job J_j needs to wait on machine P_{k+1} at least d_{kij} units of time until job J_i finishes. A negative value of d_{kij} implies that there exist d_{kij} units of idle time between job J_i and job J_j on machine P_{k+1}. Ho and Chang define a certain factor to discount the negative values. This factor assigns higher values to the first machines and lower values to last ones in order to reduce accumulated positive gaps effectively. The discount factor is defined as follows:

$$\delta_{kij} = \begin{cases} \dfrac{0.9\,(m-k-1)}{m-2} + 0.1 & \text{if } d_{kij} < 0, \\ 1 & \text{otherwise} \end{cases} \qquad \begin{array}{l}(\text{for } i, j = 1,\cdots,n; \\ \text{and } k = 1,\cdots,m-1).\end{array}$$

Combining the d_{kij} and the discount factor, Ho and Chang define the overall revised gap:

$$d_{Rij} = \sum_{k=1}^{m-1} d_{kij}\,\delta_{kij}, \text{ for } i, j = 1, \cdots, n.$$

Let $J_{[i]}$ be the job in the i^{th} position of a permutation schedule defined by permutation π. Then the heuristic works as follows:

Algorithm 8.3.1 *Gap minimization heuristic* [HC91].
begin
Let S be a feasible solution (schedule);
Construct values d_{Rij} for $i, j = 1, \cdots, n$;
$a := 1$; $b := n$;
repeat
$S' := S$;
Let $d_{R[a][u]} = \max_{a<j<b} \{d_{R[a][j]}\}$;
Let $d_{R[v][b]} = \min_{a<j<b} \{d_{R[j][b]}\}$;
if $d_{R[a][u]} < 0$ **and** $d_{R[v][b]} > 0$ **and** $| d_{R[a][u]} | \leq | d_{R[v][b]} |$
then
begin
$a = a + 1$;
Swap the jobs in the positions a and u of S;
end;
if $d_{R[a][u]} < 0$ **and** $d_{R[v][b]} > 0$ **and** $| d_{R[a][u]} | > | d_{R[v][b]} |$
then
begin
$b = b - 1$;
Swap the jobs in the positions b and v of S;
end;
if $| d_{R[a][u]} | > | d_{R[v][b]} |$
then
begin
$a = a + 1$;
Swap the jobs in the positions a and u of S;
end;
if the makespan of S increased **then** $S = S'$;
until $b = a + 2$
end;

Simulation results show that the heuristic [HC91] improves the best heuristic (among the previous ones) in three performance measures, namely makespan, mean flow time and mean utilization of machines.

An initial solution can be obtained using the following fast insertion method proposed in [NEH83].

Algorithm 8.3.2 *Fast insertion* [NEH83].

```
begin
  Order the n jobs by decreasing sums of processing times on the machines;
  Use Aker's graphical method to minimize the makespan of the first two jobs
    on all machines;
      -- The schedule defines a partial permutation schedule for the whole problem.
  for i = 3 to n do
    Insert the i^th job of the sequence into each of the i possible positions in the
      partial permutation and keep the best one defining an increased partial
      permutation schedule;
end;
```

Widmer and Hertz [WH89] and Taillard [Tai90] solved the permutation flow shop scheduling problem using tabu search. Neighbors are defined mainly as in the traveling salesman problem by one of the following three neighborhoods:

(1) Exchange two adjacent jobs.

(2) Exchange the jobs placed at the i^{th} position and at the k^{th} position.

(3) Remove the job placed at the i^{th} position and put it at the k^{th} position.

Werner [Wer90] provides an improvement algorithm, called path search, and shows some similarities to tabu search and simulated annealing. The tabu search described in [NS96] resembles very much the authors' tabu search for job shop scheduling. Therefore we refer the reader to the presentation in the job shop chapter. There are other implementations based on the neighborhood search, for instance, the simulated annealing algorithm [OP89] or the genetic algorithm [Ree95] or the parallel genetic algorithm [SB92].

8.3.2 Worst-Case Analysis

As mentioned earlier the polynomially solvable flow shop cases with only two machines are frequently used to generate approximate schedules for those problems having a larger number of machines.

It is easy to see that for any active schedule (a schedule is active if no job can start its processing earlier without delaying any other job) the following relation holds between the makespan $C_{max}(S)$ of an active schedule and the makespan C^*_{max} of an optimal schedule:

$$C_{max}(S) / C^*_{max} \leq \max_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}} \{p_{ij}\} / \min_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}} \{p_{ij}\}$$

Gonzales and Sahni [GS78] showed that $C_{max}(S) / C^*_{max} \leq m$ which is tight. They also gave a heuristic H_1 based on $\lfloor m/2 \rfloor$ applications of Johnson's algorithm with $C_{max}(S) / C^*_{max} \leq \lceil m/2 \rceil$ where S is the schedule produced by H_1.

Other worst-case performance results can be found in [NS93].

In [Bar81] an approximation algorithm has been proposed whose absolute error does not depend on n and is proved to be

$$C_{max}(S) - C^*_{max} = 0.5\,(m-1)\,(3m-1) \max_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}} \{p_{ij}\} .$$

where S is the produced schedule.

Potts [Pot85] analyzed a couple of approximation algorithms for $F2 \mid r_j \mid C_{max}$. The best one, called RJ', based on a repeated application of a modification of Johnson's algorithm has an absolute performance ratio of $C_{max}(S) / C^*_{max} \leq 5/3$ where S is the schedule obtained through RJ'.

In the following we concentrate on the basic ideas of *machine aggregation heuristics* using pairs of machines as introduced by Gonzalez and Sahni [GS78] and Röck and Schmidt [RS83]. These concepts can be applied to a variety of other NP-hard problems with polynomially solvable two-machine cases (cf. Sections 5.1 and 12.1). They lead to worst case performance ratios of $\lceil m/2 \rceil$, and the derivation of most of the results may be based on the following more general lemma which can also be applied in cases of open shop problems modeled by unrelated parallel machines.

Lemma 8.3.3 [RS83] *Let S be a non-preemptive schedule of a set $\mathcal{T}$ of n tasks on $m \geq 3$ unrelated machines P_i, $i = 1, \cdots, m$. Consider the complete graph $(\mathcal{P}, \mathcal{E})$ of all pairs of machines, where $\mathcal{E} = \{\{P_i, P_j\} \mid i, j = 1, \cdots, m$, and $i \neq j\}$. Let $\mathcal{M}$ be a maximum matching for $(\mathcal{P}, \mathcal{E})$. Then there exists a schedule S' where*

(1) *each task is processed on the same machine as in S, and S' has at most n preemptions,*

(2) *all ready times, precedence and resource constraints under which S was feasible remain satisfied,*

(3) *no pair $\{P_i, P_j\}$ of machines is active in parallel at any time unless $\{P_i, P_j\} \in \mathcal{M}$, and*

(4) *the finish time of each task increases by a factor of at most $\lceil m/2 \rceil$.*

Proof. In case of odd m add an idle dummy machine P_{m+1} and match it with the remaining unmatched machine, so that an even number of machines can be assumed. Decompose S into sub-schedules $S(q, f)$, $q \in \mathcal{M}$, $f \in \{f_q^1, f_q^2, \cdots, f_q^{K_q}\}$ where $f_q^1 < f_q^2 < \cdots < f_q^{K_q}$ is the sequence of distinct finishing times of the tasks which are processed on the machine pair q. Without loss of generality we assume

that $K_q \geq 1$. Let $f_q^0 = 0$ be the start time of the schedule and let $S(q, f_q^k)$ denote the sub-schedule of the machine pair q during interval $[f_q^{k-1}, f_q^k]$. The schedule S' which is obtained by arranging all these sub-schedules of S one after the other in the order of non-decreasing endpoints f, has the desired properties because (1): each task can preempt at most one other task, and this is the only source of preemption. (2) and (3): each sub-schedule $S(q, f)$ is feasible in itself, and its position in S' is according to non-decreasing endpoints of f. (4): the finish time C_j' of task $T_j \in \mathcal{T}$ in S' is located at the endpoint of the corresponding sub-schedule $S(q(j), C_j)$ where $q(j)$ is the machine pair on which T_j was processed in S, and C_j is the completion time of T_j in S. Due to the non-decreasing endpoint order of the sub-schedules it follows that $C_j' \leq \lceil m/2 \rceil C_j$. □

For certain special problem structures Lemma 8.3.3 can be specialized so that preemption is kept out. Then, the aggregation approach can be applied to problems $F \mid\mid C_{max}$ and $O \mid\mid C_{max}$, and to some of their variants which remain solvable in case of $m = 2$ machines. We assume that for flow shops the machines are numbered that reflects the order each job is assigned to the machines.

We present two aggregation heuristics that are based on special conditions restricting the use of machines.

Condition $C1$: No pair $\{P_i, P_j\}$ of machines is allowed to be active in parallel at any time unless $\{P_i, P_j\} \in \mathcal{M}_1 = \{\{P_{2l-1}, P_{2l}\} \mid l = 1, 2, \cdots, \lfloor m/2 \rfloor\}$.

Condition $C2$: Let $(\mathcal{P}, \mathcal{E})$ be a bipartite graph where $\mathcal{E} = \{\{P_a, P_b\} \mid a \in \{1, 2, \cdots, \lceil m/2 \rceil\}, b \in \{\lceil m/2 \rceil + 1, \cdots, m\}$, and let $\mathcal{M}_2$ be a maximal matching for $(\mathcal{P}, \mathcal{E})$. Then no pair $\{P_i, P_j\}$ of machines is allowed to be active in parallel at any time unless $\{P_i, P_j\} \in \mathcal{M}_2$.

The following Algorithms 8.3.4 and 8.3.5 are based on conditions $C1$ and $C2$, respectively.

Algorithm 8.3.4 *Aggregation heuristic* H_1 *for* $F \mid\mid C_{max}$ [GS78].

begin
for each pair $q_i = \{P_{2i-1}, P_{2i}\} \in \mathcal{M}_1$
  begin
  Find an optimal sub-schedule S_i^* for the two machines P_{2i-1} and P_{2i};
  if m is odd
  then
    $S_{\lceil m/2 \rceil}^*$:= an arbitrary schedule of the tasks on the remaining unmatched machine P_m;

$S := S_1^* \oplus S_2^* \oplus \cdots \oplus S_{\lceil m/2 \rceil}^*$;
end;
end;

As already mentioned, for $F \mid\mid C_{max}$ this heuristic was shown in [GS78] to have the worst case performance ratio of $C_{max}(H_1) / C_{max}^* \leq \lceil m/2 \rceil$. The given argument can be extended to $F \mid pmtn \mid C_{max}$ and $O \mid\mid C_{max}$, and also to some resource constrained models. Tightness examples which reach $\lceil m/2 \rceil$ can also be constructed, but heuristic H_1 is not applicable if permutation flow shop schedules are required.

In order to be able to handle this restriction consider the following Algorithm 8.3.5 which is based on condition $C2$. Assume for the moment that all machines with index less than or equal $\lceil m/2 \rceil$ are represented as a virtual machine P_1', and those with an index larger than $\lceil m/2 \rceil$ as a virtual machine P_1'. We again consider the given scheduling problem as a two machine problem.

Algorithm 8.3.5 *Aggregation heuristic H_2 for $F \mid\mid C_{max}$ and its permutation variant* [RS83].

begin
Solve the flow shop problem for two machines P_1', P_2' where each job J_j has processing time $a_j = \sum_{i=1}^{\lceil m/2 \rceil} p_{ij}$ on P_1' and processing time $b_j = \sum_{i=\lceil m/2 \rceil + 1}^{m} p_{ij}$ on P_2', respectively;
Let S be the two-machine schedule thus obtained;
Schedule the jobs on the given m machines according to the two machine schedule S;
end;

The worst case performance ratio of Algorithm 8.3.5 can be derived with the following Lemma 8.3.6.

Lemma 8.3.6 *For each problem $F \mid\mid C_{max}$ (permutation flow shops included) and $O \mid\mid C_{max}$, the application of H_2 guarantees $C_{max}(H_2) / C_{max}^* \leq \lceil m/2 \rceil$.*

Proof. Let S be an optimal schedule of length C_{max}^* for an instance of the problem under consideration. As $\mathcal{M}_2$ from condition $C2$ is less restrictive than $\mathcal{M}_1$, it follows from Lemma 8.3.3 that there exists a preemptive schedule S' which remains feasible under $C2$, and whose length is $C_{max}'/C_{max}^* \leq \lceil m/2 \rceil$. By construction of $\mathcal{M}_2$, S' can be interpreted as a preemptive schedule of the job set on the two virtual machines P_1', P_2', where P_1' does all processing which is required on the machines $P_1, \cdots, P_{\lceil m/2 \rceil}$, and P_2' does all processing which is required on the machines $P_{\lceil m/2 \rceil + 1}, \cdots, P_m$. Since on two machines preemptions are not advanta-

geous the schedule S generated by algorithm H_2 has length $C_{max}(H_2) \leq C'_{max} \leq \lceil m/2 \rceil C^*_{max}$. □

H_2 can be implemented to run in $O(n(m+\log n))$ for $F||C_{max}$ and also for its permutation variant using Algorithm 8.2.1. It is easy to adapt Lemma 8.3.3 to a given preemptive schedule S so that the $\lceil m/2 \rceil$ bound for H_2 extends to $F|pmtn|C_{max}$ as well.

The following example shows that the $\lceil m/2 \rceil$ bound of H_2 is tight for $F||C_{max}$. Take m jobs J_j, $j = 1, \cdots, m$, with processing times $p_{ij} = p > 0$ for $i = j$, whereas $p_{ij} = \varepsilon \to 0$ for $i \neq j$. H_2 uses the processing times $a_j = p + (\lceil m/2 \rceil - 1)\varepsilon$, $b_j = \lfloor m/2 \rfloor \varepsilon$ for $j \leq \lceil m/2 \rceil$, and $a_j = \lceil m/2 \rceil \varepsilon$, $b_j = p + (\lceil m/2 \rceil - 1)\varepsilon$ for $j > \lceil m/2 \rceil$. Consider job sets $\mathcal{J}^k$ which consist of k copies of each of these m jobs. For an optimal flow shop schedule for $\mathcal{J}^k$ we get $C^*_{max} = kp + (m-1)(k+1)\varepsilon$. The optimal two machine flow shop schedule for $\mathcal{J}^k$ produced by H_2 may start with all k copies of $J_{\lceil m/2 \rceil + 1}$, $J_{\lceil m/2 \rceil + 2}, \cdots, J_m$ and then continue with all k copies $J_1, J_2, \cdots, J_{\lceil m/2 \rceil}$. On m machines this results in a length of $C_{max}(H_2) = (m - 1 + k\lfloor m/2 \rfloor)\varepsilon + \lceil m/2 \rceil pk$. It follows that $C_{max}(H_2) / C^*_{max}$ approaches $\lceil m/2 \rceil$ as $\varepsilon \to 0$.

8.3.3 No Wait in Process

An interesting sub-case of flow shop scheduling is that with *no-wait constraints* where no intermediate storage is considered and a job once finished on one machine must immediately be started on the next one.

The two-machine case, i.e. problem $F2|no\text{-}wait|C_{max}$, may be formulated as a special case of scheduling jobs on one machine whose *state* is described by a single real valued variable x (the so-called *one state-variable machine problem*) [GG64, RR72]. Job J_i requires a starting state $x = A_i$ and leaves with $x = B_i$. There is a cost for changing the machine state x in order to enable the next job to start. The cost c_{ij} of J_j following J_i is given by

$$c_{ij} = \begin{cases} \int_{B_i}^{A_j} f(x)dx & \text{if } A_j \geq B_i, \\ \int_{A_j}^{B_i} f(x)dx & \text{if } B_i > A_j, \end{cases}$$

where $f(x)$ and $g(x)$ are integrable functions satisfying $f(x) + g(x) \geq 0$. The objective is to find a minimal cost sequence for the n jobs. Let us observe that problem

$F2 \mid no\text{-}wait \mid C_{max}$ may be modeled in the above way if $A_j = p_{1j}$, $B_j = p_{2j}$, $f(x) = 1$ and $g(x) = 0$. Cost c_{ij} then corresponds to the idle time on the second machine when J_j follows J_i, and hence a minimal cost sequence for the one state-variable machine problem also minimizes the completion time of the schedule for problem $F2 \mid no\text{-}wait \mid C_{max}$. On the other hand, the first problem corresponds also to a special case of the traveling salesman problem which can be solved in $O(n^2)$ time [GG64]. Unfortunately, more complicated assumptions concerning the structure of the flow shop problem result in its NP-hardness. So, for example, $Fm \mid no\text{-}wait \mid C_{max}$ is unary NP-hard for fixed $m \geq 3$ [Röc84].

As far as approximation algorithms are concerned H_1 is not applicable here, but H_2 turns out to work [RS83].

Lemma 8.3.7 *For $F \mid no\text{-}wait \mid C_{max}$, the application of H_2 guarantees $C_{max}(H_2) / C^*_{max} \leq \lceil m/2 \rceil$.*

Proof. It is easy to see that solving the two machine instance by H_2 is equivalent to solving the given instance of the m machine problem under the additional condition $C2$. It remains to show that for each no-wait schedule S of length C_{max} there is a corresponding schedule S' which is feasible under $C2$ and has length $C'_{max} \leq \lceil m/2 \rceil C_{max}$. Let $J_1, J_2, \cdots, J_n$ be the sequence in which the jobs are processed in S and let s_{ij} be the start time of job J_j, $j = 1, \cdots, n$, on machine J_i, $i = 1, \cdots, m$. As a consequence of the no-wait requirement, the successor J_{j+1} of J_j cannot start to be processed on machine P_{i-1} before J_j starts to be processed on machine P_i. Thus for $q = \lceil m/2 \rceil$ we have $s_j^{q+1} \leq s_{j+1}^{q} \leq \ldots \leq s_{j+q}^{1}$ and for the finish time C_j of job J_j we get $C_j \leq s_{j+1}^{m} \leq s_{j+2}^{m-1} \leq \ldots \leq s_{j+m-q}^{q+1} \leq s_{j+q}^{q+1}$. This shows that if we would remove the jobs between J_j and J_{j+q} from S, then S would satisfy $C2$ in the interval $[s_j^{q+1}, s_{j+q}^{q+1}]$. Hence, for each $k = 1, \ldots, q$ the sub-schedule S_k of S which covers only the jobs of the subsequence $J_k, J_{k+q}, J_{k+2q}, \cdots, J_{k+\lfloor (n-k)/q \rfloor q}$ of $J_1, \cdots, J_n$ satisfies $C2$. Arrange these sub-schedules in sequence. None is longer than C_{max}, and each job appears in one of them. The resulting schedule S' is feasible and has length $C'_{max} \leq qC_{max}$. □

Using the algorithm of Gilmore and Gomory [GG64], H_2 runs in $O(n(m + \log n))$ time. The tightness example given above applies to the no-wait flow shop as well, since the optimal schedule is in fact a no-wait schedule. Moreover, on two machines it is optimal to have any alternating sequence of jobs J_b, J_a, J_b, J_a with $a \in \{1, \ldots, \lceil m/2 \rceil\}$ and $b \in \{\lceil m/2 \rceil + 1, \ldots, m\}$, and in case of odd m this may be followed by all copies of J_1. When ε tends to zero, the length of such a schedule on m machines approaches $\lceil m/2 \rceil kp$, thus $C_{max}(H_2) / C^*_{max}$ approaches $\lceil m/2 \rceil$.

An interesting fact about the lengths of no-wait and normal flow shop schedules, respectively, has been proved by Lenstra. It appears that the no-wait constraint may lengthen the optimal flow shop schedule considerably, since $C^*_{max}(no\text{-}wait) / C^*_{max} < m$ for $m \geq 2$.

8.4 Scheduling Flexible Flow Shops

8.4.1 Problem Formulation

The hybrid or flexible flowshop problem is a generalization of the flowshop in such a way that every job can be processed by one among several machines on each machine stage. In recent years a number of effective exact methods have been developed. A major reason for this progress is the development of new job and machine based lower bounds as well as the rapidly increasing importance of constraint programming.

We consider the problem of scheduling n parts or jobs J_j, $j = 1, 2, \cdots, n$, through a manufacturing system that will be called a *flexible flow shop* (*FFS*), to minimize the schedule length. An FFS consists of $m \geq 2$ machine stages or centers with stage l having $k_l \geq 1$ identical parallel machines $P_{l1}, P_{l2}, \cdots, P_{lk_l}$ (see Figure 8.4.1). For job J_j vector $[p_{1j}, p_{2j}, \cdots, p_{mj}]^T$ of processing times is known, where $p_{lj} \geq 0$ for all l, j. Task T_{lj} of job J_j may be processed on any of the k_l machines. This is the generalization of the standard flow shop scheduling problem, whereas all the remaining assumptions remain unchanged.

The jobs have to visit the stages in the same order starting from stage 1 through stage m. 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 scheduling problem consists of assigning jobs to machines at each stage and sequencing the jobs assigned to the same machine so that some optimality criterion C is minimized.

Note that the processing time p_{lj} does not depend on the machine assigned to job J_j at stage l. This notation is applied when stage l consists of identical parallel machines. The completion time (which is a decision variable) of job J_j at stage l will be denoted by $C_j^{(l)}$.

A *partial schedule* S assigns some jobs to machines and fixes the processing order of another subset of jobs. S can be modeled by a directed graph $G = (V, A)$, where V consists of $nm+2$ nodes, i.e., one node (j, l) for each job J_j at each stage l and two additional nodes, 0 and *. A contains the arcs (directed edges) $(0,(j, l))$ and $((j, l),*)$ for all nodes (j, l). Moreover, the arcs $((j, l), (j, l-1))$ belong to A for all j and $1 \leq l \leq m-1$. Finally, whenever S fixes that job J_i precedes job J_j

at some stage l then arc $((i,l),(j,l))$ belongs to A. The *length of an arc* $((i,l),x) \in A$ is p_{li}, where x is a node of G. The length of any arc $(0,(i,l)) \in A$ is null. A *path* ρ in G is a sequence of nodes $(\rho_1, \ldots, \rho_e)$ such that ρ contains no node twice and $(\rho_u, \rho_{u+1}) \in A$ for all $1 \le u \le e-1$. The *length of a path* ρ is the sum of the lengths of the arcs (ρ_u, ρ_{u+1}), $1 \le u \le e-1$, along the path. Let $h(x,y)$ represent the length of the longest path between nodes x and y. If no path exists between x and y in G, then $h(x,y) = \infty$. Finally, the *release date* or *head* $r_j^{(l)}$ of job J_j at stage l is $h(0,(j,l))$, while its *delivery time* or *tail* $q_j^{(l)}$ is $h((j,l),*) - p_{lj}$, see Blazewicz et al. [BDP96]. Figure 8.4.1 is an example with m stages and k_l machines at each stage l.

Machines may remain idle and in-process inventory is allowed. This is important, since a restricted version of the problem was studied already by Salvador [Sal73] who presented a branch and bound algorithm for FFS with no-wait schedules and $p_{lj} > 0$ for all l, j. He identified the problem in the polymerization process where there are several effectively identical and thus interchangeable plants each of which can be considered as a flow shop. Of course, all situations where a parallel machine(s) is (are) added at least one stage of a flow shop to solve a bottleneck problem or to increase the production capacity lead to the FFS scheduling. Another interesting application of the problem was described by Brah and Hunsucker [BH91] and concerns the running of a program on a computer where the three steps of compiling, linking and running are performed in a fixed sequence and we have several processors (software) at each step. Other real life examples exist in the electronics manufacturing.

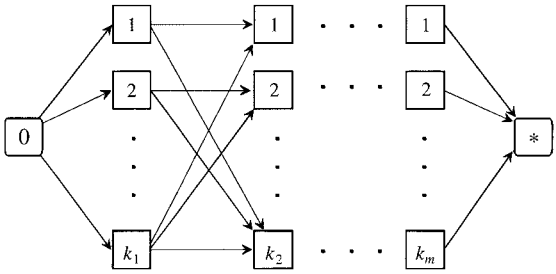

Figure 8.4.1 *Schematic representation of a flexible flow shop.*

Heuristics for the general FFS scheduling problem (in the sense stated above) were developed by Wittrock [Wit85, Wit88], and by Kochbar and Morris [KM87]. The first paper deals with a periodic algorithm where a small set of jobs is scheduled and the schedule is repeated many times, whereas the second one presents a non-periodic algorithm. The basic approach in both cases is to decompose the problem into three sub-problems: machine allocation, sequencing and

timing. The first sub-problem is to determine which jobs will visit which machine at each stage. The second sub-problem sequences jobs on each machine, and the third one consists of finding the times at which the jobs should enter the system. The heuristic algorithm developed by Kochbar and Morris considers setup times, finite buffers, blocking and starvation, machine down time, and current and subsequent state of the system. The heuristics tend to minimize the effect of setup times and blocking.

The standard $\alpha|\beta|\gamma$ notation for classifying scheduling problems by Graham et al. [GLL+79] has been extended by Vignier et al. [VBP99] to take the new machine environment into consideration. Here we will consider only models with identical parallel machines at the stages and the objective is to minimize the makespan, denoted by $Fm|k_1,k_2,\cdots,k_m|C_{max}$, and the mean flow time, $Fm|k_1, k_2,\cdots,k_m|\sum C_i$, respectively. In fact, we are not aware of efficient exact solution procedures for the general m-stage problem with other processing environments. By "general m-stage problem" we mean that m is not restricted to a small constant.

The general m-stage multiprocessor flowshop scheduling problem is strongly NP-hard for all traditional optimality criteria, since the special cases $F3||C_{max}$ and $F2||\sum C_i$ having only one machine at each stage are NP-hard in the strong sense, as shown in Garey et al. [GJS76]. Moreover, the makespan minimization problem is already NP-hard in the strong sense when $m = 2$ and $\max\{k_1,k_2\} > 1$ as shown by Gupta [Gup88]. Note that Hoogeveen et al. [HLV96] have proven that $F2|2,1|C_{max}$ is at least NP-hard in the ordinary sense, while its preemptive version, $F2|2,1, pmtn|C_{max}$, has been shown NP-hard in the strong sense.

In the following sections we will present some heuristics for simple subproblems of our problem for which the worst and average case performance is known.

Then we provide a comprehensive and uniform overview on exact solution methods for flexible flowshops with branching, bounding and propagation of constraints under two different objective functions: minimizing the makespan of a schedule and the mean flow time. This part is based on Kis and Pesch [KP05].

We do not discuss the large body of work on the two-stage special case. The review by Vignier et al. [VBP99] offers an exhaustive overview on two-stage problems.

We present a mixed integer-linear program modeling the constraints of both the minimum makespan and the minimum mean flow time problems, respectively. Then we consider the minimum makespan problem, followed by a discussion on approaches for minimizing the mean flow time. The latter two sections have a common structure: first various lower bounds are presented and compared if possible, then branching schemes and their merits are discussed.

8.4.2 Heuristics and Their Performance

The results presented in this section were obtained by Sriskandarajah and Sethi [SS89]. In the sequel the FFS scheduling problem with m machine stages and k_i machines at stage i will be denoted by $Fm \mid k_1, k_2, \cdots, k_m \mid C_{max}$.

Let us start with the problem $F2 \mid k_1 = 1,\ k_2 = k \geq 2 \mid C_{max}$, and let us assume that the buffer between the machine stages has unlimited capacity. First, consider the list scheduling algorithm in which a list of the job indices $1, 2, \cdots, n$ is given. Jobs enter the first machine stage (i.e. machine P_1) in the order defined by the list, form the queue between the stages and are processed in center 2, whenever a machine at this stage becomes available. C_{max} denotes the schedule length of the set of jobs when the list scheduling algorithm is applied, and C^*_{max} is the minimum possible schedule length of this set of jobs. Then the following theorem holds.

Theorem 8.4.1 [SS89] *For the list scheduling algorithm applied to the* problem $Fm \mid k_{m-1} = 1,\ k_m = k \geq 2 \mid C_{max}$ *we have*

$$C_{max}/C^*_{max} \leq m + 1 - \frac{1}{k},$$

and this is the best possible bound. □

The proof of this theorem is based on Grahams result [Gra66] for algorithms applied to the problem $Pm \mid\mid C_{max}$ (or $F1 \mid k_1 = k \geq 2 \mid C_{max}$). The bound is, as we remember from Section 5.1, $C_{max}/C^*_{max} \leq 2 - 1/k$.

Consider now Johnson's algorithm which, as we remember, is optimal for problem $F2 \mid\mid C_{max}$. The following can be proved.

Theorem 8.4.2 [SS89] *For Johnson's algorithm applied to problems* $F2 \mid k_1 = 1,\ k_2 = k = 2 \mid C_{max}$ *and* $F2 \mid k_1 = 1,\ k_2 = k \geq 3 \mid C_{max}$ *with* $C_{max} \leq \sum_{j=1}^{n} p_{1j} + \max_j \{p_{2j}\}$ *the following holds:*

$$C_{max}/C^*_{max} \leq 2,$$

and this is the best possible bound. □

Theorem 8.4.3 [SS89] *For Johnson's algorithm applied to the problem* $F2 \mid k_1 = 1,\ k_2 = k \geq 3 \mid C_{max}$ *with* $C_{max} > \sum_{j=1}^{n} p_{1j} + \max_j \{p_{2j}\}$ *we have*

$$C_{max}/C^*_{max} \leq 1 + (2 - \frac{1}{k})(1 - \frac{1}{k}).$$ □

Notice that the bounds obtained in Theorems 8.4.2 and 8.4.3 are better than those of Theorem 8.4.1.

Let us now pass to the problem $F2 \mid k_1 = k_2 = k \geq 2 \mid C_{max}$. The basic algorithm is the following.

Algorithm 8.4.4 *Heuristic H_a for $F2 \mid k_1 = k_2 = k \geq 2 \mid C_{max}$* [SS89].

begin
Partition the set of machines into k pairs $\{P_{11}, P_{21}\}, \{P_{12}, P_{22}\}, \cdots, \{P_{1k}, P_{2k}\}$, treating each pair as an artificial machine P_i', $i = 1, 2, \cdots, k$, respectively;
for each job $J_j \in \mathcal{J}$ **do** $p_j' := p_{1j} + p_{2j}$;
call List scheduling algorithm;
-- this problem is equivalent to the NP-hard problem $Pk \mid\mid C_{max}$ (see Section 5.1),
-- where a set of jobs with processing times p_j' is scheduled non-preemptively on a set
-- of k artificial machines; list scheduling algorithm solves this problem heuristically
for $i = 1$ **to** k **do call** Algorithm 8.2.1;
-- this loop solves optimally each of the k flow shop problems
-- with unlimited buffers, i.e. for each artificial machine P_i' the processing times p_i'
-- assigned to it are distributed among the two respective machines P_{1i} and P_{2i}
end;

Let us note, that in the last **for** loop one could also use the Gilmore-Gomory algorithm, thus solving the k flow shop problems with the no-wait condition. The results obtained from hereon hold also for the FFS with no-wait restriction, i.e. for the case of no buffer between the machine stages. On the basis of the Graham's reasoning, in [SS89] the same bound as in Theorem 8.4.1 has been proved for H_a, and this bound remains unchanged even if a heuristic list scheduling algorithm is used in the last **for** loop. Since an arbitrary list scheduling algorithm has the major influence on the worst case behavior of Algorithm H_a, in [SS89] another Algorithm, H_b, was proposed in which the *LPT* algorithm is used. We know from Section 5.1 that in the worst case *LPT* is better than an arbitrary list scheduling algorithm for $Pm \mid\mid C_{max}$. Thus, one can expect that for H_b a better bound exists than for H_a.

The exact bound R_{H_b} for H_b is not yet known, but Srishkandarajah and Sethi proved the following inequality,

$$\frac{7}{3} - \frac{2}{3k} \leq R_{H_b} \leq 3 - \frac{1}{k}.$$

The same authors proved that if *LPT* in H_b is replaced by a better heuristic or even by an exact algorithm, the bound would still be $R_{H_b} \geq 2$. The bound 2 has also been obtained in [Lan87] for a heuristic which schedules jobs in non-

increasing order of p_{2j} in FFS with $m = 2$ and an unlimited buffer between the stages.

Computational experiments performed in [SS89] show that the average performance of the algorithms presented above is much better than their worst case behavior. However, further efforts are needed to construct heuristics with better bounds (i.e. less than 2).

8.4.3 A Model

A mixed integer programming formulation for $Fm \mid k_1, k_2, \cdots, k_m \mid C_{max}$ is given by Guinet et al. [GSKD96]. The decision variables specify the order of jobs on the machines and the completion times of the jobs at each stage:

$x_{ijkl} = 1$, if job J_j is processed directly after job J_i on machine P_k in stage l,
0 otherwise,

$x_{0ikl} = 1$, if job J_i is the first job on machine P_k at stage l,
0 otherwise,

$x_{i0kl} = 1$, if job J_i is the last job on machine P_k at stage l,
0 otherwise,

$C_j^{(l)}$ = completion time of job J_j at stage l,

C_{max} = completion time of all jobs.

The mixed integer programming formulation in [GSKD96] is as follows:

minimize C_{max} (8.4.1)

subject to

$$\sum_{i=0, i\neq j}^{n} \sum_{k=1}^{k_l} x_{ijkl} = 1 \qquad \forall\ j = 1, \cdots, n,\ l = 1, \cdots, m \tag{8.4.2}$$

$$\sum_{j=0}^{n} x_{ijkl} \leq 1 \qquad \forall\ h = 0, \cdots, n,\ k = 1, \cdots, k_l,\ l = 1, \cdots, L \tag{8.4.3}$$

$$\sum_{i=0, i\neq h}^{n} x_{ihkl} - \sum_{j=0, j\neq h}^{n} x_{hjkl} = 0 \qquad \forall\ h = 1, \cdots, n,\ k = 1, \cdots, k_l,\ l = 1, \cdots, L \tag{8.4.4}$$

$$C_i^{(l)} + \sum_{k=1}^{k_l} x_{ijkl} \cdot p_{lj} + \Big((\sum_{k=1}^{k_l} x_{ijkl}) - 1 \Big) B \leq C_j^{(l)} \qquad \forall\ i = 1, \cdots, n,\ j = 1, \cdots, n,\ l = 1, \cdots, m \tag{8.4.5}$$

$$C_j^{(l-1)} + p_{lj} \leq C_j^{(l)} \qquad \forall\ j = 1, \cdots, n,\ l = 2, \cdots, m \tag{8.4.6}$$

$$C_j^{(l)} \leq C_{max} \qquad \forall\ j = 1,\cdots,n,\ l = 1,\cdots,m \tag{8.4.7}$$

$$x_{ijkl} \in \{0,1\} \qquad \forall\ i = 0,\cdots,n, j = 0,\cdots,n, \\ k = 1,\cdots,k_l,\ l = 1,\cdots,m \tag{8.4.8}$$

$$C_j^{(l)} \geq 0 \qquad \forall\ j = 1,\cdots,n,\ l = 1,\cdots,m \tag{8.4.9}$$

In this program B is a very big constant, i.e., greater than the sum of all job processing times.

The makespan minimization aspect of the problem is expressed by (8.4.1). Constraints (8.4.2), (8.4.3) and (8.4.4) ensure that each job is processed precisely once at each stage. In particular, (8.4.2) guarantees that at each stage l for each job J_j there is a unique machine such that either J_j is processed first or after another job on that machine. The inequalities (8.4.3) imply that at each stage there is a machine on which a job has a successor or is processed last. Finally, at each stage for each job there is one and only one machine satisfying both of the previous two conditions by (8.4.4). Constraints (8.4.5) and (8.4.6) take care of the completion times of the jobs. Inequalities (8.4.5) ensure that the completion times $C_i^{(l)}$ and $C_j^{(l)}$ of jobs J_i and J_j scheduled consecutively on the same machine respect this order. On the other hand, inequalities (8.4.6) imply that jobs go through the stages in the right order, i.e. from stage 1 through stage m. The constraint that the makespan is not smaller than the completion time of any job is expressed by (8.4.7). The last two constraints specify the domains of the decision variables.

To minimize the mean flow time instead of the makespan it is enough to replace the objective function (8.4.1) with the following one:

$$\min \sum_{i=1}^{n} C_i^{(m)} \tag{8.4.10}$$

Moreover, the variable C_{max} and all constraints involving it can be dropped.

8.4.4 The Makespan Minimization Problem

First we discuss various techniques for obtaining lower bounds, then we present branching schemes and also implementations and computational results.

Lower Bounds

Although we are concerned with the general m-stage problem, it is worth to recapitulate lower bounds for the two-stage special case, since several ideas stem from studying the latter problems. We will highlight the key ideas and cite papers that appear to propose them. If not mentioned otherwise, we assume that at each stage there are identical parallel machines.

I) *Reduction to classical flowshop.* Consider the classical flowshop scheduling problem obtained by dividing the job processing times at each stage by the number of machines. That is, define the new processing time of job J_j at stage l as $\tilde{p}_{lj} = p_{lj}/k_l$, $l = 1,\cdots,m, j = 1,\cdots,n$. The n new jobs with processing times $\tilde{p}_{lj}$ at the different stages constitute a classical flowshop scheduling problem. When $L = 2$, this flowshop scheduling problem can be solved to optimality by Johnson's rule [Joh54]. The optimum makespan C_{LB} of the latter problem is a lower bound on the optimum makespan of the original problem, as it is observed in Lee and Vairaktarakis [LV94]. To see this, let S^* be an optimal schedule for the two-stage multiprocessor problem and suppose the jobs are indexed in non-decreasing order of their completion time at stage 1, i.e., $i < j$ iff $C_i^{(1)} \le C_j^{(1)}$. Consider the first i jobs at stage 1 and the last $n-i+1$ jobs at stage 2. Since the last $n-i+1$ jobs cannot start earlier at the second stage than the completion of the first i jobs at the first stage, the completion time $C_{max}(S^*)$ satisfies

$$C_{max}(S^*) \ge \frac{1}{k_1}\sum_{j=1}^{i} p_{1j} + \frac{1}{k_2}\sum_{j=1}^{n} p_{2j}, \quad 1 \le i \le n .$$

Now consider the two-stage flowshop scheduling problem with n jobs having the above processing times. When sequencing the jobs at both stages in increasing order of their indices we obtain a feasible schedule and a longest path in that schedule with length $\bar{C}_{max}$. On this longest past there is a job h such that

$$\bar{C}_{max} = \sum_{j=1}^{h} \frac{p_{1j}}{k_1} + \sum_{j=h}^{n} \frac{p_{2j}}{k_2} .$$

We immediately see that $\bar{C}_{max} \le C_{max}(S^*)$. Moreover, as $C_{LB} \le \bar{C}_{max}$, the statement follows.

II) *Aggregation.* This is a very rich class of lower bounds based on computing the total amount of work on some stages or machines. Again, we begin with the case $m = 2$ and the following two lower bounds, $LB(1)$ and $LB(2)$, are enhancements of those suggested by Sriskandarajah and Sethi [SS89], generalizations of the bounds proposed by Gupta and Tunc [GT91] and Gupta [Gup88] and are reported in their present form by Guinet et al. [GSKD96].

$$LB(l) = \min_{i=1,\ldots,n} p_{3-l\,i}) + \max\left\{ (\sum_{i=1}^{n} C_i^{(m)})/k_l , \max_{i=1,\ldots,n} p_{mi} \right\}, \; l = 1,2 . \quad (8.4.11)$$

This bound is based on aggregating the work at stage l. Consider e.g., $LB(1)$. The processing of all jobs at the first stage cannot complete sooner than the max in (8.4.11). In addition to that, the last job, say job J_j, finished at this stage must be completed at the second stage too. The minimum amount of time spent by job j at the second stage is expressed by the min in (8.4.11). Hence, $LB(1)$ is a lower

bound on the makespan. By reversing the time the same argument shows that $LB(2)$ is a lower bound on the makespan as well.

Lee and Vairaktarakis introduced a different set of lower bounds for the $m = 2$ case in [LV94]. Suppose the jobs are indexed in non-decreasing order of stage 1 processing times, i.e., $p_{11} \leq \ldots \leq p_{1n}$. Let $P_{lq} = \sum_{j=1}^{q} p_{lj}$ denote the summation of the q shortest job processing times at stage l. If $k_1 \geq k_2$ then

$$LB_1 = \frac{P_{1k_2} + P_{2n}}{k_2} \tag{8.4.12}$$

is a lower bound on C_{max}. Namely, because of the flowshop constraints, on each machine at stage 2 there will be some idle time before processing may start, i.e., there will be a machine with idle time at least p_{11}, a machine with idle time p_{12}, $\cdots$, and a machine with idle time p_{1k_2}. Consequently, the makespan is no less than the average idle time plus the average workload at stage 2.

However, if $k_1 < k_2$ the above lower bound can be improved. Certainly, on each machine at stage 2 there will be idle time before processing starts and these idle times are at least $p_{11}, \cdots, p_{1k_2}$, respectively. Moreover, on $k_2 - k_1$ of these machines processing cannot start until at least two jobs are completed at stage 1. Hence, an additional idle time of at least p_{11} units is unavoidable on $k_2 - k_1$ machines at stage 2. Consequently, the following

$$LB_2 = \frac{P_{1k_2} + (k_2 - k_1)P_{11} + P_{2n}}{k_2} \tag{8.4.13}$$

is a lower bound on the makespan. By exploiting the symmetry of the two-stage multiprocessor flowshop problem we obtain another two bounds by interchanging the roles of stage 1 and stage 2. The new bounds will be

$$LB_3 = \frac{P_{2k_1} + (k_1 - k_2)P_{21} + P_{1n}}{k_1} \quad \text{if } k_1 \geq k_2, \tag{8.4.14}$$

$$LB_4 = \frac{P_{2k_1} + P_{1n}}{k_1} \quad \text{if } k_1 < k_2 . \tag{8.4.15}$$

These lower bounds can be combined to obtain the following lower bound:

$$LB = \begin{cases} \max\{LB_1, LB_3, C_{LB}\} & \text{if } k_1 \geq k_2 \\ \max\{LB_2, LB_4, C_{LB}\} & \text{if } k_1 < k_2 \end{cases}$$

where C_{LB} is the lower bound of Lee and Vairaktarakis obtained by reduction to classical flowshop.

Brah and Hunsucker proposed two bounds for the general m-stage problem, one based on machines and another based on jobs [BH91]. Suppose all jobs are sequenced on stages 1 through $l-1$ and a subset $\mathcal{A}$ of jobs is already scheduled at stage l. Before describing the two bounds, we introduce additional notation.

$\mathcal{J}$ = set of all jobs,
$\mathcal{A}$ = set of jobs already scheduled at stage l,
$S^{(l)}(\mathcal{A})$ = partial schedule of jobs in $\mathcal{A}$ at stage l,
$C[S^{(l)}(\mathcal{A})]_k$ = completion time of the partial sequence on machine k.

Notice that in order to compute $C[S^{(l)}(\mathcal{A})]_k$ we have to fix the schedule of the upstream stages.

Having fixed the schedule of all jobs on the first $l-1$ stages and that of the jobs in $\mathcal{A}$ at stage l, the average completion time of all jobs at stage l, $ACT[S^{(l)}(\mathcal{A})]$, can be computed as follows:

$$ACT[S^{(l)}(\mathcal{A})] = \frac{\sum_{k=1}^{k_l} C[S^{(l)}(\mathcal{A})]_k}{k_l} + \frac{\sum_{j \in \mathcal{J}-\mathcal{A}} p_{lj}}{k_l} \tag{8.4.16}$$

It is worth mentioning that in any complete schedule of all jobs at stage l that contains the partial schedule $S^{(l)}(\mathcal{A})$, there will be a job completing not sooner than $ACT[S^{(l)}(\mathcal{A})]$.

The maximum completion time of jobs in $\mathcal{A}$ at stage l, $MCT[S^{(l)}(\mathcal{A})]$, is given by

$$MCT[S^{(l)}(\mathcal{A})] = \max_{1 \le k \le k_l} C[S^{(l)}(\mathcal{A})]_k . \tag{8.4.17}$$

The machine based lower bound, LBM, is given by

$$LBM[S^{(l)}(\mathcal{A})] = \begin{cases} ACT[S^{(l)}(\mathcal{A})] + \min\limits_{i \in \mathcal{J}-\mathcal{A}} \left\{ \sum\limits_{l'=l+1}^{m} p_{l'i} \right\} \\ \quad \text{if } ACT[S^{(l)}(\mathcal{A})] \ge MCT[S^{(l)}(\mathcal{A})] , \\ MCT[S^{(l)}(\mathcal{A})] + \min\limits_{i \in \mathcal{A}} \left\{ \sum\limits_{l'=l+1}^{m} p_{l'i} \right\} \\ \quad \text{otherwise} \end{cases} \tag{8.4.18}$$

The rationale behind separating the two cases stems from the following observation. If $ACT[S^{(l)}(\mathcal{A})] \ge MCT[S^{(l)}(\mathcal{A})]$ then the last job finished at stage l will be a job in $\mathcal{J}-\mathcal{A}$. If $ACT[S^{(l)}(\mathcal{A})] < MCT[S^{(l)}(\mathcal{A})]$ then the last job scheduled at stage l may come from $\mathcal{A}$ or from $\mathcal{J}-\mathcal{A}$.

The job based lower bound, LBJ, is defined by

$$LBJ[S^{(l)}(\mathcal{A})] = \min_{1\le k\le k_l} \{ C[S^{(l)}(\mathcal{A})]_k \} + \max_{i\in\mathcal{J}-\mathcal{A}} \Big\{ \sum_{l'=l}^{m} p_{l'i} \Big\} . \tag{8.4.19}$$

Finally, the composite lower bound, *LBC*, is given by

$$LBC[S^{(l)}(\mathcal{A})] = \max\{ LBM[S^{(l)}(\mathcal{A})], LBJ[S^{(l)}(\mathcal{A})] \} . \tag{8.4.20}$$

The *LBM* bound (8.4.18) is improved in Portmann et al. [PVDD98]. Namely, if $ACT[S^{(l)}(\mathcal{A})] = MCT[S^{(l)}(\mathcal{A})]$ and $\mathcal{J}-\mathcal{A} \neq \emptyset$ then it may happen that

$$\min_{i\in\mathcal{A}} \Big\{ \sum_{l'=l+1}^{m} p_{l'i} \Big\} > \min_{i\in\mathcal{J}-\mathcal{A}} \Big\{ \sum_{l'=l+1}^{m} p_{l'i} \Big\} \tag{8.4.21}$$

holds, for the processing times of the jobs in $\mathcal{A}$ and in $\mathcal{J}-\mathcal{A}$ are unrelated. In this case *LBM* can be improved by the difference of the left and right hand sides of (8.4.21). That is, if $\mathcal{J}-\mathcal{A} \neq \emptyset$, the improved lower bound becomes

$$LBM[S^{(l)}(\mathcal{A})] = \begin{cases} ACT[S^{(l)}(\mathcal{A})] + \min\limits_{i\in\mathcal{J}-\mathcal{A}} \Big\{ \sum\limits_{l'=l+1}^{m} p_{l'i} \Big\} \\ \qquad \text{if } ACT[S^{(l)}(\mathcal{A})] > MCT[S^{(l)}(\mathcal{A})] , \\ MCT[S^{(l)}(\mathcal{A})] + \min\limits_{i\in\mathcal{A}} \Big\{ \sum\limits_{l'=l+1}^{m} p_{l'i} \Big\} \\ \qquad \text{if } ACT[S^{(l)}(\mathcal{A})] < MCT[S^{(l)}(\mathcal{A})] , \\ ACT[S^{(l)}(\mathcal{A})] + \\ \qquad \max\Big\{ \min\limits_{i\in\mathcal{J}-\mathcal{A}} \Big\{ \sum\limits_{l'=l+1}^{m} p_{l'i} \Big\}, \min\limits_{i\in\mathcal{A}} \Big\{ \sum\limits_{l'=l+1}^{m} p_{l'i} \Big\} \Big\} \\ \qquad \text{if } ACT[S^{(l)}(\mathcal{A})] = MCT[S^{(l)}(\mathcal{A})] . \end{cases} \tag{8.4.22}$$

III) *Bounds with heads and tails.* The set of bounds in this category share the property that they can be computed for any stage l and it is not assumed that all jobs are completely scheduled on all upstream stages. This is in contrast with bounds (8.4.18), (8.4.19) and (8.4.22) that heavily rely on this assumption. Lower bounds based on heads and tails can easily be updated whenever a scheduling decision has been made either through branching in a branch and bound procedure or through propagation of constraints. While the basic idea of the bounds (8.4.18), (8.4.19) and (8.4.22) is calculation of average processing times or average machine in process times, the main idea of the subsequent bounds is the calculation and subsequent reduction of the domains of start times of the jobs at each stage, i.e. the interval limited by the earliest and latest possible start and completion times of the jobs, see [VHHL05].

To simplify notation, we fix stage l. Assume that a partial schedule S already exists (maybe S is empty). We define a set $\mathcal{B}$ of $\overline{n}$ tasks with processing times $\tilde{p}_j = p_{lj}$ for each $T_j \in \mathcal{B}$, noting that j refers also to job J_j of the multiprocessor

flowshop scheduling problem. In addition to that, a ready time $r_j = r_j^{(l)}$ and a delivery time $q_j = q_j^{(l)}$ are defined for each task T_j, where $r_j^{(l)}$ and $q_j^{(l)}$ are determined with respect to the graph representation of the partial solution S.

The problem of scheduling n tasks on $\bar{m} = k_l$ identical parallel machines subject to release dates and delivery times to minimize the makespan, $P\bar{m} \mid r_j, q_j \mid C_{max}$, is a relaxation of the multiprocessor flowshop scheduling problem, as it is pointed out by Carlier and Pinson [CP98]. Since this problem is NP-hard in the strong sense, as the one machine special case $1 \mid r_j, q_j \mid C_{max}$ already is [GJ77], various lower bounds are proposed in Carlier [Car87]. All of these lower bounds are lower bounds for the multiprocessor flowshop problem as well [CN00].

The most basic lower bound for the $P\bar{m} \mid r_j, q_j \mid C_{max}$ problem is

$$LB_1 = \max_{T_i \in \mathcal{B}} \{r_i + p_i + q_i\} \tag{8.4.23}$$

Now consider a subset $\mathcal{B}'$ of $\mathcal{B}$ and define the quantity

$$G(J) = \min_{T_j \in \mathcal{B}'} \{r_j\} + \frac{1}{\bar{m}} \Big(\sum_{T_j \in \mathcal{B}'} p_j \Big) + \min_{T_j \in \mathcal{B}'} \{q_j\}$$

Clearly, $G(\mathcal{B}')$ is a lower bound for the $P\bar{m} \mid r_j, q_j \mid C_{max}$ problem with respect to any $\mathcal{B}' \subseteq \mathcal{B}$. Consequently, taking the maximum over all subsets $\mathcal{B}'$ of $\mathcal{B}$ we obtain another lower bound:

$$LB_2 = \max_{T_j \in \mathcal{B}'} \{G(\mathcal{B}')\}, \tag{8.4.24}$$

which can be computed in $O(\bar{n} \cdot log\, \bar{n})$ time generating Jackson's preemptive schedule for the one-machine scheduling problem with heads $\bar{m} r_j$, processing times p_j and tails $\bar{m} q_j$, see Carlier [Car87]. The optimal value of a preemptive solution of the one machine problem is $\bar{m} LB_2$.

The next lower bound tries to take into account the heads and tails of different operations in a more efficient way. Namely, let $\mathcal{B}'$ be a subset of $\mathcal{B}$ with $|\mathcal{B}'| \geq \bar{m}$. Denote $r_{i_1}, \cdots, r_{i_{\bar{m}}}$ and $q_{j_1}, \cdots, q_{j_{\bar{m}}}$ the $\bar{m}$ smallest release times and delivery times, respectively, of jobs in $\mathcal{B}'$. Define the quantity $G'(\mathcal{B}')$ by

$$G'(\mathcal{B}') = \frac{1}{\bar{m}} \Big(\sum_{u=1}^{\bar{m}} r_{i_u} + \sum_{T_j \in \mathcal{B}'} p_j + \sum_{u=1}^{\bar{m}} q_{i_u} \Big) \tag{8.4.25}$$

It is shown in [Car87] that LB_3, as defined by (8.4.26) below, is a lower bound for the $Pm \mid r_j, q_j \mid C_{max}$ problem.

$$LB_3 = \max_{\mathcal{B}' \subseteq \mathcal{B}, |\mathcal{B}'| \geq \bar{m}} \{G'(\mathcal{B}')\} \tag{8.4.26}$$

In order to show this, we may assume that each machine is used from jobs of $\mathcal{B}'$.

A (every) machine is idle from time 0 to time r_{i_1}, a second machine is idle from time 0 to time r_{i_2} and the machine $\bar{m}$ is idle from 0 to $r_{i_{\bar{m}}}$. Similarly, one machine is idle after processing for q_{j_1} time units, a second machine is idle for q_{j_2} time units, etc. Adding processing and idle times for all machines it is obvious that (8.4.25) is a lower bound for any subset $\mathcal{B}'$ of $\mathcal{B}$.

Bound (8.4.26) can straightforwardly be computed in $O(\bar{n}^3)$ time [Van94], [Per95]. However, it is shown in [CP98] that the stronger lower bound

$$LB_4 = \max\{LB_1, LB_3\} \tag{8.4.27}$$

can be computed in $O(\bar{n}\cdot log\,\bar{n} + \bar{n}\bar{m}\cdot log\,\bar{m})$ time using Jackson's Pseudo Preemptive schedule. In such a schedule, an operation may be processed on more than one machine at a time. Moreover, it can be shown that the distance between the non-preemptive optimal makespan and LB_4 is at most $2p_{max}$ [Car87].

For the sake of completeness we mention that when schedule S is empty then $r_i^{(l)} = \sum_{l'=1}^{l-1} p_{l'i}$ and symmetrically $q_i^{(l)} = \sum_{l'=l+1}^{L} p_{l'i}$ hold for each job J_i. For this special case Santos et al. [SHD95] has proven that $G'(\mathcal{B}')$ (cf. equation (8.4.25)) is a lower bound when $\mathcal{B}'$ consists of all jobs. Computational results show that, on average, the lower bound is within 8% of the optimum.

An even stronger lower bound can be obtained by solving the preemptive version of the $P\bar{m}\,|\,r_j, q_j\,|\,C_{max}$ problem using a network flow model. Fix a makespan C and define deadlines $d_j = C - q_j$ for each job J_j. Job J_j must be processed in the interval $[r_j, d_j]$ in order to complete all jobs by time C. There are at most $h \le 2n$ different r_j and d_j values and let $v_1, \ldots, v_h$ represent these values arranged increasingly, i.e., $v_1 < v_2 < \ldots < v_h$. Let $I_t = [v_t, v_{t+1})$, $t = 1, \cdots, h-1$, represent $h-1$ intervals with lengths $l_1, \cdots, l_{h-1}$. If $I_t \subseteq [r_j, d_j]$ then a part $\min\{l_t, p_j\}$ of job J_j can be processed in interval I_t. Hence we form a capacitated network with $\bar{n} + (h-1) + 2$ nodes, having one source node s, one sink node r, $\bar{n}$ nodes for representing the jobs and $h-1$ nodes for representing the intervals. Source s is connected to each job node j with an arc of capacity p_j. Each job node j is connected to each interval I_t with $I_t \subseteq [r_j, d_j]$ using an arc of capacity $\min\{l_t, p_j\}$, and finally each interval I_t is connected to the sink by an arc of capacity $\bar{m}l_t$. In this network there is a flow of value $\sum_j p_j$ if and only if the preemptive $P\bar{m}\,|\,r_j, q_j, pmtn\,|\,C_{max}$ problem has a solution with makespan C. Using dichotomic search, the smallest C admitting a compatible flow of value $\sum_j p_j$ can be found in polynomial time. It is shown in Hoogeveen et al. [HHLV95] that the difference between the preemptive makespan and LB_4 is not more than $\bar{m}/(\bar{m}-1)p_{max}$. Nonetheless, this gap is claimed to vanish in practice [CP98]. The drawback of this method is the relatively high computation time for finding the maximum flow.

We close this section by a rather tricky lower bound of Carlier and Néron.

Let $R_1, \cdots, R_{\bar{m}}$ denote the $\bar{m}$ smallest increasingly ordered machine availability times at stage l, noting that they depend on the partial schedule S. Let

$$G'_{machine}(\mathcal{B}') = \frac{1}{\bar{m}}\left(\max(R_1, r_{i_1}) + \ldots + \max(R_{\bar{m}}, r_{i_{\bar{m}}}) + \sum_{J_j \in \mathcal{B}'} p_j + q_{i_1} + \ldots + q_{i_{\bar{m}}}\right). \tag{8.4.28}$$

Now, if $R_m + q_{j_m} < UB$, then $G'_{machine}(\mathcal{B}')$ is a lower bound on UB.

Let us briefly sketch the main ideas of the proof which can be found in [CN00]. Let S be a schedule with a makespan of at most UB. If there exists a machine P_h different from $P_{\bar{m}}$ without any job from $\mathcal{B}'$ to process then the jobs from $\mathcal{B}'$ scheduled on machine $P_{\bar{m}}$ can be scheduled on machine P_h. From $R_l \leq R_{\bar{m}}$ we know this will not increase the makespan of S. If no job from $\mathcal{B}'$ is processed on machine $P_{\bar{m}}$ consider the difference $\delta = UB - R_{\bar{m}} - q_{i_{\bar{m}}} > 0$. A part δ of job $J_{i_{\bar{m}}}$ (with release date $r_{i_{\bar{m}}}$ and tail $q_{i_{\bar{m}}}$) which is scheduled on some machine can be scheduled in the interval $[UB - q_{i_{\bar{m}}} - \delta, UB - q_{i_{\bar{m}}}]$ on machine $P_{\bar{m}}$ without increasing the makespan. Thus, we can conclude, if there is a schedule with a makespan of at most UB then there is also a (preemptive) schedule with a makespan of at most UB in which all machines have to process at least a part of a job from $\mathcal{B}'$.

What remains is to sum up idle times and processing times of all machines with respect to the (preemptive) schedule. The first machine is idle from 0 to R_1 but also from 0 to r_{i_1}. Therefore it is idle from 0 to $\max\{R_1, r_{i_1}\}$. There is also a machine idle from time $UB - q_{i_1}$ until time UB. Similar conclusions for the remaining machines yield the desired result.

Branch-and-Bound Methods

We have introduced several lower bounds in the previous section. Below we discuss branching schemes and search strategies.

The first branch-and-bound procedure for the $Fm \mid k_1, \cdots, k_m \mid C_{max}$ problem is proposed in Brah and Hunsucker [BH91].

This procedure is a modification of the method developed by Bratley et al. [BFR75] for scheduling on parallel machines. At each stage l two decisions must be made: the assignment of the jobs to a machine P_{li}, and the scheduling of jobs on every machine at stage l. The enumeration is accomplished by generating a tree with two types of nodes: node $\textcircled{j}$ denotes that job J_j is scheduled on the current machine, whereas node $\boxed{j}$ denotes that J_j is scheduled on a new machine, which now becomes the current machine. The number of $\square$ nodes on each branch is equal to the number of parallel machines used by that branch, and

thus must be less than or equal to k_l at stage l. The number of possible branches at each stage l was established by Brah in [Bra88] as

$$N(n, k_l) = \binom{n-1}{k_l - 1} \frac{n!}{k_l!}.$$

Consequently, the total number of possible end nodes is equal to

$$S(n, m, \{k_l\}_{l=1}^{m}) = \prod_{l=1}^{m} \binom{n-1}{k_l - 1} \frac{n!}{k_l!}.$$

For the construction of a tree for the problem, some definitions and rules at each stage l are useful. Let the level 0_l represent the root node at stage l, and 1_l, $2_l, \cdots, z_l$ represent different levels of the stage, with z_l being the terminal level of this stage. Of course, the total number of levels is nm. The necessary rules for the procedure generating the branching tree are the following.

Rule 1 Level 0_l contains only the dummy root node of stage l, $l = 1, 2, \cdots, m$ (each l is starting of a new stage).

Rule 2 Level 1_l contains the nodes $\boxed{1}$, $\boxed{2}$, $\cdots$, $\boxed{x}$, where $x = n - k_l + 1$ (any number larger than x would violate Rules 5 and 7).

Rule 3 A path from level 0_l to level j_l, $i = 1, 2, \cdots, m$, $j = 1, 2, \cdots, n$, may be extended to the level $(j+1)_l$ by any of the nodes $\boxed{1}$, $\boxed{2}$, $\cdots$, $\boxed{n}$, $\textcircled{1}$, $\textcircled{2}$, $\cdots$, $\textcircled{n}$ provided the rules 4 to 7 are observed (all unscheduled jobs at stage l are candidates for $\square$ and $\bigcirc$ nodes as long as they do not violate Rules 4 to 7).

Rule 4 If $\boxed{a}$ or $\textcircled{a}$ has previously appeared as a node at level j_l, then a may not be used to extend the path at that level (this assures that no job is scheduled twice at one stage).

Rule 5 $\boxed{a}$ may not be used to extend a path at level j_l, which already contains some node $\boxed{r}$ with $r > a$ (this is to avoid duplicate generation of sequences in the tree).

Rule 6 No path may be extended in such a way that it contains more than k_l $\square$ nodes at each stage l (this guarantees that no more than k_l machines are used at stage l).

Rule 7 No path may terminate in such a way that it contains less than k_l $\square$ nodes at each stage l unless the number of jobs is less than k_l (there is no advantage in keeping a machine idle if the processing cost is the same for all of the machines).

A sample tree representation of a problem with 4 jobs and 2 parallel machines is given in Figure 8.4.2. All of the end nodes can serve as a starting point for the next stage 0_{l+1} ($l < m$). All of the nodes at a subsequent stage may not be candidates due to their higher value of lower bounds, and thus not all of the

nodes need to be explored. It may also be observed that all of the jobs at stage l will not be readily available at the next stage, and thus inserted idle time will increase their lower bounds and possibly remove them from further considerations. This will help to reduce the span of the tree. The number of search nodes could be further reduced, if the interest is in the subclass of active schedules called *non-delay* schedules. These are schedules in which no machine is kept idle when it could start processing some task.

The use of these schedules does not necessarily provide an optimal schedule, but the decrease in the number of the nodes searched gives a strong empirical motivation to do that, especially for large problems [Fre82].

Finally we describe the idea of the branch and bound algorithm for the problem. It uses a variation of the depth-first least lower bound search strategy, and is as follows.

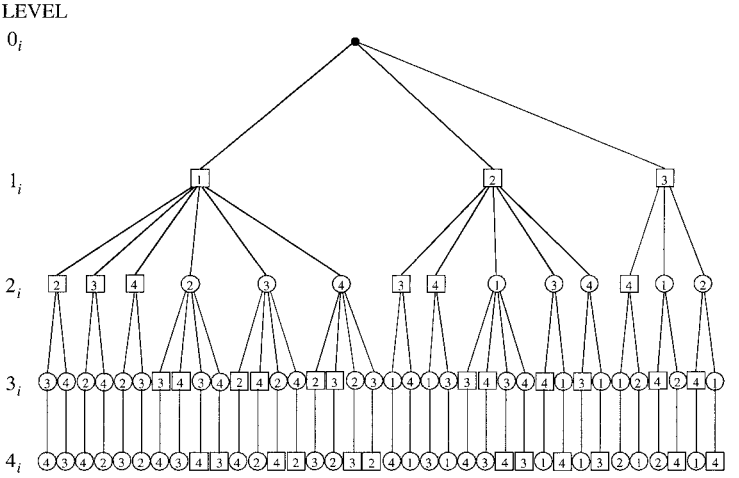

Figure 8.4.2 *Tree representation of four jobs on two parallel machines.*

Step 1 Generate $n-k_1+1$ $\square$ nodes at stage 1 and compute their lower bounds. Encode the information about the nodes and add them to the list of unprocessed nodes. Initialize counters (number of iterations, time) defining end of computation.

Step 2 Remove a node from the list of unprocessed nodes with the priority given to the deepest node in the tree with the least lower bound. Break ties arbitrarily.

Step 3 Procure all information about the retrieved node. If this is one of the end nodes of the tree go to Step 5, while if this is the last node in the list of unprocessed nodes then go to Step 6.

Step 4 Generate branches from the retrieved node and compute their lower bounds. Discard the nodes with lower bounds larger than the current upper bound. Add the remaining nodes to the list of unprocessed nodes and go to Step 2.

Step 5 Save the current complete schedule, as the best solution. If this is the last branch of the tree, or if the limit on the number of iterations or computation time has reached, then pass to the next step, otherwise go to Step 2.

Step 6 Print the results and stop.

As we see, the algorithm consists of three major parts: the branching tree generation, the lower bound computing, and the list processing part. The first two parts are based on the concepts described earlier with some modifications utilizing specific features of the problem. For the list processing part, the information is first coded for each branching node. If the lower bound is better than the best available C_{max} value of a complete solution (i.e. the current upper bound), provided it is available at the moment, the node is stored in the list of unprocessed nodes. The information stored for each branching node is the following:

$$KODE = NPR \times 1\,000\,000 + NPS \times 10\,000 + LSN \times 100 + JOB$$

$$LBND = NS \times 10\,000\,000 + NSCH \times 100\,000 + LB$$

where *NPR* is the machine number in use, *NPS* is the sequence number of this machine, *LSN* is number of the last □ nodes, *JOB* is the index of the job, *NS* is the index of the stage, *NSCH* is the number in the processing sequence, and *LB* is the lower bound of the node.

The stage and the level numbers are coded in the opposite manner to their position in the tree (the deepest node has the least value). Thus, the deepest node is stored on top of the list and can be retrieved first. If two or more nodes are at the same stage and level, the one with the least lower bound is retrieved first and processed. Once a node is retrieved, the corresponding information is decoded and compared with the last processed node data. If the node has gone down a step in the tree, the necessary information, like sequence position and completion time of the job on the retrieved node, is established and recorded. However, if the retrieved node is at a higher or the same level as the previous node, the working sequence and completion time matrix of the nodes lower than the present level and up to the level of the last node are re-initialized. The lower bound is then compared with the best known one, assuming it is available, and is either eliminated or branched on except when this is the last node in the tree. The qualifying nodes are stored in the list of unprocessed nodes according to the priority rule described in Step 2 of the algorithm. However, in case this is the last node in the

tree, and it satisfies the lower bound comparison test, the working sequence position and job completion time matrix along with the completion time of the schedule is saved as the best known solution.

Of course, the algorithm described above is only a basic framework for further improvements and generalizations. For example, in order to improve the computation speed for large problems some elimination criteria, like the ones developed in [Bra88] can be used together with the lower bounds. The lower bound in Step 1 and Step 4 of the algorithm are computed according to (8.4.20), using (8.4.18) and (8.4.19). The algorithm could also be applied for schedule performance measures other than the schedule length, if corresponding lower bounds would be elaborated. Moreover, the idea of the algorithm can be used in a heuristic way, e.g. by setting up a counter of the number of nodes to be fully explored or by defining a percentage improvement index on each new feasible solution.

The algorithm of Portmann et al. [PVDD98] extends that of Brah and Hunsucker in several ways. First, it uses the improved machine based lower bound (8.4.22) instead of (8.4.18) when computing (8.4.20). Moreover, it computes an upper bound before starting to schedule the jobs at a new stage l. The upper bound is computed by a genetic algorithm (*GA*) that determines a schedule of all jobs at stages l through m. The schedule of the jobs at the first $l-1$ stages is fixed and is given by the path from the root of the branching tree to the root node of stage l. For details of *GA* we refer the reader to [RC92].

The results of a detailed computational study show that the method of Portmann et al. is able to solve problems to optimality with up to five stages and ten or fifteen jobs. However, it seems that the method is very sensitive to the pattern of the number of parallel machines at the stages. Another conclusion is that the algorithm proves the optimality of solutions, within a given time limit, more frequently when *GA* is used.

A Method Based on Constraint Propagation

The method of Carlier and Néron [CN00] is significantly different from that of Brah and Hunsucker and of Portmann et al. The novelty of the approach consists in working on all m parallel machine problems at the same time. Namely, instead of solving the parallel machine problem completely at a stage, like in the branch-and-bound algorithm of Brah and Hunsucker, the method selects a stage and the next job to be processed at that stage. Having scheduled the selected job, heads and tails are adjusted and the method proceeds with selecting a new stage.

First we discuss how to select the job to be scheduled next at some stage with respect to a fixed upper bound UB. To simplify notation fix a stage l and consider the $\bar{m}$ machine problem with $\bar{m} = k_l$. We identify the processing of job J_i at stage l with task T_i. The processing time of task T_i is $\bar{p}_i = p_{li}$ and its starting time (to be determined) will be t_i. Let $\mathcal{B}$ denote the set of all $\bar{m}$ tasks. A central

notion is that of selection. A *selection A* for an $\bar{m}$-machine problem is an ordered list of tasks $\{T_{i_1}, T_{i_2}, \cdots, T_{i_{h-1}}, T_{i_h}\}$ such that: if T_i precedes T_j in A then $t_i < t_j$, or $t_i = t_j$ and $i < j$. A selection is complete if $\mathcal{B}$ is totally ordered. To complete the definition note that in a selection more than one task can be processed at the same time, but the total number of tasks processed simultaneously cannot exceed m.

Carlier [Car84] has proposed a simple list scheduling algorithm, the *Strict* algorithm, to schedule tasks with respect to a selection at their earliest possible date, the result is called *strict schedule*. It is shown that strict schedules dominate all other schedules. Consequently, it is enough to work with strict schedules.

Let us fix an upper bound UB for the m-machine problem. A task $T_i \in \mathcal{B}$ is an *input* (*output*) of the $\bar{m}$-machine problem if and only if there exists a schedule $S = \{ t_j \mid T_j \in \mathcal{B} \}$ with makespan at most UB and verifying $t_j \geq t_i$ (respectively $t_j + \tilde{p}_j \leq t_i + \tilde{p}_i$) for all $T_j \in \mathcal{B} - \{T_i\}$. Inputs and outputs will be selected by computing lower bounds after fixing a task T_i to be scheduled before or after all other unscheduled tasks. However, lower bounds may not detect that no schedule of the remaining tasks with makespan at most UB exists.

For solving the makespan minimization problem, Carlier and Néron solve the decision version of the problem and apply a dichotomic search to find the smallest UB for which a solution exists.

The decision problem is solved by branch-and-bound in which branching consists of fixing a task as input (or output) of a stage. More concretely, the branch-and-bound method proceeds as follows:

Step 1. Determine the most critical (machine) center, which is the set of parallel machines on some stage that will most likely create a bottleneck when scheduling all jobs (see below). Decide if the selection is built according to inputs or outputs. If selection based on outputs is chosen then reverse the problem.

Step 2. If *bestsolution* $\leq UB$ then answer YES and stop. Otherwise, if all nodes are explored then answer NO and stop. Otherwise proceed with Step 3.

Step 3. Choose the node N in the branch-and-bound tree to be explored. If the current center, i.e. the parallel machine problem under consideration in node N, is completely selected then proceed with Step 4, otherwise proceed with Step 6.

Step 4. If all centers are completely selected in N and *solution* $\leq UB$ then answer YES and stop. Otherwise, if there exists a center in N not completely selected then choose the most critical center among the not completely selected ones as the current center of N and proceed with Step 5. In all other cases proceed with Step 6.

Step 5. Determine a solution for N. If the makespan of the solution found is not greater than UB then answer YES and stop.

Step 6. Compute lower bounds with respect to the current center of N. If *lowerbounds* > UB then discard node N and go to Step 2. Otherwise proceed with Step 7.

Step 7. Apply local enumerations to N and proceed with Step 8.
Step 8. Determine the list of feasible inputs for the current center of N. For each feasible input i create a new node by adding i to the partial selection of the current center of N and adjust heads and tails. Go to Step 2.

Below we provide some details of this algorithm:

- *The most critical center*: a lower bound is computed for each $\bar{m} = k_l$-machine problem. The $\bar{m}$-machine problem with the largest lower bound defines the most critical center which will be selected first.
- *The current center*: the center where the selection is built and it is always the most critical center.
- The search tree is visited in a depth-first manner such that, among the children of a node, the child with the smallest release date of its input is chosen for exploration.
- *Solutions* are generated during the exploration of the tree using the Strict list scheduling algorithm. The (ordered) list of operations for each center is determined by either a complete selection, if available, or by sorting the operations in decreasing tail order (steps 4 and 5).
- Lower bounds are computed using eq. (8.4.25) and also eq. (8.4.28).
- Local enumerations at Step 7 refer to two things. On the one hand, unscheduled operations are selected in all possible ways while respecting UB in order to improve their heads and tails. On the other hand, a restricted multiprocessor flowshop problem is solved during the construction of the selection of the most critical center.
- The selection of inputs at Step 8 consists in finding jobs that can be scheduled next (before all other unscheduled jobs) without augmenting a lower bound beyond UB.

The adjustments of heads and tails start from the current center and are propagated through the other centers. The efficiency of the head and tail based lower bounds heavily depends on this propagation phase. Moreover it influences the number of feasible inputs and therefore the size of the branching tree.

Assume task T_e has been detected as a possible input in the current machine center. There might be a partial selection within this center which is not yet complete. Let $\tilde{\mathcal{B}}$ be the set of unselected tasks. If T_e is an input all other tasks cannot start before the release time r_e of e, i.e. $r_i := \max\{r_i, r_e\}$ for all tasks T_i of $\tilde{\mathcal{B}}$. For all machines P_k of the current center the machine availability time can be updated to $R_k := \max\{R_k, r_e\}$. The adjustment of the machine availability times again might cause an increase of the release dates of all tasks from $\tilde{\mathcal{B}}$, i.e. $r_i := \max\{r_i, R_1\}$, because a task cannot start before a machine becomes available.

In the domain of possible start times for task e on any machine the latest possible start time is limited by

$$\max\left\{ UB - p_e - q_e,\ UB - \frac{1}{\bar{m}} \cdot \left(\sum_{T_i \in \tilde{\mathcal{B}}} p_i + p_e + q_{j_1} + \ldots + q_{j_{\bar{m}}} \right) \right\}$$

which implies the updating of the tail

$$q_e := \max\left\{ q_e,\ \frac{1}{\bar{m}} \cdot \left(\sum_{T_i \in \tilde{\mathcal{B}}} p_i + p_e + q_{j_1} + \ldots + q_{j_{\bar{m}}} \right) - p_e \right\}.$$

The complexity of this adjustment is at most $O(\bar{m}\bar{n})$.

Consider a partial selection and set $\tilde{\mathcal{B}}$ of unselected tasks at a node in the branch-and-bound tree. Let $G'(\tilde{\mathcal{B}})$ be the lower bound (8.4.25) and UB the current upper bound. Let δ be the smallest non-negative integer such that

$$\frac{1}{\bar{m}} \cdot \left(\sum_{u=1}^{\bar{m}} r_{i_u} + \delta + \sum_{T_j \in \tilde{\mathcal{B}}} p_j + \sum_{u=1}^{\bar{m}} q_{i_u} \right) > UB \quad \text{and} \quad r_{i_{\bar{m}+1}} - r_{i_1} \geq \delta .$$

Thus, we can conclude that $t_{i_1} < r_{i_1} + \delta$ otherwise the aforementioned new value $G'(\tilde{\mathcal{B}})$, where the release date of T_{i_1} has been increased by δ, will be strictly greater than UB. As $t_{i_1} + p_{i_1} + q_{i_1} \leq C_{i_1} + q_{i_1} \leq UB$ we can set $q_{i_1} := \max\{q_{i_1}, UB - (r_{i_1} + \delta + p_{i_1}) + 1\}$. If the new q_{i_1} is greater than the previous one, the same deduction can be applied to q_{i_2}. Similarly, adjustments can be derived for the release dates leading to the following updates $r_{i_1} := \max\{r_{i_1}, UB - (q_{i_1} + \delta' + p_{i_1}) + 1\}$.

The modification of the release dates of the tasks of the current center are propagated to the subsequent machine centers and the new tails are propagated to the previous machine centers.
For more details see [CN00].

As far as the benefits of this method are concerned, most of the problems reported hard by Vignier [Vig97] are very easy to solve by constraint propagation. Those that are not solved immediately, are hard for the new method as well. The method seems to perform well on problem instances in which there is a "bottleneck" center having one machine only.

8.4.5 The Mean Flow Time Problem

We are aware of only very few results on solving multiprocessor flowshop with respect to the mean flow time objective. The general problem has been studied by Azizoglu et al. [ACK01] and a special case where an optimal permutation schedule is sought has been studied by Rajendran and Chaudhuri [RC92]. We commence with a lower bound for the optimal permutation schedule problem and continue with that for the general case. Then a branch-and-bound method for each of the two problems will be presented.

Lower Bounds

Permutation flowshops

Before presenting the lower bound proposed by Rajendran and Chaudhuri for the permutation flowshop problem we introduce additional notation.

σ = a permutation of jobs (indices of the jobs) that defines an available partial schedule,

n' = the number of scheduled jobs in σ,

$\mathcal{U}$ = the set of unscheduled jobs,

$R_k^{(l)}(\sigma)$ = the release time of machine P_k at stage l w.r.t. σ,

$C(\sigma j, l)$ = the completion time of an unscheduled job $J_j \in \mathcal{U}$ at stage l when appended to σ,

$F(\sigma)$ = the total flow-time of jobs in σ,

$LBC_j^{(l)}(\sigma)$ = the lower bound on the completion time of job J_j at stage l,

$LB(\sigma)$ = the lower bound on the total flow-time of all schedules beginning with partial schedule σ.

In a *permutation schedule* the completion times of the jobs at the stages are determined w.r.t. a permutation σ of jobs. The completion times of the jobs in σ are determined iteratively by using the processing times and machine assignments. Namely, assuming that $\sigma = \sigma' j$ and that job J_j is assigned to machine $P_{k(j,l)}$ at stage l, the completion time of job J_j at the first stage is

$$C^{(1)}(\sigma' j) = R_{k(j,1)}^{(1)} + p_{1j} .$$

The completion time at each stage $l = 2, \cdots, m$ (in this order) is determined by

$$C^{(l)}(\sigma' j) = \max\{ C^{(l-1)}(\sigma' j), R_{k(j,l)}^{(l)} \} + p_{lj} .$$

Finally, the release times of the machines at the stages $l = 2, \cdots, m$ are given by

$$R_k^{(l)}(\sigma' j) = \begin{cases} C^{(l)}(\sigma' j) & \text{if } k = k(j,l) \\ R_k^{(l)}(\sigma) & \text{otherwise} \end{cases} .$$

Now we turn to the lower bound. To this end we need other expressions that are defined next. The earliest time when an unscheduled job in $\mathcal{U}$ becomes available at stage l can be computed as follows:

$$s^{(l)} = \max \left\{ \begin{array}{l} \min\{\, R_k^{(1)}(\sigma) \mid 1 \le k \le k_1 \,\} + \min\limits_{J_j \in \mathcal{U}} \left\{ \sum\limits_{q=1}^{l-1} p_{qj} \right\}, \\ \min\{\, R_k^{(2)}(\sigma) \mid 1 \le k \le k_2 \,\} + \min\limits_{J_j \in \mathcal{U}} \left\{ \sum\limits_{q=1}^{l-1} p_{qj} \right\}, \\ \ldots \\ \min\{\, R_k^{(l-1)}(\sigma) \mid 1 \le k \le k_{l-1} \,\} + \min\limits_{J_j \in \mathcal{U}} \{ p_{l-1\,j} \} \end{array} \right\} \quad (8.4.30)$$

Therefore, the earliest starting time of an unscheduled job is given by

$$\max\{\, \min\{\, R_k^{(l)}(\sigma) \mid 1 \le k \le k_l \,\}, s^{(l)} \,\}$$

Let $R_r^{(l)}$ denote $\min\{\, R_k^{(l)}(\sigma) \mid 1 \le k \le k_l \,\}$. With this notation the lower bound on the completion time of job J_{j_1}, where $J_{j_1} \in \mathcal{U}$, at stage l is given by

$$LBC_{j_1}^{(l)}(\sigma) = \max\{R_r^{(l)}, s^{(l)}\} + p_{lj_1} + \sum_{q=l+1}^{m} p_{qj_1} \,. \quad (8.4.31)$$

We place tentatively J_{j_1} on machine P_r and update the machine's release time as

$$R_r^{(l)} = \max\{R_r^{(l)}, s^{(l)}\} + p_{lj_1} \,. \quad (8.4.32)$$

Now, updating $R_r^{(l)}$ is correct only if J_{j_1} is an unscheduled job with smallest processing time. Let $j_1, j_2, \cdots, j_{n-n'}$ be a permutation of the indices of all unscheduled jobs in $\mathcal{U}$ satisfying $p_{j_1}^{(l)} \le p_{j_2}^{(l)} \le \ldots \le p_{j_{n-n'}}^{(l)}$, we compute $LBC_{j_t}^{(l)}(\sigma)$ for $t = 1, \ldots, n-n'$, in this order. Then we obtain the lower bound

$$LB^{(l)}(\sigma) = F(\sigma) + \sum_{J_j \in \mathcal{U}} LBC_j^{(l)}(\sigma) \,,$$

at stage l on the total flow time of all permutation schedules beginning with partial schedule σ.

Finally, a lower bound on the total flow time of any schedule beginning with σ is obtained by computing $LB^{(l)}(\sigma)$ for all stages $1 \le l \le m$ and taking the maximum:

$$LB(\sigma) = \max_{1 \le l \le m} LB^{(l)}(\sigma) \,. \quad (8.4.33)$$

The general case

When schedules are not restricted to permutation schedules Azizoglu et al. [ACK01] propose two other lower bounds obtained by solving two different relaxations of the following parallel machine problem. Let $\Pi^{(l)}$ be the total flow time problem on k_l identical parallel machines and n tasks with processing times

$p_{l1},\ldots,p_{ln}$ and ready times $r_1^{(l-1)},\ldots,r_n^{(l-1)}$. If $F^{(l)}$ is the optimal flow time of problem $\Pi^{(l)}$ then $F^{(l)}$ is a lower bound on the optimal solution to the L-stage flowshop problem. However, $\Pi^{(l)}$ is NP-hard as is its single machine special case. Hence, we compute lower bounds on $F^{(l)}$.

The first lower bound, LB_1, is the optimum of a relaxation of $\Pi^{(l)}$ when all job ready times are set to $\min_j\{r_j^{(l-1)}\}$. This problem can be solved to optimality in polynomial time by the SPT rule, cf. Blazewicz et al. [BEPSW01].

In the second lower bound, LB_2, job ready times are kept, but instead of solving a parallel machine problem, a single machine problem is considered. More precisely, define n new tasks with processing times p_{lj}/k_l and ready times $r_j^{(l-1)}$, $j = 1,\ldots,n$. Total flow time minimization on a single machine with ready times is NP-hard, Lenstra et al. [LRKB77], however its preemptive version can be solved with the shortest remaining processing time (*SRPT*) rule, Schrage [Sch68]. The preemptive optimum is a lower bound on the non-preemptive single machine problem, therefore on $F^{(l)}$.

In the next section we describe algorithms using the bounds presented in this section.

Branch-and-Bound Procedures

Permutation flowshops

Rajendran and Chaudhuri propose a very simple algorithm for solving the permutation flowshop problem. Let k denote the minimum number of parallel machines over all stages, that is, $k = \min_l\{k_l\}$. The algorithm starts by generating $\binom{n}{k}$ nodes, one for each subset of k jobs out of the set of n jobs. In each of these nodes the k jobs are placed on k distinct machines in every stage, and the partial schedule σ is defined accordingly. Then, the lower bound $LB(\sigma)$ (eq. 8.4.33) is computed for each node and the node with the smallest lower bound is selected for exploration. Exploring a node consists in generating $n-n'$ new nodes, one for each of the $n-n'$ unscheduled jobs. When generating a new node using an unscheduled job J_j, then the operations of job J_j are joined to σ starting with the operation at stage 1 and finishing with the operation at stage m. An operation is always placed on a machine having the smallest release time. After computing a lower bound for each child generated, the procedure proceeds by choosing the next node to branch from. The algorithm stops when a node with $n-1$ scheduled jobs is chosen for exploration. Notice that in this case the lower bound matches the flow time of the schedule obtained by scheduling the only unscheduled job. Consequently, when the algorithm stops the node chosen augmented with the unscheduled job constitutes an optimal solution to the permutation flowshop problem.

As far as the power of the above method is concerned, instances up to 10 jobs, 15 stages and up to 4 machines at each stage are solved while exploring only small search trees of less than 10.000 nodes in a short computation time (less than a minute) on a mainframe computer.

The general case

For the general case, Azizoglu et al. propose a new branching scheme which is different from that of Brah and Hunsucker (described in Section 8.4.4) developed for the makespan minimization problem. In each stage, there are n nodes at the first level of the tree, each node representing the assignment of a particular job to the earliest available machine. A node at the n'^{th} level of the tree corresponds to a partial sequence with n' jobs scheduled. Each node at level n' branches to $(n-n')$ nodes each is representing the assignment of an unscheduled job to the earliest available machine.

The number of possible branches is thus $n!$ at each stage. Therefore the total number of leaves at the m^{th} stage is $(n!)^m$.

In fact, the branching scheme of Azizoglu et al. generates only a subset of nodes generated by that of Brah and Hunsucker. The following example of Azizoglu et al. illustrates the difference between the two branching schemes. Suppose there are four jobs satisfying $p_{11} \leq p_{12} + p_{13}$. At the first stage the branching scheme of Brah and Hunsucker would consider to assign job J_1 to the first machine and jobs J_2, J_3 and J_4 to the second machine in this order. In contrast, the new branching scheme under the assumption on job processing times at the first stage would not process job J_4 on the second machine after jobs J_2 and J_3, for processing job J_4 on the first machine would dominate the former partial schedule.

Another dominance relation between schedules comes from the following observation. If $\max\{R_r^{(l)}, r_i^{(l-1)}\} + p_{li} \leq r_j^{(l-1)}$, where J_i and J_j are distinct jobs not yet scheduled at stage l and $R_r^{(l)}$ is the earliest time point when a machine becomes available at stage l, then processing job J_i next dominates any schedule in which job J_j is processed next. The branch-and-bound tree generated contains only non-dominated nodes. A lower bound is computed for each node not eliminated using either LB_1 or LB_2 (defined in the previous section).

Computational results show that the new branching scheme with LB_1 outperforms the algorithm using the new branching scheme and LB_2 and also the algorithms using the branching scheme of Brah and Hunsucker with either lower bound. The largest problem instances on which the methods were tested consisted of 15 jobs, 2 stages and at most 5 parallel machines at a stage, and 12 jobs, 5 stages and 4 machines at a stage. Moreover, a general observation is that the larger the number of machines at the first stage, the more difficult the problem becomes. The results are in contrast with the permutation schedule case where

instances with considerably more stages can easily be solved.

For several other interesting conclusions about the properties of the proposed algorithm and also that of the lower bounds we refer the interested reader to [ACK01].

References

ACK01 M. Azizoglu, E. Cakmak, S. Kondakci, A flexible flowshop problem with total flow time minimization, *European J. Oper. Res.* 132, 2001, 528-538.

Ake56 S. B. Akers, A graphical approach to production scheduling problems, *Oper. Res.* 4, 1956, 244-245.

Bak74 K. R. Baker, *Introduction to Sequencing and Scheduling*, J. Wiley, New York, 1974.

Bak75 K. R. Baker, A comparative study of flow shop algorithms, *Oper. Res.* 23, 1975, 62-73.

Bar81 I. Barany, A vector-sum theorem and its application to improving flow shop guarantees, *Math. Oper. Res.* 6, 1981, 445-452.

BDP96 J. Blazewicz, W. Domschke, E. Pesch, The job shop scheduling problem: Conventional and new solution techniques, *European J. Oper. Res.* 93, 1996, 1-33.

BFR75 P. Bratley, M. Florian, P. Robillard, Scheduling with earliest start and due date constraints on multiple machines, *Naval Res. Logist. Quart.* 22, 1975, 165-173.

BH91 S. A. Brah, J. L. Hunsucker, Branch and bound algorithm for the flow shop with multiple processors, *European J. Oper. Res.* 51, 1991, 88-99.

Bru88 P. Brucker, An efficient algorithm for the job-shop problem with two jobs, *Computing* 40, 1988, 353-359.

Car82 J. Carlier, The one machine sequencing problem, *European J. Oper. Res.* 11, 1982, 42-47.

Car84 J. Carlier, *Problèmes d'ordonnancement à contraintes de ressources: algorithmes et complexité*, Thèse d'État, MASI, 1984.

Car87 J. Carlier, Scheduling jobs with release dates and tails on identical machines to minimize makespan, *European J. Oper. Res.* 29, 1987, 298-306.

CDS70 H. G. Campbell, R. A. Dudek, M. L. Smith, A heuristic algorithm for the n job, m machine sequencing problem, *Management Sci.* 16B, 1970, 630-637.

CMM67 R. W. Conway, W. L. Maxwell, L. W. Miller, *Theory of Scheduling*, Addison Wesley, Reading, Mass., 1967.

CP98 J. Carlier, E. Pinson, Jackson's pseudo preemptive schedule for the $Pm|r_i, q_i|C_{max}$ problem, *Ann. Oper. Res.* 83, 1998, 41-58.

CN00 J. Carlier, E. Néron, An exact method for solving the multi-processor flowshop, *RAIRO Rech. Oper.* 34, 2000, 1-25.

CS81 Y. Cho, S. Sahni, Preemptive scheduling of independent jobs with release and due times on open, flow and job shops, *Oper. Res.* 29, 1981, 511-522.

Dan77 D. G. Dannenbring, An evaluation of flow shop sequencing heuristics, *Management Sci.* 23, 1977, 1174-1182.

DL93 J. Du, J. Y.-T- Leung, Minimizing mean flow time in two-machine open shops and flow shops, *J. Algorithms* 14, 1993, 24-44.

DPS92 R. A. Dudek, S. S. Panwalkar, M. L. Smith, The lessons of flowshop scheduling research, *Oper. Res.* 40, 1992, 7-13.

GG64 P. C. Gilmore, R. E. Gomory, Sequencing a one-state variable machine: A solvable case of the traveling salesman problem, *Oper. Res.* 12, 1964, 655-679.

GJ77 M. Garey, D. S. Johnson, Two-processor scheduling with start times and deadlines, *SIAM J. Comput.* 6, 1977, 416-426.

GJS76 M. R. Garey, D. S. Johnson, R. Sethi, The complexity of flowshop and jobshop scheduling, *Math. Oper. Res.* 1, 1976, 117-129.

GLL+79 R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling theory: a survey, *Ann. Discrete Math.* 5, 1979, 287-326.

GS78 T. Gonzalez, S. Sahni, Flowshop and jobshop schedules: Complexity and approximation, *Oper. Res.* 26, 1978, 36-52.

GSKD96 A. Guinet, M. M. Solomon, P. K. Kedia, A. Dussauchoy, A computational study of heuristics for two-stage flexible flowshops, *Int. J. Prod. Res.* 34, 1996, 1399-1415.

GT91 J. N. D. Gupta, E. Tunc, Schedules for the two-stage hybrid flowshop with parallel machines at the second stage, *Int. J. Prod. Res.* 29, 1991, 1489-1502.

Gup71 J. N. D. Gupta, A functional heuristic algorithm for the flow-shop scheduling problem, *Oper. Res. Quart.* 22, 1971, 39-47.

Gup88 J. N. D. Gupta, Two-stage hybrid flowshop scheduling problem, *J. Oper. Res. Soc.* 39, 1988, 359-364.

HC91 J. C. Ho, Y.-L- Chang, A new heuristic for the n-job, M-machine flow-shop problem, *European J. Oper. Res.* 52, 1991, 194-202.

HHLV95 H. Hoogeveen, C. Hurkens, J. K. Lenstra and A. Vandevelde, Lower bounds for the multiprocessor flow shop, In: *Proceedings of the 2nd Workshop on Models and Algorithms for Planning and Scheduling*, Wernigerode, May 22-26, 1995.

HLV96 J. A. Hoogeveen, J. K. Lenstra, B. Veltman, Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard, *European J. Oper. Res.* 89, 1996, 172-175.

HR88 T. S. Hundal, J. Rajgopal, An extension of Palmer's heuristic for the flow-shop scheduling problem, *Internat. J. Prod. Res.* 26, 1988, 1119-1124.

IS65 E. Ignall, L. E. Schrage, Application of the branch-and-bound technique to some flow-shop scheduling problems, *Oper. Res.* 13, 1965, 400-412.

Joh54 S. M Johnson, Optimal two- and three-stage production schedules with setup times included, *Naval Res. Logist. Quart.* 1, 1954, 61-68.

Joh58 S. M. Johnson, Discussion: Sequencing n jobs on two machines with arbitrary time lags, *Management Sci.* 5, 1958, 299-303.

KK88 T. Kawaguchi, S. Kyan, Deterministic scheduling in computer systems: A survey, *J. Oper. Res. Soc. Japan* 31, 1988, 190-217.

KM87 S. Kochbar, R. J. T. Morris, Heuristic methods for flexible flow line scheduling, *J. Manuf. Systems* 6, 1987, 299-314.

KP05 T. Kis and E. Pesch, A review of exact solution methods for the non-preemptive multiprocessor flowshop problem, *European J. Oper. Res.* 164, 2005, 592-608.

Lan87 M. A. Langston, Improved LPT scheduling identical processor systems, *RAIRO Technique et Sci. Inform.* 1, 1982, 69-75.

LLRK78 B. J. Lageweg, J. K. Lenstra, A. H. G. Rinnooy Kan, A general bounding scheme for the permutation flow-shop problem, *Oper. Res.* 26, 1978, 53-67.

LLR+93 E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys, Sequencing and scheduling: algorithms and complexity, in: S. C. Graves, A. H. G. Rinnooy Kan, P. H. Zipkin, (eds.), *Handbooks in Operations Research and Management Science*, Vol. 4: Logistics of Production and Inventory, Elsevier, Amsterdam, 1993.

Lom65 Z. A. Lomnicki, A branch-and-bound algorithm for the exact solution of the three-machine scheduling problem, *Oper. Res. Quart.* 16, 1965, 89-100.

LRKB77 J. K. Lenstra, R. H. G. Rinnooy Kan, P. Brucker, Complexity of machine scheduling problems, *Ann. Discrete Math.* 1, 1977, 343-362.

LV94 C. Y. Lee and G. L. Vairaktarakis, Minimizing makespan in hybrid flowshop, *Oper. Res. Letters* 16, 1994, 149-158.

McM69 G. B. McMahon, Optimal production schedules for flow shops, *Canadian Oper. Res. Soc. J.* 7, 1969, 141-151.

Mit58 L. G. Mitten, Sequencing n jobs on two machines with arbitrary time lags, *Management Sci.* 5, 1958, 293-298.

Mon79 C. L. Monma, The two-machine maximum flow time problem with series-parallel precedence relations: An algorithm and extensions, *Oper. Res.* 27, 1979, 792-798.

Mon80 C. L. Monma, Sequencing to minimize the maximum job cost, *Oper. Res.* 28, 1980, 942-951.

MRK83 C. L. Monma, A. H. G. Rinnooy Kan, A concise survey of efficiently solvable special cases of the permutation flow-shop problem, *RAIRO Rech. Opér.* 17, 1983, 105-119.

MS79 C. L. Monma, J. B. Sidney, Sequencing with series-parallel precedence, *Math. Oper. Res.* 4, 1979, 215-224.

NEH83 M. Nawaz, E. E. Enscore, I. Ham, A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, *Omega* 11, 1983, 91-95.

NS93 E. Nowicki, C. Smutnicki, New results in the worst-case analysis for flow-shop scheduling, *Discrete Appl. Math.* 46, 1993, 21-41.

NS96 E. Nowicki, C. Smutnicki, A fast tabu search algorithm for the permutation flow-shop problem, *European J. Oper. Res.* 91, 1966, 160-175

OP89 I. H. Osman, C. N. Potts, Simulated annealing for permutation flow-shop scheduling, *Omega* 17, 1989, 551-557.

OS90 F. A. Ogbu, D. K. Smith, The application of the simulated annealing algorithm to the solution of the $n/m/C_{max}$ flowshop problem, *Comput. Oper. Res.* 17, 1990, 243-253.

Pal65 D. S. Palmer, Sequencing jobs through a multi-stage process in the minimum total time - a quick method of obtaining a near optimum, *Oper. Res. Quart.* 16, 1965, 101-107.

Per95 M. Perregaard, *Branch and bound methods for the multiprocessor jobshop and flowshop scheduling problems*, Department of Computer Science, University of Copenhagen, Master's Thesis, 1995.

Pot85 C. N. Potts, Analysis of heuristics for two-machine flow-shop sequencing subject to release dates, *Math. Oper. Res.* 10, 1985, 576-584.

PSW91 C. N. Potts, D. B. Shmoys, D. P. Williamson, Permutation vs. non-permutation flow shop schedules, *Oper. Res. Lett.* 10, 1991, 281-284.

PVDD98 M.-C. Portmann, A. Vignier, D. Dardilhac and D. Dezalay, Branch and bound crossed with GA to solve hybrid flowshops, *European J. Oper. Res.* 107, 1998, 389-400.

RC92 C. Rajendran and D. Chaudhuri, A multi-stage parallel-processor flowshop problem with minimum flowtime, *European J. Oper. Res.* 57, 1992, 111-122.

Ree95 C. Reeves, A genetic algorithm for flowshop sequencing, *Comput. Oper. Res.* 22, 1995, 5-13.

Röc84 H. Röck, The three-machine no-wait flow shop problem is NP-complete, *J. Assoc. Comput. Mach.* 31, 1984, 336-345.

RR72 S. S. Reddi, C. V. Ramamoorthy, On the flow shop sequencing problem with no wait in process, *Oper. Res. Quart.* 23, 1972, 323-331.

RS83 H. Röck, G. Schmidt, Machine aggregation heuristics in shop scheduling, *Methods Oper. Res.* 45, 1983, 303-314.

Sal73 M. S. Salvador, A solution of a special class of flowshop scheduling problems, *Proceedings of the Symposium on the theory of Scheduling and its Applications*, Springer, Berlin, 1975, 83-91.

SB92 S. Stöppler, C. Bierwirth, The application of a parallel genetic algorithm to the $n/m/P/C_{\max}$ flowshop problem, in: T. Gulledge, A. Jones, (eds.), *New Directions for Operations Research in Manufacturing*, Springer, Berlin, 1992, 161-175.

Sch68 L. Schrage, A proof of the optimality of shortest remaining processing time discipline, *Operations Research* 16, 1968, 687-690.

SHD95 D. L. Santos, J. L. Hunsucker and D. E. Deal, Global lower bounds for flow shops with multiple processors, *European J. Oper. Res.* 80, 1995, 112-120.

Sid79 J. B. Sidney, The two-machine maximum flow time problem with series-parallel precedence relations, *Oper. Res.* 27, 1979, 782-791.

SS89 C. Sriskandarajah and S. Sethi, Scheduling algorithms for flexible flow-shops: worst and average performance, *European J. Oper. Res.* 43, 1989, 143-160.

Sta88 E. F. Stafford, On the development of a mixed-integer linear programming model for the flowshop sequencing problem, *J. Oper. Res. Soc.* 39, 1988, 1163-1174.

Szw71 W. Szwarc, Elimination methods in the $m \times n$ sequencing problem. *Naval Res. Logist. Quart.* 18, 1971, 295-305.

Szw73 W. Szwarc, Optimal elimination methods in the $m \times n$ sequencing problem, *Oper. Res.* 21, 1973, 1250-1259.

Szw78 W. Szwarc, Dominance conditions for the three-machine flow-shop problem, *Oper. Res.* 26, 1978, 203-206.

Tai90 E. Taillard, Some efficient heuristic methods for the flow shop sequencing problem, *European J. Oper. Res.* 47, 1990, 65-74.

TSS94 V. S. Tanaev, Y. N. Sotskov, V. A. Strusevich, *Scheduling Theory. Multi-Stage Systems*, Kluwer, Dordrecht, 1994.

Van94 A. Vandevelde, *Minimizing the makespan in a multiprocessor flow shop*, Eindhoven University of Technology - Master's Thesis, 1994.

Vig97 A. Vignier, *Contribution à la résolution des problèmes d'ordonnancement de type monogamme, multimachines*, PhD Thesis, 1997.

VBP99 A. Vignier, J.-C. Billaut and C. Proust, Les problèmes d'ordonnancement de type flow shop hybride: état de l'art, *RAIRO Rech. Oper.* 33, 1999, 117-182.

VHHL05 A. Vandevelde, H. Hoogeveen, C. Hurkens, J. K. Lenstra, Lower bounds for the head-body-tail problem on parallel machines: a computational study of the multiprocessor flow shop, *INFORMS J. Comput.* 17, 2005, 305-320.

Wag59 H. M. Wagner, An integer linear-programming model for machine scheduling, *Naval Res. Logist. Quart.* 6, 1959, 131-140.

Wer90 F. Werner, On the combinatorial structure of the permutation flow shop problem, *ZOR* 35, 1990, 273-289.

WH89 M. Widmer, A. Hertz, A new heuristic method for the flow shop sequencing problem, *European J. Oper. Res.* 41, 1989, 186-193.

Wit85 R. J. Wittrock, Scheduling algorithms for flexible flow lines, *IBM J. Res. Develop.* 29, 1985, 401-412.

Wit88 R. J. Wittrock, An adaptable scheduling algorithms for flexible flow lines, *Oper. Res.* 33, 1988, 445-453.