Alcom-FT Technical Report Series
ALCOMFT-TR-03-192

Lower bounds for the head-body-tail problem on parallel machines: a computational study of the multiprocessor flow shop *

Ann Vandevelde

Netherlands Controlling Authority for Milk and Milk Products
The Netherlands

Han Hoogeveen

Institute of Information and Computing Sciences
Utrecht University
The Netherlands

Cor Hurkens

Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
The Netherlands

Jan Karel Lenstra

CWI
The Netherlands

December 5, 2003

Abstract

The multiprocessor flow shop is the generalization of the flow shop in which each machine is replaced by a set of identical machines. As finding a minimum-length schedule is $\mathcal{NP}$-hard, we set out to find good lower and upper bounds. The lower bounds are based on relaxation of the capacities of all machine sets except one. This results in a parallel machine scheduling problem with release dates and delivery times, for which we derive a number of lower bounds. We pay special attention to the time complexity of algorithms for computing these bounds. To obtain the upper bounds a constructive algorithm in subsequent stages is used. We present an experimental comparison of the various lower and upper bounds for the multiprocessor flow shop problem.

Key words. Multiprocessor flow shop, parallel machine scheduling, head-body-tail problems, preemptive schedule, lower bound, computational study.

1 Introduction

In this paper we consider a generalization of the flow shop scheduling environment, which we call the multiprocessor flow shop. We start by defining the l-stage flow shop. There are n jobs, which are denoted by $J_1, \ldots, J_n$, that have to undergo processing on l consecutive stages; each

*This work was partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

job J_j ($j = 1, \ldots, n$) starts its visit of the production facility at stage 1, then proceeds to stage 2, and so on, and finally gets completed at stage l. This is modelled by splitting each job J_j up in a chain of l operations $O_{1j}, \ldots, O_{lj}$, where the hth operation has to be executed at stage h ($h = 1, \ldots, l$). Executing operation O_{hj} requires an uninterrupted period of length p_{hj}, where we assume that p_{hj} is a nonnegative integer; the execution of operation O_{hj} ($h = 2, \ldots, l$) cannot start until operation $O_{h-1,j}$ has been completed. In the traditional flow shop problem, there is only one machine available at each stage. We assume that in each stage h ($h = 1, \ldots, l$) there are m_h identical parallel machines available for processing the operations O_{hj} ($j = 1, \ldots, n$), which means that operation O_{hj} ($j = 1, \ldots, n$) has to be processed by exactly one of the m_h machines, which takes p_{hj} time units on each choice of machine. We further assume that each machine is continuously available from time zero onwards and that each machine can handle at most one job at a time. Our goal is to minimize the schedule length or makespan, which is defined as the time at which the last job is completed at stage l. Extending the $\alpha|\beta|\gamma$ notation scheme proposed by Graham, Lawler, Lenstra, and Rinnooy Kan (1979), we denote this problem as $F(P)||C_{\max}$ or $Fl(Pm_1, Pm_2, \ldots, Pm_l)||C_{\max}$, depending on whether the number of stages and the number of machines do or do not belong to the input. The multiprocessor flow shop has also appeared in the literature under the names *flexible flow shop* and *hybrid flow shop*.

One of the first results in scheduling has been derived by Johnson (1954), who presents an $O(n \log n)$ algorithm to solve $F2||C_{\max}$, that is, the two-stage problem with one machine per stage. This work was motivated by the need to model and plan the simplest conveyor belt, which has only two stations with one machine/person per station. This was quickly followed by a generalization with m stages with one machine per stage. In many practical situations, there is a number of machines/persons available at each station of the conveyor belt; this is modelled by replacing the single machine by a set of parallel machines, which leads to the multiprocessor flow shop model. The two-stage problem has first been studied by Buten and Shen (1973), who present an approximation algorithm with claimed worst-case ratio 2 for the two-stage problem with m_1 and m_2 identical parallel machines. After that, this two-stage problem, or special cases of it, have been the subject of many studies. Hoogeveen, Lenstra, and Veltman (1996) show that even the simplest case, with $m_1 = 2$ and $m_2 = 1$, is already strongly $\mathcal{NP}$-hard, even if preemption is allowed. Langston (1987), Sriskandarajah and Sethi (1989), Chen (1994), Chen (1995), and Lee and Vairakterakis (1994) present approximation algorithms for the two-stage problem and analyze their worst-case performance behavior. Chen (1994) and Lee and Vairakterakis (1994) present an overview and show that the best algorithm known so far from a worst-case point of view has worst-case ratio $2 - \frac{1}{\max\{m_1,m_2\}}$; this bound also holds for the preemptive case (Chen, 1994). Gupta, Hariri, and Potts (1997) analyze the special case with $m_2 = 1$; they present a computational study, in which they analyze a number of heuristics, and a branch-and-bound algorithm that is able to solve instances with up to 250 jobs and up to 10 machines in the first stage.

The multiprocessor flow shop problem with more than two stages is less well studied. Lee and Vairakterakis (1994) show that the approximation algorithm developed for the two-stage multiprocessor flow shop problem can be generalized to an approximation algorithm for the l stage problem with worst-case bound $l - \frac{1}{\max\{m_1,m_2\}} - \frac{1}{\max\{m_3,m_4\}} \cdots - \frac{1}{\max\{m_{k-1},m_k\}}$. Brah and Hunsucker (1991) present a branch-and-bound algorithm that is able to solve instances with up to 8 jobs and up to 5 stages. Santos, Hunsucker, and Deal (1995) and Portmann, Vignier, Dardilhac, and Dezalay (1998) report that the limited success of this algorithm is partly due to the weak lower bound; in both papers a strengthening of Brah and Hunsucker's lower bound is presented. Vandevelde (1994) presents a number of lower bounds based on the idea of increasing the machine capacity to infinity in all stages but one; this results in a parallel machine scheduling problem with release dates and delivery times. This problem is still $\mathcal{NP}$-hard in the strong sense, but it can be relaxed further to obtain some strong lower bounds. Horn (1974), Labetoulle, Lawler, Lenstra, and Rinnooy Kan (1984), Vandevelde (1994), and Carlier and Pinson (1998) present algorithms to derive lower bounds for this subproblem. In her computational study Vandevelde compares the strength of these lower bounds. Perregaard (1995) has implemented the algorithms suggested for lower bound computation and compared their running times. He uses the lower bounds and a number of smart branching rules in a branch-and-bound framework, which is quite successful. Gharbi and Haouari (2002) present a new lower bound, which is used in a branch-and-bound algorithm that is capable to solve instances with up to 300 jobs of the parallel machine scheduling problem with release dates and delivery times. Recently, Haouari and Gharbi (2003) have presented an improvement of Horn's lower bound. Local search algorithms have been presented by Perregaard (1995), Portmann, Vignier, Dardilhac, and Dezalay (1998) and Mastrolilli and Gamberdella (2000).

The goal of this paper, which is largely based on the work by Vandevelde (1994), is to find good lower and upper bounds and compare the quality of the lower bounds. The paper is organized as follows. In Section 2, we discuss our lower bounds and derive relationships between these bounds and the preemptive bound by Horn (1974) and Labetoulle, Lawler, Lenstra, and Rinnooy Kan (1984). In Section 3 we describe efficient algorithms to compute some of these bounds. We also describe an approximation algorithm to find a reasonably good preemptive schedule, with a bound on the absolute error, which is shown to be tight. Next, in Section 4 we describe approximation algorithms for computing reasonable solutions to the multiprocessor flow shop scheduling problem. In Section 5 experimental results are discussed and some conclusions are given.

2 Lower bounds

By relaxing the machine capacity constraints in all stages except stage h, we obtain a *parallel machine* problem, which we call the *relaxation for stage h*. In the relaxation for stage h, we have the following data for job J_j: a *release time* or *head* $r_j = \sum_{k=1}^{h-1} p_{kj}$, a *processing time* or *body* $p_j = p_{hj}$, and a *delivery time* or *tail* $q_j = \sum_{k=h+1}^{l} p_{kj}$. A schedule is a set of starting times $S_j \geq r_j$, for each $j \in J$, where we use J to denote the index set $\{1, 2, \ldots, n\}$. Job j is completed at time $C_j = S_j + p_j$ and delivered at time $D_j = C_j + q_j$. The *maximum delivery time* (which is sometimes also called the makespan) is the earliest time at which all jobs are delivered. Using the three-field notation scheme, we denote this problem as $P|r_j, q_j|D_{\max}$ and set $m = m_h$; we use $D_{\max}$ instead of the more common $C_{\max}$ in the third field to avoid any possible confusion about the objective function. This problem is also known as the *head-body-tail* problem. It is known to be equivalent to minimizing maximum lateness, where the *lateness* of job J_j is defined as $L_j = C_j - d_j$ and the *due date* of job J_j is equal to $d_j = -q_j$; since $d_j \leq 0$, each job is late then. The parallel machine problem subject to release dates and delivery times is strongly $\mathcal{NP}$-hard. Carlier (1987) presents a branch-and-bound algorithm for it that uses the lower bounds (1) and (2) below and a branching rule based on splitting execution intervals. If we want to check whether it is possible to have all jobs delivered at time D, then we get a feasibility problem with release dates r_j and *deadlines*, where a deadline puts an upper bound on the machine completion time C_j; we denote the deadline of job J_j $(j = 1, \ldots, n)$ by $\bar{d}_j$, which get the value $\bar{d}_j = D - q_j$. The feasibility problem is an important problem in the area of *constraint satisfaction*; we refer to the book by Baptiste, Le Pape, and Nuijten (2001) for more information.

In this section we describe and propose several lower bounds for $P|r_j, q_j|D_{\max}$. We study their efficient computation and practical effectiveness. Since we want to employ them in a branch-and-bound algorithm, we need algorithms of low time complexity to compute them; here we face the trade-off between a stronger bound, which helps to reduce the number of nodes in the branch-and-bound tree, and a larger running time to compute the bound.

The first bound is due to Carlier (1987). The idea behind it is to compute the minimum makespan when only one job J_j of the original instance is maintained. Since each choice of J_j renders a lower bound, the maximum of these lower bounds is a lower bound too. This bound is called the *job based bound JB*, and it is defined by

$$JB = \max_{j \in J} \{r_j + p_j + q_j\} = \max_{j \in J} \sum_{k=1}^{l} p_{kj} \ . \tag{1}$$

It is independent of the choice of h and can be calculated in $\mathcal{O}(n)$ time.

A generalization of the job based bound has been proposed by Jurisch (1995) in the context of a multiprocessor job shop scheduling problem. He shows that the optimum can be computed efficiently for instances consisting of two jobs. Obviously, this will only improve upon JB if there exists at least one stage with only a single machine. We do not discuss this bound any further.

Our second bound, called the *set based bound SB*, is also due to Carlier (1987). The idea behind it is that the value D of the maximum delivery time should be such that the total execution time mD of the machines is large enough to account for the minimum possible idle time due to the release dates and the delivery times as well as the total processing time. We sort the release dates and delivery times; we use $\bar{r}_j$ and $\bar{q}_j$ $(j = 1, \ldots, n)$ to denote the jth smallest release date and delivery time, respectively. Since each job is processed by at most one machine at a time, we have that in the interval $[0, \bar{r}_1]$ each machine is idle, in the interval $[\bar{r}_1, \bar{r}_2]$ at least $(m-1)$ machines are idle, etc.; the first time at which there is a job to process for each machine is time $\bar{r}_m$. Hence, the total idle time in the beginning of the schedule amounts to at least $(\bar{r}_1 + \ldots + \bar{r}_m)$. We derive a similar lower bound on the total idle time at the end of the schedule by noting that after time $D - \bar{q}_1$ each machine must be idle, if the maximum delivery time is equal to D, at most one machine can be active in the interval $[D - \bar{q}_2, D - \bar{q}_1]$, etc.; the last time at which each machine can be active is time $D - \bar{q}_m$. Therefore, the total amount of idle time at the end of the schedule amounts to at least $(\bar{q}_1 + \ldots + \bar{q}_m)$. The remaining time, which is equal to mD minus the idle time, should be at least equal to $\sum_{j=1}^{n} p_j$. Working things out, we find that the lower bound for the maximum delivery time

$$SB = (\bar{r}_1 + \ldots + \bar{r}_m + \sum_{j \in J} p_j + \bar{q}_1 + \ldots + \bar{q}_m)/m \ . \tag{2}$$

Since all data are integers, $\lceil SB \rceil$ is a valid lower bound too. The bound can be calculated in $\mathcal{O}(n)$ time using median finding techniques. We implicitly assume that there are at least as many jobs as machines, since the optimal schedule is trivial and has makespan JB, otherwise.

The bound SB can be calculated with respect to each subset I of J with $|I| \geq m$, and the maximum over all subsets is defined as the *subset based bound SSB*:

$$SSB = \max_{I \subseteq J, |I| \geq m} SB(I) \ , \tag{3}$$

where $SB(I)$ is the set based bound restricted to job set I. Although there are 2^n possible subsets, the bound SSB can be calculated in $\mathcal{O}(n^2)$ time using an integer linear programming formulation. The method is described in Subsection 3.1. Again, we can round SSB up to the nearest integer.

Carlier and Pinson (1998) present an algorithm to compute a 'schedule', which they call *Jackson's pseudo-preemptive schedule*, whose makespan is equal to the maximum of JB and SSB. They state that it can be made to run in $O(n \log n + nm \log m)$ time. Perregaard (1995) has implemented the algorithm to run in $O(nm \log n)$ time and reports that in practice the running time is independent of m. The proof by Carlier and Pinson that the outcome is indeed equal to $\max\{JB, SSB\}$ is rather cumbersome, and we therefore present our own proof in Subsection 3.4.

The optimal solution of the preemptive parallel machine problem $P|r_j, q_j, pmtn|D_{\max}$ is a lower bound on $P|r_j, q_j|D_{\max}$ and we refer to this as the *preemptive bound PB*. Horn (1974) proposes a flow algorithm to test whether a certain trial value for PB is feasible or not that runs in $\mathcal{O}(n^3)$ time. PB can be found using binary search. We claim that

$$0 \leq PB - \max\{JB, SSB\} \leq \frac{m-1}{m} p_{\max} \ . \tag{4}$$

The first inequality states that both JB and SSB are lower bounds on PB, which is obvious. The second inequality is proven in Subsection 3.3 by exhibiting an approximation algorithm for computing a preemptive schedule. This algorithm has also been considered by Sanlaville (1995) and is quite similar to the one described by Carlier and Pinson (1998) to compute the value of $\max\{JB, SSB\}$, which we describe in Section 3.4. In Subsection 3.5 we present an example showing that the upper bound $\max\{JB, SSB\} + \frac{m-1}{m} p_{\max}$ on PB is tight.

Using the observation above, we can compute $\lceil PB \rceil$ in $\mathcal{O}(n^3 \log p_{\max})$ time by searching the interval of possible values for $\lceil PB \rceil$ through bisection, where we use Horn's algorithm to test whether there exists a preemptive schedule with makespan equal to the given trial value.

Haouari and Gharbi (2003) have recently presented a method to improve on PB. The computation of the improved bound is again based on finding the smallest trial value for which a feasible preemptive schedule exists, but they retain the following feature of a non-preemptive schedule: *if J_j must be executed in the interval $[r_j, \bar{d}_j]$ and $\bar{d}_j - r_j < 2p_j$, then necessarily J_j is processed in the interval $[\bar{d}_j - p_j, r_j + p_j]$*. This information can easily be incorporated in Horn's algorithm and yields a bound that dominates the preemptive bound.

Next we present a different way to compute a bound that may improve on the preemptive bound by retaining a non-preemptive feature. Recall that in the computation of SB we use the sum of the m smallest release dates and delivery times. In a preemptive schedule, it is well possible that both r_j and q_j are included for some job J_j. But if preemption is not allowed, then this can only be the case if this job J_j occupies a machine of its own. Hence, we can adjust the set based bound, which results in the *adjusted set based bound*, by taking the following two options into account if r_j and q_j both appear in SB:

- Some machine processes job J_j and the other machines process all other jobs.

- Job j is not both the first and the last job to be processed on a machine.

In the first case we have two lower bounds, one on the machine with the single job and one on the other machines and jobs. In the second case SB can be strengthened by adding the condition that at most one of r_j and q_j appear in the bound. To use this observation we introduce the notation $SB(A, B, c)$, where A is the set of jobs for which we calculate the bound, B is the set of jobs of which not both head and tail appear in the bound, and c is the number of machines on which the jobs in A have to be processed; we call this the *constrained set based bound*. As long as there is a job of which both head and tail appear in the bound $SB(A, B, c)$, we can improve the bound. We define the *adjusted set bound* $ASB(A, B, c)$ as the optimal value found by using this procedure recursively:

$$ASB(A,B,c) = \begin{cases} \min\{\max\{r_j + p_j + q_j, ASB(A\backslash\{j\}, B, c-1)\}, ASB(A, B \cup \{j\}, c)\} \\ \qquad\qquad \text{if both } r_j \text{ and } q_j \text{ appear in } SB(A,B,c) \text{ for some } j \\ \\ SB(A,B,c) \qquad\qquad \text{otherwise} \end{cases}$$

Initially, we have to determine $ASB(J, \emptyset, m)$, that is, we have not 'branched' on the decision of, either giving some job J_j a machine of its own, or using only its head or tail in the computation of the set based bound. Suppose that we have to compute $ASB(A, B, c)$ in the recursive procedure. We first compute $SB(A, B, c)$, for which we describe an $O(m \log m)$ algorithm in Subsection 3.2. If there is no job $J_j \in A$ for which its head and tail are used in the computation of $SB(A, B, c)$, then we cannot adjust the bound $SB(A, B, c)$ any further, and we stop. Otherwise, we select such a job J_j and 'branch' on the decision of giving it a machine of its own, or requesting that we cannot use both r_j and q_j in the computation of the bound. When J_j gets its own machine, then $D_j = r_j + p_j + q_j$. The remaining jobs, which are the jobs in $A\backslash\{j\}$, have to be executed by the remaining $c-1$ machines with the additional constraint that the jobs in B do not have a machine of their own; hence, we have that $ASB(A\backslash\{j\}, B, c-1)$ is a lower bound on the delivery time of these jobs. Therefore, we find that the maximum of these two expressions is a lower bound on the delivery time for the case that J_j gets a machine of its own; this corresponds with the first part of the upper part of the expression. If we decide not to give J_j a machine of its own, then we must include it into the set B, which yields the second part of the expression. Obviously, we follow the better branch, and hence we take the minimum of these two parts.

What is left is the number of recursive calls to compute $SB(A, B, c)$. Obviously we only take into account the jobs with the $2c$ smallest heads and $2c$ smallest tails. In any iteration, either a job gets its own machine, or B is augmented. Since at most $c-1$ jobs can be given their own machine and since the cardinality of B is bounded by $2c$, the depth of the tree that we build when computing $ASB(J, \emptyset, m)$ is bounded by $2m$, which implies that the number of recursive calls is bounded by 2^{2m}. Since $SB(A, B, c)$ can be computed in $O(m \log m)$ time, the overall time complexity is equal to $O(2^{2m} m \log m)$.

The bound ASB can be calculated for every subset. Taking the maximum over all subsets would yield a strong lower bound, but we have not been able to find an efficient implementation yet. Therefore, we define the *adjusted subset bound* $ASSB$ as the bound ASB restricted to the subset of jobs for which the value of SSB is attained:

$$ASSB = ASB(I^*, \emptyset, m) \ , \tag{5}$$

where I^* is a subset for which $SB(I^*) = \max_{I \subseteq J, |I| \geq m} SB(I)$.

Our last set of lower bounds also tries to improve on the set based bound. In case the number of machines does not divide the number of jobs, we may improve the set based bound as follows. Define $m^* = n \bmod m$ and $n^* = m^* \lceil n/m \rceil$. The m^* machines with the most jobs together process at least n^* jobs. On this observation we base the *rest bound* RB, defined by

$$RB = \frac{1}{m^*}(\bar{r}_1 + \ldots + \bar{r}_{m^*} + \sum_{j=1}^{n^*} \bar{p}_j + \bar{q}_1 + \ldots + \bar{q}_{m^*}) \ . \tag{6}$$

The rest bound can be calculated in $\mathcal{O}(n)$ time.

It seems profitable to regard small subsets of jobs with large processing times. We therefore define the *subset rest bound* SRB as

$$SRB = \max_k RB(I_k) \ , \tag{7}$$

where $I_k = \{j | p_j \geq \bar{p}_k, \ j \in J\}$. This bound can be calculated in $\mathcal{O}(n^2)$ time.

Let $D^*_{\max}$ denote the optimal solution value for $P|r_j, q_j|D_{\max}$. The relations between the various lower bounds are shown in Figure 1, where $A \rightarrow B$ if A dominates B. If there is no arc between A and B, then both $A < B$ and $A > B$ is possible, depending on the instance. In particular, adjustment may take SB beyond PB. The improvement of PB by Haouari and Gharbi (2003) is not displayed in the figure; it is in between PB and $D^*_{\max}$, but incomparable to the bounds that are incomparable to PB.

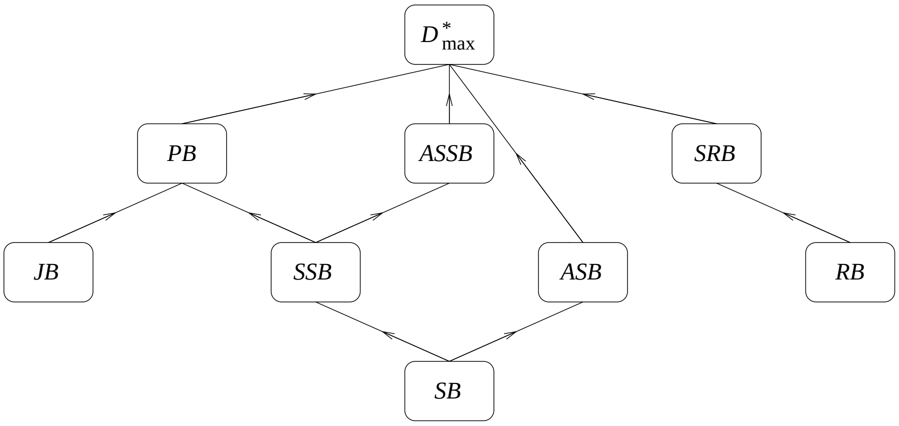

Figure 1: Relations between the lower bounds for $P|r_j, q_j|D_{\max}$.

The last bound that we discuss is the lower bound by Gharbi and Haouari (2002), which we did not include in our experiments. This lower bound is based on the lower bound by Webster (1996) for the special case of $P||C_{\max}$, where machine M_i is not available in the period $[0, u_i]$ $(i = 1, \ldots, m)$. Webster identifies a number of cases in which the *Longest Processing Time* (LPT) rule yields an optimal schedule, where the LPT rule schedules from among the set of unprocessed jobs the one with the largest processing time on the first available machine. Gharbi and Haouari modify the instance of $P|r_j, q_j|D_{\max}$ such that an instance of $P||C_{\max}$ remains that satisfies the requirements of one of the special cases identified by Webster. This modification proceeds as follows: First, they put $u_i \leftarrow \bar{r}_i$, for $i = 1, \ldots, m$; next, they add m dummy tasks J_{n+i} ($i = 1, \ldots, m$) with processing time $p_{n+i} = \bar{q}_i$, for $i = 1, \ldots, m$; and finally they modify the processing times, where they see to it that none of these are increased. The resulting instance of $P||C_{\max}$ with machine unavailabilities is then solved to optimality through the LPT-rule, which yields a lower bound on the outcome value of the original $P|r_j, q_j|D_{\max}$ problem. Obviously, this approach can be applied to any subset of the jobs, but there is no efficient implementation known that computes the maximum over all subsets. Because of the possible reduction of the processing times, this lower bound is uncomparable to any of the other lower bounds.

3 Efficient computations

In this section we address the problem of computing lower bounds for the head-body-tail problem efficiently. We will first show how the subset based bound can be computed in $\mathcal{O}(n^2)$ time and how the constrained set based bound can be computed in $O(m \log m)$ time. Next we discuss ways to circumvent or short-cut the expensive computation of the preemptive bound. We will describe a method to find the value of $\max\{JB, SSB\}$ by computing the length of the relaxed preemptive schedule proposed by Carlier and Pinson (1998).

3.1 Efficient computation of the subset lower bound

In this section we show how the subset based bound can be efficiently calculated. After formulating $SB(I)$ as a linear program and taking its dual, we are able to decompose the computation of SSB into n^2 subproblems and to solve each of these in constant time using appropriate data structures. We start by giving an alternative formulation for the set based bound:

$$mSB(J) = p(J) + \min_{J_R \subseteq J, |J_R| = m} r(J_R) + \min_{J_Q \subseteq J, |J_Q| = m} q(J_Q) \tag{8}$$

Here $p(J)$ denotes the sum of p_j over all $j \in J$, etc.; we have to choose the subsets J_R and J_Q, which both have cardinality m, such that the above expression is minimized. The problem of determining SSB boils down to the problem of finding the subset I of the job set J that maximizes $SB(I)$. Hence, this is a max-min problem, which we can solve efficiently once we have further reformulated the problem by applying the following steps, which we explain below. Here we use the following notation. We first introduce binary variables x_j and y_j, which indicate whether job J_j is selected in J_R (then $x_j = 1$) and in J_Q (then $y_j = 1$). The variables u, v, α_j, β_j are dual variables. We use $(a)^+$ as a short-hand notation for $\max\{a, 0\}$.

$$mSSB(J) = \max_{I \subseteq J, |I| \geq m} mSB(I) \tag{9}$$

$$= \max_{I \subseteq J, |I| \geq m} \left(\begin{array}{l} \min p(I) + \sum_{j \in I} x_j r_j + \sum_{j \in I} y_j q_j \\ \text{s.t.} \left\{ \begin{array}{rcl} \sum_{j \in I} x_j & = & m \\ \sum_{j \in I} y_j & = & m \\ x_j, y_j & \in & \{0, 1\} \end{array} \right. \end{array} \right) \tag{10}$$

$$= \max_{I\subseteq J, |I|\geq m} \left(\begin{array}{l} \min p(I) + \sum_{j\in I} x_j r_j + \sum_{j\in I} y_j q_j \\ \text{s.t.} \left\{ \begin{array}{rcl} \sum_{j\in I} x_j & = & m \\ \sum_{j\in I} y_j & = & m \\ x_j & \leq & 1 \\ y_j & \leq & 1 \\ x_j, y_j & \geq & 0 \end{array} \right. \end{array} \right) \tag{11}$$

$$= \max_{I\subseteq J, |I|\geq m} \left(\begin{array}{l} \max p(I) + m(u+v) + \sum_{j\in I} \alpha_j + \sum_{j\in I} \beta_j \\ \text{s.t.} \left\{ \begin{array}{rcl} u + \alpha_j & \leq & r_j \\ v + \beta_j & \leq & q_j \\ \alpha_j, \beta_j & \leq & 0 \end{array} \right. \end{array} \right) \tag{12}$$

$$= \max_{I\subseteq J, |I|\geq m} \max_{u,v} \left(m(u+v) + \sum_{j\in I} (p_j - (u-r_j)^+ - (v-q_j)^+) \right) \tag{13}$$

$$= \max_{I\subseteq J, |I|\geq m} \max_{\substack{u = r_1, r_2, \cdots, r_n \\ v = q_1, q_2, \cdots, q_n}} \left(m(u+v) + \sum_{j\in I} (p_j - (u-r_j)^+ - (v-q_j)^+) \right) \tag{14}$$

$$= \max_{\substack{u = r_1, r_2, \cdots, r_n \\ v = q_1, q_2, \cdots, q_n}} \max_{I\subseteq J, |I|\geq m} \left(m(u+v) + \sum_{j\in I} (p_j - (u-r_j)^+ - (v-q_j)^+) \right) \tag{15}$$

$$= \max\{ \max_{I\subseteq J, |I|=m} \sum_{j\in I} (p_j + r_j + q_j), \tag{16}$$

$$\max_{\substack{u = r_1, r_2, \cdots, r_n \\ v = q_1, q_2, \cdots, q_n \\ |I^+(u,v)| > m}} \max_{I\subseteq J, |I|\geq m} \left(m(u+v) + \sum_{j\in I} (p_j - (u-r_j)^+ - (v-q_j)^+) \right) \}$$

$$= \max\{ \max_{I\subseteq J, |I|=m} \sum_{j\in I} (p_j + r_j + q_j), \tag{17}$$

$$\max_{\substack{u = r_1, r_2, \cdots, r_n \\ v = q_1, q_2, \cdots, q_n \\ |I^+(u,v)| > m}} \left(m(u+v) + \sum_{j\in I^+(u,v)} (p_j - (u-r_j)^+ - (v-q_j)^+) \right) \}$$

where $I^+(u,v) = \{j \in J | p_j - (u-r_j)^+ - (v-q_j)^+ > 0\}$.
The above equalities are accounted for as follows:

(10) Follows from stating the minimization problem as a 0-1 linear programming problem.

(11) Follows from (10) as the integrality constraints in (10) can be neglected.

(12) Follows directly from (11) by dualizing the inner minimization problem; u and v are free variables.

(13) Follows as for fixed u and v in (12) the maximum is obviously attained by taking $\alpha_j = \min\{0, r_j - u\} = -\max\{u - r_j, 0\}$, and $\beta_j = \min\{0, q_j - v\} = -\max\{v - q_j, 0\}$.

(14) Is derived from (13) by the observation that the expression between brackets is continuous, piecewise linear, and concave, in both u and v. It suffices to evaluate the expression in its breakpoints $u \in \{r_1, \cdots, r_n\}$, and $v \in \{q_1, \cdots, q_n\}$.

(15) Results from (14) as we may maximize in any order.

(16) Results from the following observation. Fix $u \in \{r_1, \cdots, r_n\}$ and $v \in \{q_1, \cdots, q_n\}$. If $I^+(u,v) := \{j \in J | p_j - (u-r_j)^+ - (v-q_j)^+ > 0\}$ contains at most m elements, then the inner maximization in (15) is solved by taking for I the jobs with the m highest values for $p_j - (u-r_j)^+ - (v-q_j)^+$. Then $m(u+v) + \max_{I \subseteq J, |I|=m} \sum_{j \in I} p_j - (u-r_j)^+ - (v-q_j)^+$ is bounded from above by $m(r_{\max} + q_{\max}) + \max_{I \subseteq J, |I|=m} \sum_{j \in I} p_j - (r_{\max} - r_j)^+ - (q_{\max} - q_j)^+$, which equals $\max_{I \subseteq J, |I|=m} \sum_{j \in I} (p_j + r_j + q_j)$, so (15) $\leq$ (16).
Conversely, $\max_{I \subseteq J, |I|=m} \sum_{j \in I} (p_j + r_j + q_j)$ is bounded from above by $m(r_{\max} + q_{\max}) + \max_{I \subseteq J, |I| \geq m} \sum_{j \in I} p_j - (r_{\max} - r_j)^+ - (q_{\max} - q_j)^+$, hence (16) $\leq$ (15).

(17) Results from the observation that if $I^+(u,v)$ contains more than m elements, then the inner maximization in (15) is solved by taking $I = I^+(u,v)$.

A straightforward implementation of the computation of the latter maximum in (17) would result in an $\mathcal{O}(n^3)$ time algorithm. We next indicate how this maximum can be found in $\mathcal{O}(n^2)$ time. First, in $\mathcal{O}(n \log n)$ time, we establish the orderings of J with non-decreasing values for r_j, q_j, $p_j + q_j$, and $p_j + r_j + q_j$, respectively. This results, for instance, in a non-decreasing sequence $(q_{j_k})_{k=1}^n$. Next, we iterate over all values for $u \in \{r_1, \cdots, r_n\}$, and compute the following for each u, in order $\mathcal{O}(n)$ time.

1. By use of the aforementioned orderings we set up auxiliary arrays A_κ and A_λ giving, for each job J_j, the first index k for which $p_j + q_j - q_{j_k} \leq 0$, and the first index l for which $p_j + q_j + r_j - u - q_{j_l} \leq 0$, respectively.

2. Notice that, for fixed u, job J_j with $p_j - (u - r_j)^+ > 0$ contributes positively to the sum with constant term $p_j - (u - r_j)^+$ for $v = q_{\min}, \cdots, q_j$, and with an affine term $p_j - (u - r_j)^+ + q_j - v$ for $v = q_j, \cdots, q_{j_{k-1}}$, where k is the first index for which $p_j - (u - r_j)^+ + q_j - q_{j_k} \leq 0$, given by $k = \min\{A_\kappa(j), A_\lambda(j)\}$. The contribution of job J_j is depicted in Figure 2. To compute it, we only need the values of the breakpoints,

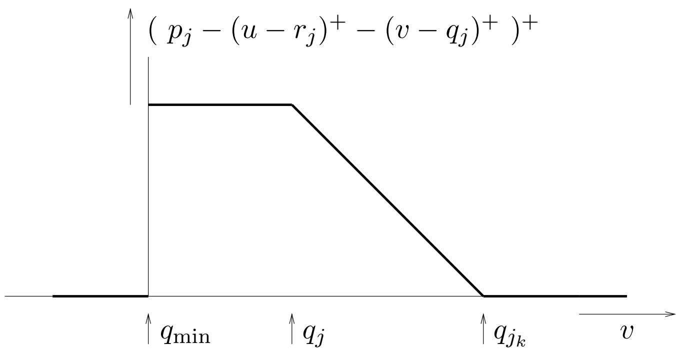

Figure 2: Contribution of job J_j, for fixed u.

which are equal to three values $v = q_{j_l}$, and the slope of the second piece. Therefore, for each job J_j we store, in constant time, the three indices l corresponding to the breakpoints, and the amount by which the contribution differs between $v = q_{j_{l-1}}$ and $v = q_{j_l}$.

3. By aggregating all these changes over all j, and then aggregating all changes over all $v = q_{j_l}$, for $l = 1, ..., n$, we find, in linear time, the sum of the contributions of all jobs to $\sum_{j \in J}(p_j - (u - r_j)^+ - (v - q_j)^+)^+$ for each v, as well as the number of jobs contributing positively to this sum.

Note that the number of machines m is only involved in the values, but never in the complexity of the algorithm. Hence, the complexity is truly $\mathcal{O}(n^2)$.

3.2 An efficient computation of the constrained set based bound

In this subsection we describe an algorithm to compute $SB(A, B, c)$. Recall that A is the set of jobs under consideration, B is the subset of A containing all jobs of which we cannot use both the head and the tail when computing the set based bound, and c is the number of machines. We assume that $2|A \setminus B| + |B| \geq 2c$, since the bound is invalid otherwise. Obviously, we need to select c of the r_j values and c of the q_j values. Since we will pick the smallest ones, subject to some constraints, we order the r_j and q_j values in non-increasing order, respectively, and put these in an ordered list; in case of a tie, we put the one with smaller index first. Given the c selected r_j and q_j values, we denote R and Q as the sets containing the corresponding jobs. Obviously, R and Q must minimize

$$\sum_{j:J_j \in R} r_j + \sum_{j \in A} p_j + \sum_{j:J_j \in Q} q_j, \tag{18}$$

subject to the constraints that $|R| = |Q| = c$ and that for all $J_j \in B$ we have that $J_j \notin (R \cap Q)$, that is, $B \cap R \cap Q = \emptyset$. Note that the total processing term is constant and therefore can be ignored.

Claim 1 There exists a combination of sets R and Q that minimize (18) with the following properties:

- If $J_l \in R$ and $J_k \notin R$, whereas r_k precedes r_l in the ordered list of r_j values, then $J_k \in B$ and $J_k \in Q$.
- If $J_l \in Q$ and $J_k \notin Q$, whereas q_k precedes q_l in the ordered list of q_j values, then $J_k \in B$ and $J_k \in R$.

Proof. Suppose to the contrary that R and Q do not satisfy this property: let J_k and J_l be a pair of jobs for which the first part of the property does not hold (we only prove it for the first part; the proof of the second part is similar). Obviously, since $r_k \leq r_l$, replacing R by $R \cup \{J_k\} \setminus \{J_l\}$ leads to a smaller or equal value, which move is feasible, since either $J_k \notin B$, or $J_k \notin Q$. Since the current choice of R and Q minimizes (18), replacing J_l by J_k in R must yield the same bound. We continue with the new R and Q, and after a finite number of updates we end up with two sets R and Q that satisfy the additional properties. ∎

We restrict ourselves from now on to sets R and Q that possess the properties specified in Claim 1. We will show below that it is easy to compute the best choice of R and Q *given that* *we use the jobs that occupy the first a and b positions in the list of r_j and q_j values, respectively*; we say that a job occupies a position in the list of r_j (q_j) values if its head (tail) occupies this position. We can then compute the sets R and Q that minimize (18) by finding the optimum R and Q for each relevant combination of (a, b) and then choose the best one. We denote the best choice of R and Q for a given (a, b) combination by $R(a, b)$ and $Q(a, b)$, respectively; we denote the corresponding value of this solution by $V(a, b)$. Not all combinations (a, b) are feasible of course. In the next two definitions we give a characterization for feasibility of (a, b).

Definition 1. A job J_k is *constrained* in (a, b) if $J_k \in B$ and if r_k and q_k occur in the first a and b positions in the r_j and q_j list, respectively, and unconstrained, otherwise. If no confusion is possible, then we skip the part 'in (a, b)'.

Definition 2. The combination (a, b) is *feasible*, if $a \geq c$, $b \geq c$, and if the number of constrained jobs is equal to $a + b - 2c$, and infeasible, otherwise.

It is easily verified that for any combination (a, b) the sets R and Q with the properties specified in Claim 1 and with $|R| = |Q| = c$ and $B \cap R \cap Q = \emptyset$ do exist only if the combination (a, b) is feasible. Given a feasible combination (a, b), we can determine $R(a, b)$ and $Q(a, b)$ as follows. We start by putting all jobs occupying the first a positions in the r_j list into $R(a, b)$; we fill $Q(a, b)$ with the unconstrained jobs that occupy the first b position in the q_j list. Now $R(a, b)$ contains $(a - c)$ jobs too many and $Q(a, b)$ contains $(a - c)$ jobs too few. Therefore, we must transfer $(a - c)$ of the constrained jobs that are currently in $R(a, b)$ to $Q(a, b)$. The transfer of job J_j from $R(a, b)$ to $Q(a, b)$ affects the value of the lower bound by $(q_j - r_j)$. Hence, we must transfer the $(a-c)$ constrained jobs with minimum $(q_j - r_j)$ value, after which we compute the outcome value. A straightforward implementation yields the constrained set based bound in $O(c^2 \log c)$ time, since the number of relevant (a, b) combinations is $O(c)$, as we will show below.

We now present a more efficient implementation that runs in $O(c \log c)$ time. We first present our algorithm that evaluates the relevant (a, b) combinations in an incremental fashion; at the end of this subsection we show that we have considered all relevant combinations.

We assume that we are given the sorted lists of r_j and q_j values, which are stored in two arrays; since we never need to consider more than $2c$ of the r_j and q_j values, respectively, the size of the array is $O(c)$. We first determine the value $\bar{a}$ such that the combination $(\bar{a}, c)$ is feasible and $(\bar{a}+1, c)$ is infeasible. For the combination $(\bar{a}, c)$ we can compute the sets $R(\bar{a}, c)$ and $Q(\bar{a}, c))$ in $O(c)$ time by picking the first c jobs from the q_j array for $Q(\bar{a}, c)$ and then select the first c jobs from the r_j array that can be included in $R(\bar{a}, c)$. Next, we determine for each job in $R(\bar{a}, c) \cup Q(\bar{a}, c)$ whether it is constrained or unconstrained in $(\bar{a}, c)$. We sort the $(\bar{a} - c)$ constrained jobs in order of nondecreasing $(q_j - r_j)$ value, where we use the value of the index as a tiebreaker, and put these in a heap with the maximum element in the root; this can be done in $O(c \log c)$ time. We use a heap, since this datastructure allows the possibility of adding or removing an element and bringing it back into the right shape with the maximum element in the root in $O(\log c)$ time. We always denote the maximum element in the heap by $(q_l - r_l)$, which corresponds to job J_l. Finally, we compute the value of this combination from (18).

Starting with the feasible combination $(\bar{a}, c)$, the corresponding heap, and the value of the bound $V(\bar{a}, c)$, we compute all feasible combinations (a, b) in an incremental fashion. Given (a, b) and the corresponding data, we find the maximum value $\hat{a}$, such that $(\hat{a}, b+1)$ is feasible, update the heap, compute the value $V(\hat{a}, b + 1)$, after which we put $a \leftarrow \hat{a}$, $b \leftarrow b + 1$, and continue, unless some stopping criterion is satisfied. The algorithm is as follows; we have added some explanation below.

Algorithm

Step 0. Put $a \leftarrow \bar{a}$, $b \leftarrow c$, $V(a,b) \leftarrow V(\bar{a},c)$.

Step 1. Store the combination (a,b) and the corresponding value $V(a,b)$. Put the value of the dummy variable V equal to $V(a,b)$. Let J_k be the job occupying position $b+1$ in the q_j array. Check if J_k is constrained in $(a,b+1)$. If it is unconstrained, then go to Step 3.

Step 2. Compute the value $(q_k - r_k)$. If J_k succeeds J_l in the $(q_j - r_j)$ order, then put $V(a,b+1) \leftarrow V(a,b)$, put $b \leftarrow b+1$, and go to Step 1. Otherwise, we update the heap by replacing $(q_l - r_l)$ by $(q_k - r_k)$. Put $V(a,b+1) \leftarrow V(a,b) - (q_l - r_l) + (q_k - r_k)$, put $b \leftarrow b+1$, and go to Step 1.

Step 3. Let J_i be the job that is currently occupying position a in the r_j array. If J_i is constrained in $(a,b+1)$, then go to Step 4; go to Step 5, otherwise.

Step 4. Compute $(q_i - r_i)$. If J_i preceeds J_l in the $(q_j - r_j)$ order, then we remove $(q_i - r_i)$ from the heap. If J_i succeeds J_l in the $(q_j - r_j)$ order, then we remove $(q_l - r_l)$ from the heap, and we put $V \leftarrow V - (q_l - r_l) + (q_i - r_i)$. Put $a \leftarrow a - 1$. If $a > c$, then go to Step 3; if $a = c$, then stop.

Step 5. Remove $(q_l - r_l)$ from the heap, put $a \leftarrow a-1$, put $b \leftarrow b+1$, and put $V(a,b) \leftarrow V - (q_l - r_l) - r_i + q_k$. Go to Step 1.

After we have run the algorithm, we have found all relevant feasible combinations with their values, as we will show below. Obviously, when we change something in the heap, then the properties of the heap get restored, and the current maximum element, which is always denoted by $(q_l - r_l)$, is put in the root. Now we give a short explanation of the algorithm. In Step 1 we check whether J_k is constrained in $(a,b+1)$. If this is the case, then $(a,b+1)$ is feasible, and we get the additional option of using q_k instead of r_k, the consequences of which are carried out in Step 2. If J_k is unconstrained in $(a,b+1)$, then $(a,b+1)$ is infeasible, and to restore feasibility, we have to remove jobs from the r_j array until we have removed one unconstrained job; this is carried out in Step 4 and Step 5. We stop in Step 4 if $a = c$, since we need to remove at least one more job from the r_j array, and this clearly cannot lead to a feasible combination anymore.

The time needed to update the values is $O(\log c)$ in each step. Since in each step we either decrease a by one or increase b by one, the number of updates is $O(c)$, and therefore the implementation above requires $O(c \log c)$ time.

What is left to prove is that we do consider each one of the relevant combinations (a,b). For this, we need the following property.

Claim 2. If the algorithm finds the feasible combination (a,b), then the combination $(a+1,b)$ and all combinations (a',b') with $a' \geq a+1$ and $b' \geq b$ are infeasible.

Proof. We start with the second part. Since the combination (a,b) is feasible and $(a+1,b)$ is not, the number of constrained jobs in $(a+1,b)$ must be too small to be feasible. A simple counting argument shows then that there are too few constrained jobs in (a',b') as well, for any $a' \geq a+1$ and $b' \geq b$.

Suppose to the contrary that the first part of the property does not hold; let (a,b) be the combination with minimum b value found by the algorithm such that $(a+1,b)$ is feasible. Because of the construction of the combination $(\bar{a},c)$, we have that $b > c$. Let $(\hat{a}, b-1)$

be the previous feasible combination found by the algorithm. Since $(\hat{a}, b-1)$ is feasible and $(\hat{a}+1, b-1)$ is not feasible, we can apply the second part of the claim, from which we find that (a', b) is infeasible, for any value $a' \geq \hat{a}+1$. Since $(a+1, b)$ is feasible, $a < \hat{a}$. But this implies that the algorithm has come by the combination $(a+1, b)$ and found that it was infeasible. ■

Theorem 1. The algorithm described above finds at least one of the combinations (a, b) that minimize (18).

Proof. Suppose that the minimum is attained by the combination (a', b') with $c \leq a' \leq \bar{a}$, which we overlooked. Since we consider the whole range of b values that lead to a feasible combination (a, b), our algorithm must have found some feasible combination $(\hat{a}, b')$. If $\hat{a} < a'$, then Claim 2 implies that the combination (a', b') is not feasible. Therefore, we must have that $\hat{a} > a'$. Since both combinations (a', b') and $(\hat{a}, b')$ are feasible, all jobs occupying a position k in the r_j array with $a' < k \leq \hat{a}$ must be constrained in $(\hat{a}, b')$, which implies that we are given more freedom to solve the problem for $(\hat{a}, b')$ than for (a', b'), and hence the value corresponding to (a', b') cannot be smaller than the value found for $(\hat{a}, b')$. ■

3.3 Approximating the preemptive bound

In this section we discuss some issues involved in computing the preemptive bound PB. Labetoulle, Lawler, Lenstra, and Rinnooy Kan (1984) present a polynomial time algorithm to solve the preemptive problem that is based on Horn's maximum flow algorithm (Horn, 1974), which tests whether there exists a feasible schedule for this problem with makespan no more than a given trial value. Labetoulle et al. show that the outcome of the preemptive problem can be determined by applying a bisection procedure that makes at most $\min\{n^2, \log(np_{\max})\}$ calls of Horn's algorithm, which needs $O(n^3)$ per check.

We show that there exist upper and lower bounds on PB that differ at most $\frac{m-1}{m}p_{\max}$, which implies that we need to apply Horn's algorithm at most $\lceil \log p_{\max} \rceil$ times in a bisection procedure to find $\lceil PB \rceil$. As a lower bound we use the maximum of the job based bound and the subset based bound, which is computed in $O(n^2)$ time. As an upper bound we use the length of the preemptive schedule that we get when we schedule the jobs according to the rule that we select among the available jobs the ones with highest *priority*, where the priority of a job is equal to the *sum of the tail and the remaining processing time*. This rule works as follows. Suppose that there are k unscheduled jobs available that all have maximum priority among the unscheduled available jobs. If the number of idle processors amounts to $m' < k$, then we assign to each of the k jobs a fraction m'/k of a machine; this is known as *processor sharing*. If $m' \geq k$, then we assign to each of the k jobs an entire machine and we assign jobs to the remaining $m' - k$ machines in the same fashion. The assignment of the machines to the jobs remains unchanged, until some *revision moment* occurs; this can be either when a new job is released, when the processing of a job is completed, or when jobs get equal priority. Note that each type of revision moment can occur at most n times. Finally, we convert the parts of the schedule in which processor sharing takes place into a feasible preemptive schedule of equal length; this is achieved by assigning for each interval between two consecutive revision moments the portions of the jobs that have been processed in the interval to the machines through McNaughton's wrap-around rule (McNaughton, 1959).

We will show in Theorem 2 that the length of the schedule produced by the heuristic, which we will denote by $D^H_{\max}$, is at most equal to $\max\{JB, SSB\} + \frac{m-1}{m}p_{\max}$. Since $D^H_{\max}$ is equal to the length of a feasible preemptive schedule, and hence is an upper bound on PB, we therefore then have identified an upper and a lower bound on PB that are no more than $\frac{m-1}{m}p_{\max}$ apart. Our proof is based on the concept of identifying a *smallest counterexample*: if there exists an instance for which our heuristic produces a schedule with length more than the claimed upper bound, then there must exist a smallest instance with this property, where the notion of smallest refers to the number of jobs and, in case of an equal number of jobs, to the number of revision moments. We start with some observations to characterize such a smallest counterexample, after which we prove that this alleged counterexample is not a counterexample at all.

Throughout this subsection, we let $\mathcal{I}$ and σ denote the instance of the smallest counterexample and the 'schedule' that is determined by the heuristic before applying the wrap-around rule, respectively. We further let J_j denote the *critical* job that is completed first in σ, where a critical job is defined as a job with delivery time equal to $D^H_{\max}$.

Observation 1. There is no job in σ that is processed after time T, where T is the completion time of J_j.

Proof. The proof is based on identifying a smaller counterexample if there is processing after time T. First, note that deleting the jobs started after time T does not influence the schedule up to time T, even if these jobs were available before T. Let J_i be any job started before time T that has not finished processing at time T; let π_i denote the remaining processing time of J_i at time T. Decreasing p_i by π_i and increasing q_i by π_i yields an instance that leads to a schedule that is identical to σ up to time T. If we remove all jobs that are processed after time T in this way, then we obtain an instance $\mathcal{I}'$ with no more jobs that needs at least one revision moment less, which implies that it is a smaller instance. We will now show that $D^H_{\max}(\mathcal{I}') > \max\{JB(\mathcal{I}'), SSB(\mathcal{I}')\} + \frac{m-1}{m}p_{\max}(\mathcal{I}')$, which disqualifies $\mathcal{I}$ as the smallest instance with this property. As the schedules for $\mathcal{I}$ and $\mathcal{I}'$ are identical up to time T, we have that in both cases J_j is delivered at time $T + q_j$, which implies that $D^H_{\max}(\mathcal{I}') \geq D^H_{\max}(\mathcal{I})$. Since we did not increase the processing time and the priority of any job, $p_{\max}(\mathcal{I}') \leq p_{\max}(\mathcal{I})$ and $JB(\mathcal{I}') \leq JB(\mathcal{I})$. Moreover, equation (13) implies that $SSB(\mathcal{I}') \leq SSB(\mathcal{I})$, as $(p'_j - (u-r_j)^+ - (v-q'_j)^+) \leq (p_j - (u-r_j)^+ - (v-q_j)^+)$, where p'_j and q'_j denote the adjusted processing times and tails. Hence,

$$\begin{aligned} D^H_{\max}(\mathcal{I}') \geq D^H_{\max}(\mathcal{I}) \quad &> \quad \max\{JB(\mathcal{I}), SSB(\mathcal{I})\} + \frac{m-1}{m}p_{\max}(\mathcal{I}) \\ &\geq \quad \max\{JB(\mathcal{I}'), SSB(\mathcal{I}')\} + \frac{m-1}{m}p_{\max}(\mathcal{I}'). \end{aligned} \qquad \blacksquare$$

Observation 2. Let t denote the last revision moment in σ before time T. During the interval $[t, T]$, J_j does not have an entire machine assigned to it, and all jobs processed in $[t, T]$ have equal tails.

Proof. Since by assumption $D^H_{\max}(\mathcal{I}) > JB(\mathcal{I}) \geq r_j + p_j + q_j$, there is a time interval during which J_j does not have an entire machine. Let $[t_1, t_2]$ denote the last time interval during which J_j does not have a own machine of its own, where t_1 and t_2 are two consecutive revision moments. We know that in $[t_1, t_2]$ no machine is idle, that no job with lower priority than J_j is processed, and that at least m other jobs with equal or higher priority are processed. We first show that $t_2 = T$. Suppose to the contrary that $t_2 < T$. As J_j gets its own machine at time t_2 although at least m other jobs have at least equal priority at time t_2, at least one of these m jobs, say J_i, must have completion time equal to t_2 and $D_i(\sigma) = t_2 + q_i$. Since J_j gets fully processed after time t_2, $D_j(\sigma)$ is equal to t_2 plus the priority of J_j at time t_2. Since the priority of J_i is at least equal to the priority of J_j at time t_2, J_i is delivered at or after time $D_j(\sigma) = D^H_{\max}(\mathcal{I})$, which implies that it is a critical job that is completed before J_j. This contradicts the definition of J_j, from which we conclude that $t_2 = T$. As no job is processed after time T, each job processed in $[t_1, t_2]$ is delivered at or after time $D_j(\sigma)$, since the priority of these jobs is at least equal to the priority of J_j. As J_j is a critical job, the other jobs processed in $[t_1, t_2]$ must be delivered at time $D_j(\sigma)$ too, which implies that all these jobs must have tails equal to q_j. ■

As a direct consequence of Observation 2, we know that at time T all machines are occupied by jobs with equal tails. Define t_0 as the earliest revision moment such that during the time interval $[t_0, T]$ all machines are occupied and no job with smaller tail than q_j is processed; because of Observation 1, we know that no job with tail smaller than q_j is available in $[t_0, T]$. We further define t_{-1} as the last revision moment before t_0, where $t_{-1} = 0$ if no revision takes place before time t_0. Note that $t_0 > 0$, as otherwise we have that $SB(\mathcal{I}) = T + q_j = D_j(\sigma) = D^H_{\max}(\mathcal{I})$, which is not the kind of property we want our counterexample to possess. It follows immediately from the definition of t_0 that at most $m-1$ jobs with tails greater than or equal to q_j are available between time t_{-1} and t_0, which implies that at most $m-1$ of the jobs processed before time t_0 are also processed after time t_0. We 'cut open' the instance at time t_0. Let $\mathcal{I}_1$ and $\mathcal{I}_2$ denote the instances containing all jobs that are processed before and after time t_0, respectively. Define S as the set containing all jobs that appear in both $\mathcal{I}_1$ and $\mathcal{I}_2$; $|S| \leq m-1$. Note that none of the jobs in $\mathcal{I}_2 \setminus S$ is available before time t_0, since they are not processed before time t_0: there are at most $m-1$ jobs available in $[t_{-1}, t_0]$ with tail greater than or equal to q_j, and the priority of any other job (if it exists) drops below q_j somewhere before time t_0, since they are completed at time t_0.

Observation 3. $\mathcal{I}_2$ contains at most $n-1$ jobs.

Proof. Suppose to the contrary that $\mathcal{I}_2$ contains all n jobs, that is, no job is completed at or before time t_0. Then the jobs with the m smallest release dates are processed continuously from their respective release dates until time t_0, and all m machines are busy from time t_0 to time T, at which the jobs with the m smallest tails are completed. Hence, $SB(\mathcal{I}) = T + q_j = D_j(\sigma)$, and we do not have a counterexample. ■

We adjust the instance $\mathcal{I}_2$ by redefining the release dates and processing times of the jobs in S. All jobs J_i in S get a release date equal to t_0 and a processing time equal to the processing time remaining at time t_0. We denote this instance by $\mathcal{I}'_2$. Since the jobs in $\mathcal{I}'_2 \setminus S$ are not available at time t_{-1} and since there is no revision moment between time t_{-1} and t_0, all jobs in $\mathcal{I}'_2$ are released at or after time t_0. By definition of t_0, the jobs in $\mathcal{I}'_2$ keep all machines occupied from time t_0 until time T.

Theorem 2. For any instance, the length $D^H_{\max}$ of the schedule produced by the heuristic satisfies $PB \leq D^H_{\max} \leq \max\{JB, SSB\} + \frac{m-1}{m} p_{\max}$.

Proof. As we have already mentioned, $PB \leq D^H_{\max}$. What remains to be shown is that $D^H_{\max} \leq \max\{JB, SSB\} + \frac{m-1}{m} p_{\max}$. Suppose to the contrary that there exists an instance for which the above statement is not satisfied; then there also exists a smallest counterexample $\mathcal{I}$ to the above statement. On basis of the Observations 1 through 3, we arrive at the instances $\mathcal{I}_1$ and $\mathcal{I}'_2$. We apply the heuristic to the instances $\mathcal{I}_1$ and $\mathcal{I}'_2$ separately; let σ_1 and σ_2 and $D^H_{\max}(\mathcal{I}_1)$ and $D^H_{\max}(\mathcal{I}'_2)$ denote the corresponding schedules and the corresponding solution values. We have that σ_1 and σ are identical up to time t_0; similarly, σ_2 and σ are identical from time t_0 onwards. Since J_j is completed at time $T > t_0$, we have that $D^H_{\max}(\mathcal{I}) = D^H_{\max}(\mathcal{I}'_2) = SB(\mathcal{I}'_2)$, as all machines are continuously busy processing from time t_0, at which the first m jobs get released, until time T, at which the m jobs with the smallest tails are completed.

We cannot straightforwardly relate $SB(\mathcal{I}'_2)$ to $SSB(\mathcal{I})$, because of the redefinition of the release dates of the jobs in S. We therefore delete the jobs in S from $\mathcal{I}'_2$: let $\mathcal{I}_3$ denote the remaining instance. If $\mathcal{I}_3$ contains at least m jobs, then $SB(\mathcal{I}_3)$ is well defined and

$$\begin{aligned} D^H_{\max}(\mathcal{I}) = SB(\mathcal{I}'_2) &\leq SB(\mathcal{I}_3) + \frac{1}{m}p(S) \leq SB(\mathcal{I}_3) + \frac{m-1}{m}p_{\max}(\mathcal{I}) \\ &\leq SSB(\mathcal{I}) + \frac{m-1}{m}p_{\max}(\mathcal{I}), \end{aligned}$$

where the second inequality follows from $|S| \leq m-1$. Now suppose that $\mathcal{I}_3$ contains at most $m-1$ jobs. Since all jobs from $\mathcal{I}'_2$ are processed between times t_0 and T, we have that $m(T-t_0) = p(\mathcal{I}'_2) = p(S) + p(\mathcal{I}_3)$. Let J_k be a job with maximum processing time from the jobs in $\mathcal{I}_3$. By definition of $\mathcal{I}_3$, we have $r_k \geq t_0$ and $q_k \geq q_j$; we further have $p_k > p(\mathcal{I}_3)/m$. Filling this in, we get

$$\begin{aligned} D^H_{\max}(\mathcal{I}) = T + q_j &= t_0 + \frac{1}{m}p(S) + \frac{1}{m}p(\mathcal{I}_3) + q_j \\ &\leq \frac{m-1}{m}p_{\max} + r_k + p_k + q_k, \end{aligned}$$

from which it immediately follows that $D^H_{\max}(\mathcal{I}) \leq JB(\mathcal{I}) + \frac{m-1}{m}p_{\max}$, so that $\mathcal{I}$ is not a counterexample. ■

3.4 Another algorithm to compute max{$\boldsymbol{JB}, \boldsymbol{SSB}$}

In this section we discuss another algorithm to determine the value of $\max\{JB, SSB\}$. This algorithm has been proposed by Carlier and Pinson (1998), who state that it can be implemented to run in $O(n \log n + nm \log m)$ time using appropriate data structures. Carlier and Pinson show that the outcome value is equal to $\max\{JB, SSB\}$, but their proof is quite complicated. We therefore present our own proof of this result in this subsection. We use the algorithm in the next subsection.

The idea behind the algorithm is to consider a relaxation of the preemptive parallel machine scheduling problem in which a job can be processed simultaneously by more than one processor with the restriction that at any point in time t no more than $t - r_j$ units of the body of J_j ($j = 1, \ldots, n$) have been executed; if exactly $t - r_j$ units of job J_j have been processed at time t, then it is called *blocked* at time t. As a consequence, J_j can never be completed before time $r_j + p_j$, and therefore the maximum delivery time in such a 'schedule' is always bounded from below by JB. We use the notion of *unblocked* to refer to the remaining jobs that are processed by the machines. At any point in time, the idle machines are assigned to the available jobs with maximum priority, where the priority of a job is defined as the sum of the tail and the remaining processing time. Let k be the number of jobs with maximum priority, and let m' denote the number of machines that have not yet been assigned to the jobs. If $m' \leq k$, then we assign a fraction m'/k of a machine to each of the k jobs; if $m' > k$, then we assign an entire machine to each of the blocked jobs, and we assign the remaining machines to the unblocked jobs, if any. If all k jobs are blocked, then we assign the remaining $m' - k$ machines to the remaining jobs in a similar fashion. Note the similarity to the rule used in Subsection 3.3; the only difference is that we now may assign more than one machine to a job.

The assignment of machines to jobs has to be adjusted at the revision moments. A revision moment occurs when a new job becomes available, when a job is completed, when two jobs get the same priority, and when a job becomes blocked. We first show that the number of revision moments is bounded; we need this observation later on.

The first two types of events, which we call *major events*, can occur at most $2n$ times in total. We need a bound on the number of revision moments that can occur between two successive major events. Let l denote the number of jobs that are being processed. We first observe that if $l < m$, then more than one machine is assigned to each of the unblocked jobs; if $l \geq m$, then at most one machine is assigned to each job. In the latter case, no job can get blocked. Since two jobs with equal priority keep the same priority until one of them becomes blocked or is completed, we conclude that l cannot decrease between two major events.

First, we bound the number of revision moments for the case that $l < m$. Because of the assignment strategy, the priority of an unblocked job decreases at a faster rate than the priority of a blocked job. Therefore, we only have to revise the assignment if one of the unblocked jobs becomes blocked or if the priority of the unblocked jobs becomes equal to the priority of an unprocessed job. Hence, after at most $2m$ revision moments we arrive at the case that $l \geq m$. Moreover, at most $m - 1$ jobs have become blocked in the meantime. In case $l \geq m$, the priority of the blocked jobs decreases at least as fast as the priority of the unblocked jobs, and hence a revision moment can occur only if the priority of the unblocked jobs becomes equal, either to the priority of a blocked job, or to the priority of an unprocessed job. No job gets blocked in this case.

At most $m - 1$ jobs get blocked between two major events, which implies that at most $2n(m - 1)$ times a job gets blocked. Together with the major events, this leads to $2mn$ possible moments at which jobs of equal priority are separated. Hence, there can be at most $2mn$ moments at which two jobs obtain equal priority. We conclude that the total number of revision moments is bounded by $4mn$. The $O(nm)$ bound can be shown to be tight.

Lemma 1. The 'schedule' σ obtained by applying Carlier and Pinson's rule has length greater than or equal to $\max\{JB, SSB\}$.

Proof. Due to the blocking constraint, we know that the length of σ, which we denote by $D^C_{\max}(\sigma)$, cannot be less than JB. Hence, we must show that $D^C_{\max}(\sigma) \geq SSB$ as well. Let $\mathcal{S}$ denote the subset of the jobs that maximizes SSB. Since Carlier and Pinson's rule does not yield a longer schedule when applied to a subset of the jobs, we are done if we show that Carlier and Pinson's rule creates a schedule σ' for the instance $\mathcal{S}$ with $D^C_{\max}(\sigma') \geq SB(\mathcal{S})$.

To show this, we show that the total amount of idle time in σ' in the interval $[0, D^C_{\max}(\sigma')]$ amounts to at least $\bar{r}_1 + \ldots + \bar{r}_m + \bar{q}_1 + \ldots + \bar{q}_m$; the desired inequality then immediately follows, as this total amount of idle time is equal to $mD^C_{\max}(\sigma') - p(S)$. We first look at the front of σ': because of the blocking restriction, we have at least $\bar{r}_1 + \ldots + \bar{r}_m$ units of idle time at the start of σ'. Now we look at the rear end of σ'. Define T as the moment in σ' at which the $(n - m)$th job, say J_j, is completed; at time T, there are at most m jobs left that have to finish their completion. We need a lower bound on the amount of idle time in σ' in the interval $[T, D^C_{\max}(\sigma')]$. For any J_i that is blocked at time T or is released at or after time T, we have that the processing time remaining at time T can amount to no more than

$D_{\max}^C(\sigma') - T - q_i$, since it would get completed too late otherwise. We further know that all the others of these m jobs have priority at most equal to q_j at time T, as J_j is completed at time T. Since $q_j \leq D_{\max}^C(\sigma') - T$, the processing time that remains at time T of each of the last m jobs J_i is at most equal to $q_j - q_i \leq D_{\max}^C(\sigma') - T - q_i$. Therefore, the total amount of idle time at the end of the schedule is at least equal to the sum of these m tails, which is obviously bounded from below by $\bar{q}_1 + \ldots + \bar{q}_m$. ∎

What remains to be shown is that the length of Carlier and Pinson's schedule is bounded from *above* by $\max\{JB, SSB\}$ as well. We will show this result by using the concept of a smallest counterexample, where the notion of smallest is with respect to the number of jobs and to the number of revision moments in case of an equal number of jobs. Suppose that there exists an instance for which Carlier and Pinson's algorithm produces a 'schedule' with length $D_{\max}^C > \max\{JB, SSB\}$; let $\mathcal{I}$ denote the smallest such instance and let σ denote the 'schedule' determined by Carlier and Pinson's algorithm. We start with some observations concerning $\mathcal{I}$ and σ.

Observation 4. At some point in time, all machines are occupied in σ.

Proof. Suppose to the contrary that there is no point in time at which all machines in σ are used for processing jobs. This implies that there are no unblocked jobs in σ. Hence, each J_j is completed at time $r_j + p_j$, which implies that $D_{\max}^C(\sigma) = JB$. ∎

Observation 5. Let t_1 and t_2 denote the first and last time in σ at which all m machines are assigned to jobs. Then none of the machines is idle in σ during the interval $[t_1, t_2]$.

Proof. Suppose that at some time $t \in [t_1, t_2]$ there is a machine idle in σ. This implies that there are no unblocked jobs at time t. We create instances $\mathcal{I}_1$ and $\mathcal{I}_2$ as follows. $\mathcal{I}_1$ contains all jobs that are released before time t; the heads, bodies, and tails all remain the same. $\mathcal{I}_2$ contains all jobs that have not completed processing at time t; for any $J_j \in \mathcal{I}_2$, r_j is set equal to $\max\{r_j, t\}$, p_j is set equal to the remaining processing time at time t, and q_j is left unchanged. Note that both $\mathcal{I}_1$ and $\mathcal{I}_2$ contain at least one job less than $\mathcal{I}$ and that the intersection of $\mathcal{I}_1$ and $\mathcal{I}_2$ does not have to be empty. Denote the 'schedules' determined through Carlier and Pinson's algorithm for these instances by σ_1 and σ_2. The 'schedule' σ_1 is equal to σ until time t, after which each job in $\mathcal{I}_1$ is blocked and gets its own machine; σ_2 and σ are identical from time t onwards, as all jobs processed at time t are blocked in σ. Hence,

$$\begin{aligned} \max\{D_{\max}^C(\sigma_1), D_{\max}^C(\sigma_2)\} = D_{\max}^C(\sigma) &> \max\{JB(\mathcal{I}), SSB(\mathcal{I})\} \\ &\geq \max\{JB(\mathcal{I}_1), SSB(\mathcal{I}_1), JB(\mathcal{I}_2), SSB(\mathcal{I}_2)\}, \end{aligned}$$

which implies that at least one of $\mathcal{I}_1$ and $\mathcal{I}_2$ constitutes a smaller counterexample. ∎

Observation 6. Let again t_1 be the first time in σ at which all machines are occupied. Then none of the jobs is released before time t_1.

Proof. Suppose to the contrary that there is a job J_j with $r_j < t_1$. First suppose that $r_j + p_j \geq t_1$. Since t_1 is the first time at which all machines are busy, Carlier and Pinson's algorithm assigns exactly one machine to J_j from time r_j to time t_1. Define $\mathcal{I}'$ as the instance obtained from $\mathcal{I}$ by setting r_j equal to t_1 and by decreasing p_j by $t_1 - r_j$ units. Then Carlier and Pinson's rule will come up with a schedule that is equal to σ except for the processing of J_j from time r_j to time t_1. Moreover, since the total of the head and the body remains unchanged, both JB and SSB do not increase. Hence, $\mathcal{I}'$ constitutes a counterexample as well. Since the number of jobs in $\mathcal{I}$ and $\mathcal{I}'$ is the same and the number of revision moments has decreased, $\mathcal{I}'$ is a smaller counterexample, which contradicts the definition of $\mathcal{I}$.

Second, suppose that $r_j + p_j < t_1$. Define $\mathcal{I}'$ as the instance obtained from $\mathcal{I}$ by removing J_j. The 'schedule' determined through Carlier and Pinson's rule is equal to σ except for the execution of J_j, and the length of this 'schedule' will not decrease, since $D_j^C(\sigma) = r_j + p_j + q_j \le JB < D_{\max}^C(\sigma)$. It is readily shown that in this case $\max\{JB(\mathcal{I}'), SSB(\mathcal{I}')\} \le \max\{JB(\mathcal{I}), SSB(\mathcal{I})\}$, which implies that $\mathcal{I}'$ would be a smaller counterexample. ∎

Because of Observation 6, we know that all machines start simultaneously. We will now show that all machines end processing at the same time. Let J_j be a job with $D_j^C(\sigma) = D_{\max}^C(\sigma)$; suppose that J_j is completed at time T. As J_j is not blocked, all machines are occupied at time T.

Observation 7. There is no job in σ that is processed after time T.

Proof. The proof is identical to the proof of Observation 1. In case there are jobs processed after time T, we can obtain a smaller counterexample. ∎

Observation 8. The tail q_j of J_j is minimum among the tails of the jobs in $\mathcal{I}$.

Proof. Suppose to the contrary that q_j is not minimum. Let J_k be a job with minimum tail; suppose that J_k is completed at time $\hat{T}$. Because of the assignment strategy, all other jobs that are processed at time $\hat{T}$ must be either blocked or have priority equal to the priority of J_k; the last case can only occur in case of a job with tail equal to q_k that is completed at time $\hat{T}$ as well. As $D_{\max}^C(\sigma) > JB$, J_j is not blocked at time T, which implies that $\hat{T} < T$. As all processed jobs are completed or blocked at time $\hat{T}$, we can split the instance at time $\hat{T}$, just as in the proof of Observation 5. Along the lines of the proof of Observation 5, we can show that instance $\mathcal{I}_2$ constitutes a smaller counterexample. ∎

By now, we have shown that $D_{\max}^C(\sigma) = T + q_j$. Moreover, we have shown that the total processing time of all jobs in $\mathcal{I}$ is $m(T - t_1)$ and that there are at least m jobs in $\mathcal{I}$ with minimum release date, which is equal to t_1. Since q_j is minimum among the tails of the jobs in $\mathcal{I}$, we have that

$$SB(\mathcal{I}) \ge t_1 + (T - t_1) + q_1 = T + q_1 = D_{\max}^C(\sigma).$$

Therefore, we have proved the following theorem.

Theorem 3. The 'schedule' obtained through Carlier and Pinson's rule for the relaxed preemptive parallel machine scheduling problem has length equal to $\max\{JB, SSB\}$. ∎

3.5 Tightness of the upper bound on *PB*

In Section 3.3 we have shown that the gap between PB and $\max\{JB, SSB\}$ is at most equal to $\frac{m-1}{m} p_{\max}$. In this section we present a family of instances to show that this upper bound on the difference is tight. We use Carlier and Pinson's rule to determine the value of $\max\{JB, SSB\}$ for these instances.

We use D to denote the maximum processing time, and we use ϵ to denote a small positive value. Given any value of m, D, and ϵ, we construct an instance for which $PB - \max\{JB, SSB\} \ge \frac{m-1}{m} p_{\max} - \epsilon$ as follows. Define $\delta = \frac{m-1}{m} D - \epsilon$, and define $N = \lceil \frac{m\delta}{\epsilon} \rceil + 1$. The instance consists of $N(2m-1)$ jobs, which are partitioned into N batches; all jobs have processing time equal to D. The ith batch consists of the jobs $J_1^i, \ldots, J_m^i$, which all have head $r_0^i = (i-1)\frac{2m-1}{m} D$ and tail $q_0^i = (N+1-i)\frac{2m-1}{m} D$, and the jobs $J_{m+1}^i, \ldots, J_{2m-1}^i$, each with head $r_1^i = r_0^i$, and tail $q_1^i = q_0^i - D$. Through Carlier and Pinson's priority rule, we find that for this instance $\max\{JB, SSB\} = N\frac{2m-1}{m} D + D$. We claim that there exists no preemptive schedule with length PB smaller than $N\frac{2m-1}{m} D + D + \delta$.

Suppose to the contrary, that such a schedule does exist. By definition of the tails and the heads we find that only the $m-1$ jobs $J^i_{m+1}, \ldots, J^i_{2m-1}$ are available in the interval $[PB - q^i_0, i\frac{2m-1}{m}D]$. Note that this interval has length $r^{i+1}_0 + q^i_0 - PB = (N+1)\frac{2m-1}{m}D - PB > \frac{m-1}{m}D - \delta = \epsilon$ and that the preemptive schedule leaves at least one machine idle in this period. Disregarding the last $m-1$ jobs of the last batch, one finds that the set based bound, including wasted idle time, is at least

$$\frac{1}{m}[(N(2m-1) - (m-1))D + (N-1)\epsilon + mq^N_0] =$$
$$\frac{1}{m}[(N(2m-1) + m)D + (N-1)\epsilon] \geq$$
$$\frac{1}{m}[(N(2m-1) + m)D + m\delta] > PB,$$

which is a contradiction.

4 Approximation algorithms

In this section we discuss heuristics for the non-preemptive head-body-tail problem and for the multiprocessor flow shop scheduling problem. The heuristics are simple, but may provide good starting points for local search.

Approximating $P|r_j, q_j|D_{\max}$. The non-preemptive longest tail rule is a commonly used approximation algorithm for $P|r_j, q_j|D_{\max}$. If at time t a machine becomes available, then from the available jobs the one with largest tail is scheduled. If no job is available, then t is set to the next release date. In case of parallel machines, the longest tail rule may schedule a job with small tail but with huge processing time too late. We therefore propose two new selection criteria:

- Select the job with maximum value of $q_j + \frac{a}{n}p_j$, where n is the total number of jobs and a the number of already scheduled jobs.
- Use the longest tail rule, but give priority to an available job with largest $q_j + p_j$ if it would be completed after the estimated length of the optimal schedule at the next scheduling opportunity. For the estimated length of the optimal schedule we use the lower bound $\max\{JB, SSB\}$.

The parallel machine problem is symmetric in the release dates and delivery times. We therefore can apply the approximation algorithms to the problem where the release date and delivery time of each job are interchanged. Reversing the resulting schedule then yields a right justified schedule for the original data.

We can solve the parallel machine problem by means of branch-and-bound. Carlier (1987) proposes a branch-and-bound algorithm for this problem. In our implementation we do not use the lower and upper bounds proposed by Carlier, but a mixture of our bounds, described in Section 2. The higher effort in calculating the bounds in each node results in a smaller search tree and a faster algorithm. We limit the depth of the tree to 18 nodes. The optimal solution value, or the minimum lower bound in any leaf of the search tree will be referred to as the *non-preemptive bound NPB*.

Approximating $F(P)||C_{\max}$. For each stage in the multiprocessor flow shop we can set up a corresponding parallel machine problem. Each lower bound for the parallel machine problem of one stage is a lower bound for the multiprocessor flow shop. To obtain the bound for $F(P)||C_{\max}$ we maximize over all lower bounds for the stages.

We used two constructive algorithms to obtain upper bounds on $F(P)||C_{\max}$. Except for a lower bound the branch-and-bound algorithm also provides a non-preemptive schedule for one parallel machine problem. To make sure that schedules of different stages do not overlap, we can adjust the heads or tails of the jobs. In the first algorithm *up-stage* we solve the parallel machine problem in stage 1. Next we solve the parallel machine problem in stage 2 where we use the completion times in stage 1 as heads. Repeating this strategy for all subsequent stages a non-preemptive schedule for the multiprocessor flow shop problem is obtained. The second algorithm *down-stage* algorithm uses the same strategy in reverse order.

5 Computational results and conclusions

Test problems. To test the lower and upper bounds, we generated random test problems. For the number of stages and the number of machines per stage, we used the patterns $(m_1, m_2, \ldots, m_l)$ as shown in Table 1.

Table 1: Stage-machine configurations.

number	2 stages	4 stages	6 stages	8 stages	10 stages
1	2-2	2-2-2-2	2-2-2-2-2-2	2-2-2-2-2-2-2-2	2-2-2-2-2-2-2-2-2-2
2	1-2	2-4-4-6	1-2-3-4-5-6	1-1-2-2-3-3-4-4	1-1-2-2-3-3-4-4-5-5
3	1-4	2-4-2-4	1-2-3-1-2-3	1-3-1-3-1-3-1-3	1-2-3-4-5-1-2-3-4-5
4	3-5	2-3-4-2	1-2-4-4-2-1	1-2-3-4-1-2-3-4	2-2-3-3-4-4-3-3-2-2
5		3-1-2-3	5-5-1-1-5-5	1-2-3-4-4-3-2-1	5-4-3-2-1-1-2-3-4-5
6			4-2-1-1-2-4	5-4-3-2-2-3-4-5	1-2-4-2-1-3-4-4-2-2
7				1-3-2-3-1-4-2-3	5-4-3-2-3-4-5-2-3-5
8					1-3-2-4-1-3-2-4-1-4

We test our bounds on instances with 10, 20, 40, and 80 jobs. The processing times are chosen so as to create three different types of problems.

Type 1 The processing times of all operations are drawn from the discrete uniform distribution on $[10, 30]$.

Type 2 The processing times in stage h depend on the number of machines m_h in this stage. The processing times of the operations to be executed in stage h are drawn from the discrete uniform distribution on $[2m_h, 4m_h]$ to create problems with an equal workload per stage.

Type 3 The processing times of all operations are drawn from the discrete uniform distribution on $[5, 15]$ with probability 0.9 and from the discrete uniform distribution on $[50, 70]$ with probability 0.1.

For each stage-machine configuration, number of jobs, and type of processing time distribution, we generated five instances. This resulted in 1,800 multiprocessor and 12,000 parallel machine problems.

Results. An overview of the test results of the lower bound on both the parallel machine and the multiprocessor flow shop problems is given in Figure 3. As in both problems preemption is not allowed and all processing times are integers we rounded up all values for the lower bounds to the nearest integer.

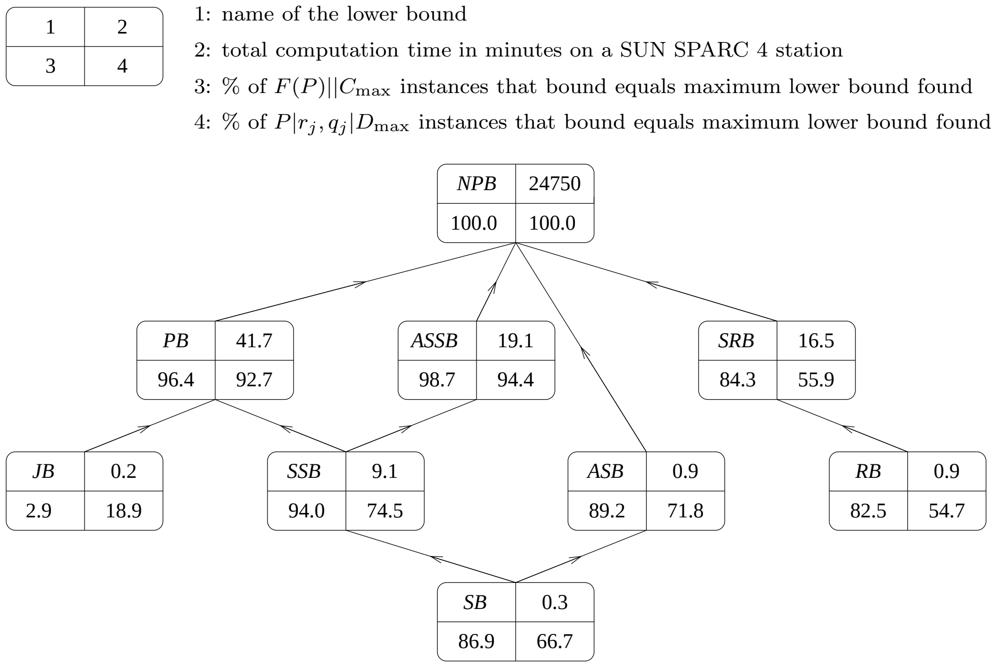

Figure 3: Performance scheme of the lower bounds.

Additional information:

- For all our test instances, $\max\{JB, \lceil SSB \rceil\} = \lceil PB \rceil$. Instances exist for which equality does not hold.

- The branch-and-bound algorithm leaves 3.6% of the parallel machine problems unsolved for the used maximum level of 18 nodes. 81% of the unsolved problems are instances with 10 jobs.

- In 54% of the parallel machine instances with $m > 1$ we obtained a better upper bound in the root node than the standard longest tail rule by using the two new selection criteria on both the original and the reverse problem.

For most parallel machine instances the minimum makespan equals one of the lower bounds. The optimal schedules for the multiprocessor flow shop instances are unknown. To see whether a good lower bound can be obtained by using the relaxation to a parallel machine problem, we compare the lower and upper bounds. We use the deviation Δ defined as

$$\Delta = \text{best of up-stage and down-stage upper bound}/NPB - 1 \tag{19}$$

as performance measure. We sort the deviations of all instances of the same type of processing time distribution and show the corresponding plots in Figure 4.

The deviation depends on the upper and the lower bounds. The type 1 problems form easy instances. For the type 2 problems the lower bounds seem to be weak, which is due to the lack of a distinct bottleneck. The type 3 problems show erratic behavior: most of them are simple, but some are very hard.

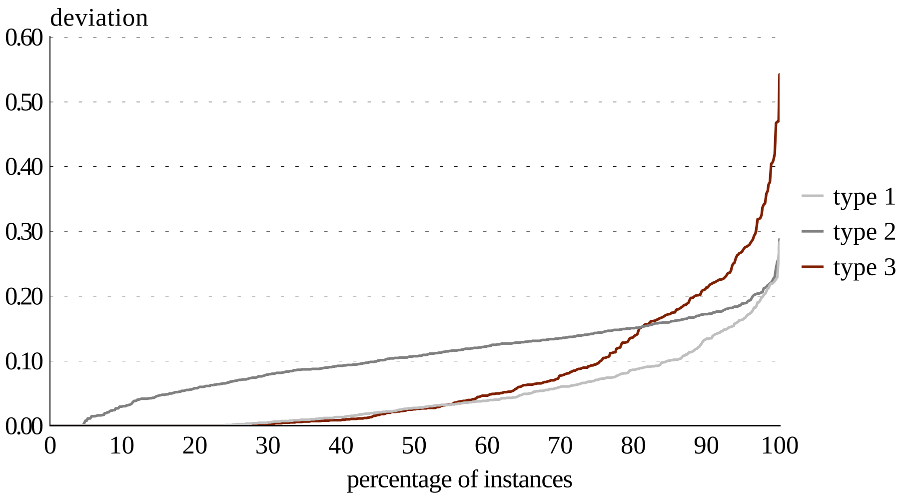

Figure 4: Deviations per type of processing time.

We have also tested the effect of the different patterns on the mean deviation. It turns out that the pattern does not influence the mean deviation in case of type 2 problems. For type 1 and type 3 problems we can draw the following conclusions.

- The all equal pattern has the highest mean deviation, since there is no clear bottleneck; especially the combination of a type 3 problem and an all equal pattern is hard.
- The mean deviation becomes higher if the number of times that an increase of the number of machines is followed by a decrease of the number of machines per stage is higher. Patterns with a consistent increase or decrease in the number of machines per stage are easy.
- The top patterns (like 1-2-4-4-2-1) is much harder than the valley pattern (like 4-2-1-1-2-4).

Conclusion. We have studied the quality of lower and upper bounds for $F(P)||C_{\max}$. The lower bounds are all based on the single stage relaxation to $P|r_j, q_j|D_{\max}$. We have given efficient implementations, which yield strong lower bounds in reasonable time. Solving problem $P|r_j, q_j|D_{\max}$ exactly by branch-and-bound contributes little and is time consuming. Moreover, it is not worth the effort to compute the preemptive bound, since it is almost always equal to the maximum of the job based bound and the subset based bound, which can be computed much more efficiently. Adjusting the set based bound and the subset based bound, however, can be done quickly and may give a substantial improvement. In case there is no distinct bottleneck stage the lower bounds seem to be weak. Using simple adjustments on the longest tail rule and by solving the reverse problem, we find good upper bounds for $F(P)||C_{\max}$. The proposed algorithms up-stage and down-stage can be used as good starting solutions for local search.

Acknowledgements

The authors thank an anonymous referee and an Associate Editor for their helpful comments. The second author was partially supported by EC Contract IST-1999-14186 (Project ALCOM-FT).

References

[1] P. Baptiste, C. Le Pape, and W. Nuijten (2001). *Constraint-based scheduling: Applying constraint programming to scheduling problems*. Kluwer, Boston.

[2] S.A. Brah and J.L. Hunsucker (1991). Branch-and-bound algorithms for the flow shop with multiple processors. *European Journal of Operational Research 51*, 88-99.

[3] R.E. Buten and V.Y. Shen (1973). A scheduling model for computer systems with two classes of processors. *Proceedings of the 1973 Sagamore Computer Conference on Parallel Processing*, 130-138.

[4] J. Carlier (1987). Scheduling jobs with release dates and tails on identical machines to minimize the makespan. *European Journal of Operational Research 29*, 298-306.

[5] J. Carlier and E. Pinson (1998). Jackson's pseudo-preemptive schedule for the $Pm|r_i, q_i|C_{\max}$ scheduling problem. *Annals of Operations Research 83*, 41-58.

[6] B. Chen (1994). Scheduling multiprocessor flow shops. D.Z. Du and J. Sun (eds.). *New Advances in Optimization and Approximation*, Kluwer, Boston, 1-8.

[7] B. Chen (1995). Analysis of classes of heuristics for scheduling two-stage flow shops with parallel machines at one stage. *Journal of the Operational Research Society 46*, 234-244.

[8] A. Gharbi and M. Haouari (2002). Minimizing makespan on parallel machines subject to release dates and delivery times. *Journal of Scheduling 5*, 329-355.

[9] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics 5*, 287-326.

[10] J.N.D. Gupta, A.M.A. Hariri, and C.N. Potts (1997). Scheduling a two-stage hybrid flow shop with parallel machines at the first stage. *Annals of Operations Research 69*, 171-191.

[11] M. Haouari and A. Gharbi (2003). An improved max-flow-based lower bound for minimizing maximum lateness on identical parallel machines. *Operations Research Letters 31*, 49-52.

[12] J.A. Hoogeveen, J.K. Lenstra, and B. Veltman (1996). Minimizing the makespan in a multiprocessor flow shop is strongly NP-hard. *European Journal of Operational Research 89*, 172-175.

[13] W.A. Horn (1974). Some simple scheduling algorithms. *Naval Research Logistics Quarterly 21*, 177-185.

[14] S.M. Johnson (1954). Optimal two- and three-stage production schedules with set-up times included. *Naval Research Logistics Quarterly 1*, 61-68.

[15] B. Jurisch (1995). Lower bounds for the job-shop scheduling problem on multi-purpose machines. *Discrete Applied Mathematics 58*, 145-156.

[16] J. Labetoulle, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan (1984). Preemptive scheduling of uniform machines subject to release dates. W.R. Pulleyblank (ed.). *Progress in Combinatorial Optimization*, Academic Press, Toronto, 245-261.

[17] M.A. Langston (1987). Interstage transportation planning in the deterministic flow-shop environment. *Operations Research 35*, 556-564.

[18] C.Y. Lee and G.L. Vairaktarakis (1994). Minimizing makespan in hybrid flow-shops. *Operations Research Letters 16*, 149-158.

[19] M. Mastrolilli and L.M. Gambardella (2000). Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling 3*, 3-20.

[20] R. McNaughton (1959). Scheduling with deadlines and loss functions. *Management Science 6*, 1-12.

[21] M. Perregaard (1995). *Branch-and-bound methods for the multi-processor job shop and flow shop scheduling problems.* Master's Thesis. Department of Computer Science, University of Copenhagen.

[22] M.-C. Portmann, A. Vignier, D. Dardilhac, and D. Dezalay (1998). Branch and bound crossed with GA to solve hybrid flowshops. *European Journal of Operational Research 107*, 389-400.

[23] E. Sanlaville (1995). Nearly on line scheduling of preemptive independent tasks. *Discrete Applied Mathematics 57*, pp 229–241.

[24] D.L. Santos, J.L. Hunsucker, and D.E. Deal (1995). Global lower bounds for flow shops with multiple processors. *European Journal of Operational Research 80*, 112-120.

[25] C. Sriskandarajah and S.P. Sethi (1989). Scheduling algorithms for flexible flow-shops: Worst and average performance. *European Journal of Operational Research 43*, 143-160.

[26] A. Vandevelde (1994). *Minimizing the makespan in a multiprocessor flow shop.* Master's Thesis. Department of Mathematics and Computer Science, Technische Universiteit Eindhoven.

[27] S.T. Webster (1996). A general lower bound for the makespan problem. *European Journal of Operational Research 89*, 516-524.