4OR (2008) 6:375–392
DOI 10.1007/s10288-007-0053-5

RESEARCH PAPER

Maximizing the minimum completion time on parallel machines

Mohamed Haouari · Mahdi Jemmali

Received: 18 October 2006 / Revised: 15 March 2007 / Published online: 4 September 2007
© Springer-Verlag 2007

Abstract We propose an exact branch-and-bound algorithm for the problem of maximizing the minimum machine completion time on identical parallel machines. The proposed algorithm is based on tight lower and upper bounds as well as an effective symmetry-breaking branching strategy. Computational results performed on a large set of randomly generated instances attest to the efficacy of the proposed algorithm.

Keywords Scheduling · Identical parallel machines · Branch-and-bound

MSC classification 90B35

1 Introduction

We investigate the problem of maximizing the minimum machine completion time on identical parallel machines. This problem has been first described by Deuermeyer et al. (1982) and is formally described as follows: a set J of n jobs has to be scheduled on m identical parallel machines (with $n > m \geq 2$). Each job $j \in J$ has to be processed non-preemptively for p_j units of time. The processing times are assumed to be deterministic and integers. Each machine processes at most one job at one time and each job cannot be processed by more than one machine at one time. The machines process jobs continuously from time zero onwards and all jobs are ready for

M. Haouari (✉) · M. Jemmali
Combinatorial Optimization Research Group-ROI, Ecole Polytechnique de Tunisie,
BP 743, 2078 La Marsa, Tunisia
e-mail: mohamed.haouari@ept.rnu.tn

M. Haouari
Faculty of Business Administration, Bilkent University, Ankara, Turkey

processing at time zero (thus, a machine never needs to wait idle for a job becoming available). Let $J_i \subset J$ denote the set of jobs assigned to machine M_i $(i = 1, \ldots, m)$, then the completion time of M_i is defined as $C_i = \sum_{j \in J_i} p_j$. The problem is to find a schedule that maximizes the minimum machine completion time $C_{\min} = \min_{1 \leq i \leq m} C_i$. Using the standard three-field notation of Lawler et al. (1993), this problem is denoted $P||C_{\min}$.

An application of $P||C_{\min}$ arises in the context of regional allocation of investments as follows. Given a set of n investment projects to be carried out in m different regions. Each project j $(j = 1, \ldots, n)$ is expected to generate a revenue p_j (or alternatively, p_j might represent the number of generated new jobs) and will be entirely carried out in a chosen region i $(i = 1, \ldots, m)$. For the sake of equity, the projects should be allocated to the different regions so as to maximize the minimal total regional revenue (or number of generated new jobs). Another application arises in the process of scheduling the replacement parts for the repeated repair of a machine. This problem occurs for instance in the context of modular gas turbine aircraft engines maintenance and is described as follows. A machine includes m identical working parts which frequently require replacement. A finite inventory of n spares is initially available. In most cases the inventory consists of both new as well as refurbished parts. Hence, these spares may have different field lives. Let p_j $(j = 1, \ldots, n)$ denotes the (deterministic) field life of spare j. The problem is to find a replacement part sequencing that maximizes the total time elapsed before the inventory is eventually replenished.

To the best of our knowledge, the literature related to $P||C_{\min}$ is solely devoted to the investigation of approximation algorithms. Deuermeyer et al. (1982) show that the longest processing time (LPT) heuristic has a performance guarantee of 3/4. This analysis has been tightened by Csirik et al. (1992) who show that the exact worst-case ratio of LPT is $(3m-1)/(4m-2)$. Finally, Woeginger (1997) gave a polynomial-time approximation scheme for this problem. Moreover, the *on-line* version of the problem has been addressed by Azar and Epstein (1997) and Tan and He (2002). Actually, this scanty literature is in sharp contrast to the huge number of papers that have been devoted so far to the investigation of the "dual" of $P||C_{\min}$. Indeed, if the objective is to minimize $\{\max_{1 \leq i \leq m} C_i\}$ instead of maximizing $\{\min_{1 \leq i \leq m} C_i\}$, then the problem turns out to be the well-known $P||C_{\max}$. This latter problem is definitely the most classical $\mathcal{NP}$-hard parallel machine scheduling problem. At this point, it is worth noting that while these two problems are in general different, it is easy to see that for the special two-machine case they are strictly equivalent. Hence, $P||C_{\min}$ is $\mathcal{NP}$-hard.

The objective of this paper is to present an exact branch-and-bound algorithm for $P||C_{\min}$. The proposed algorithm is based on tight lower and upper bounds as well as an effective branching scheme which make it feasible to solve to optimality many hard $P||C_{\min}$ instances. The remainder of the paper is organized as follows. In Sect. 2, we describe several upper bounds. In Sect. 3, we describe an optimization-based heuristic. In Sect. 4, we describe the branching scheme as well several important features of the branch-and-bound algorithm. In Sect. 5, we present the results of extensive computational experiments. Finally, some concluding remarks are provided.

In the sequel, w.l.o.g it is assumed that $p_1 \geq p_2 \geq \cdots \geq p_n$ and $C_1 \leq C_2 \leq \cdots \leq C_m$.

2 Upper bounds

It is well-known that the global effectiveness of a branch-and-bound algorithm strongly relies on the tightness of the embedded upper bounding procedure (for a maximization problem). In this section, we propose several upper bounds for $P \,||\, C_{\min}$.

2.1 A simple upper bound

Obviously, we have

$$\sum_{i=1}^{m} C_i \geq mC_1 \tag{2.1}$$

Since, $\sum_{i=1}^{m} C_i = \sum_{j=1}^{n} p_j$, thus we get a simple $O(n)$ upper bound

$$U_0 = \left\lfloor \frac{\sum_{j=1}^{n} p_j}{m} \right\rfloor \tag{2.2}$$

where $\lfloor a \rfloor$ represents the largest integer that is smaller than or equal to a.

2.2 Upper bounds derived from the $P||C_{\max}$

Now, we elucidate the close relationship between $P||C_{\min}$ and $P||C_{\max}$ and we show that lower bounds of the latter can be used to derive upper bounds for the former.

Lemma 2.1 *Given an instance I of $P||C_{\min}$ and a valid lower bound $L(I)$ on the optimal make span of the corresponding $P||C_{\max}$, then a valid upper bound on the optimal solution of the $P||C_{\min}$ instance is*

$$U(I) = \left\lfloor \frac{\sum_{j=1}^{n} p_j - L(I)}{m-1} \right\rfloor \tag{2.3}$$

Proof It suffices to observe that $L(I) \leq C_m \Rightarrow \sum_{i=1}^{m-1} C_i \leq \sum_{j=1}^{n} p_j - L(I)$. Moreover, $(m-1)C_1 \leq \sum_{i=1}^{m-1} C_i$. Thus, we get $C_1 \leq \frac{\sum_{j=1}^{n} p_j - L(I)}{m-1}$.

Consequently, any valid lower bound for $P||C_{\max}$ having a complexity $O(f(n))$ might be used to derive a corresponding upper bound for $P||C_{\min}$ having a complexity $O(\max(n, f(n))$ (in practice, both bounds would have the same complexity). The $P||C_{\max}$ is a fundamental scheduling problem that has been extensively investigated, and several lower bounds have been proposed in the literature. For instance, Dell'Amico and Martello (1995) have proposed the following simple linear-time lower

bound

$$L_1 = \max\left(p_1,\, p_m + p_{m+1},\, \left\lceil \sum_{j=1}^{n} p_j/m \right\rceil\right) \tag{2.4}$$

(recall that the jobs are indexed so that $p_1 \geq p_2 \geq \cdots \geq p_n$). Moreover, Dell'Amico and Martello (1995) have developed a tricky $O(n^2)$-time lower bound L_{DM} which is based on the equivalence between the decision problems corresponding to the bin packing problem (BPP) and $P||C_{\max}$, respectively. Indeed, checking whether n jobs could be processed on the m machines such that the makespan does not exceed a trial value C is equivalent to checking whether n items can be packed into m bins where the capacity of each bin is equal to C. A "no" answer to $P||C_{\max}$ decision problem is provided if a lower bound on the minimal number of bins (machines) that are required for packing the n items (jobs) exceeds m. Consequently, a valid lower bound for $P\,||\,C_{\max}$ is $C^* + 1$ where C^* is the largest value of C that yields a "no" answer. The BPP lower bound that has been proposed by Dell'Amico and Martello (1995) can be briefly described as follows: for each integer $p \in \{p_{m+2},\, p_{m+3}, \ldots, p_n\}$ and $p \leq \frac{C}{2}$, let

$$J_1 = \left\{j \in J;\ C - p < p_j\right\}$$
$$J_2 = \left\{j \in J;\ \frac{C}{2} < p_j \leq C - p\right\}$$
$$J_3 = \left\{j \in J;\ p \leq p_j \leq \frac{C}{2}\right\}$$

and define

$$BPP_1(C, p) = |J_1| + |J_2| + \max\left(0, \left\lceil \frac{\sum_{j\in J_3} p_j - C\,|J_2| + \sum_{j\in J_2} p_j}{C} \right\rceil\right)$$

$$BPP_2(C, p) = |J_1| + |J_2| + \max\left(0, \left\lceil \frac{|J_3| - \sum_{j\in J_2} \left\lfloor \frac{C - p_j}{p} \right\rfloor}{\left\lfloor \frac{C}{p} \right\rfloor} \right\rceil\right)$$

Thus, a valid BPP lower bound is $LB_{BPP} = \max_p\{\max(BPP_1(C, p), BPP_2(C, p))\}$. Hence, the corresponding $P||C_{\max}$ bound is denoted by L_{DM}.

Haouari et al. (2006) observe that a practical way for (possibly) improving a lower bound is to consider subset of machines and/or jobs. This observation is based on the following lemma.

Lemma 2.2 (Haouari and Gharbi 2004) *In any feasible schedule of a parallel machines problem with n jobs and m machines, there is at least a set of k machines* $(1 \leq k \leq m)$ *which must process at least* $\lambda_k = k\,\lfloor n/m \rfloor + \min(k, \rho)$ *jobs, where* $\rho = n - m\,\lfloor n/m \rfloor$.

Based on this lemma, Haouari et al. (2006) have derived from L_1 and L_{DM} two stronger $P||C_{\max}$ lower bounds to which we refer as L_2 and L_3, respectively. The complexity of these two bounds are $O(n^2)$ and $O(n^3)$, respectively. In the sequel, we denote by U_i $(i = 1, 2, 3)$ to the upper bounds that are derived from L_i, respectively.

Example 2.3 Consider the instance with $n = 5$, $m = 3$, $p_1 = 118$, $p_2 = 107$, $p_3 = 86$, $p_4 = 81$, $p_5 = 80$. We have $U_0 = \lfloor \frac{472}{3} \rfloor = 157$, and $L_1 = \max(118, 86 + 81, \lceil \frac{472}{3} \rceil) = 167$. Thus, $U_1 = \left\lfloor \frac{472-167}{3-1} \right\rfloor = 152$.

2.3 Enhancement procedures

In order to get tighter upper bounds, we propose two enhancement procedures which aim at strengthening previously developed upper bounds.

2.3.1 A lifting procedure for $P||C_{\min}$

An interesting consequence of Lemma 2.2 is the following corollary.

Corollary 2.4 *In any feasible schedule of a parallel machines problem with n jobs and m machines, there is at most a set of k machines* $(1 \leq k \leq m)$ *which must process at most* $\mu_k = k \lfloor n/m \rfloor + \max(0, \rho - m + k)$ *jobs.*

Proof Obviously, for $k = m$ we have $\mu_k = n$. For $1 \leq k \leq m - 1$, we have $\mu_k = n - \lambda_{m-k}$. Thus, we get $\mu_k = \begin{cases} k \lfloor n/m \rfloor, \text{for } 1 \leq k \leq m - \rho \\ k \lfloor n/m \rfloor + \rho - m + k, \text{for } m - \rho < k \leq m. \end{cases}$

Hence, there are at most μ_k jobs that are scheduled on any subset of k machines $(k = 1, \ldots, m)$. Consequently, if we consider the auxiliary $P||C_{\min}$ instance I_k that is defined on k machines and the job-set $J_k = \{1, \ldots, \mu_k\}$ (i.e., J_k includes the μ_k jobs that have the largest processing times), then its minimum completion time $C_{\min}(I_k)$ is a valid upper bound on the minimum completion time of the $P||C_{\min}$ instance. Therefore, if $U(I_k)$ denotes an upper bound on $C_{\min}(I_k)$, then a valid lifted upper bound for the $P||C_{\min}$ instance is

$$\bar{U} = \min_{1 \leq k \leq m} U(I_k) \tag{2.5}$$

Consequently, from each upper bound U_i $(i = 0, \ldots, 3)$ we can use (2.5) to derive a corresponding lifted upper bound $\bar{U}_i$. Clearly, if the computation of U requires $O(f(n))$ time, then $\bar{U}$ requires the computation of m bounds and can therefore be computed in $O(mf(n))$ time.

Example 2.5 Continued. We have $\rho = 2$. Thus, for $k = 1$ we get $\mu_1 = 1$, $J_1 = \{1\}$ and $U_0(I_1) = \lfloor \frac{118}{1} \rfloor = 118$. For $k = 2$, we get $\mu_2 = 3$, $J_2 = \{1, 2, 3\}$ and $U_0(I_2) = \lfloor \frac{118+107+86}{2} \rfloor = 155$. For $k = 3$, we get $\mu_3 = 5$, $J_3 = \{1, 2, 3, 4, 5\}$ and $U_0(I_3) = \lfloor \frac{118+107+86+81+80}{3} \rfloor = 157$. Thus we get $\bar{U}_0 = 118$. Note that in this case we have $\bar{U}_0 < U_1 < U_0$.

2.3.2 Second enhancement procedure

This procedure is based on the following observation. Given an upper bound U, we can derive a possibly better valid upper bound $\hat{U}$ by solving the following subset sum problem (SSP).

For each job j ($j = 1, \ldots, n$), denote by x_j the binary variable that takes the value 1 if job j is assigned to the machine having the minimum completion time and 0 otherwise. Thus, a valid upper bound is

$$\hat{U} = \text{Max} \sum_{j \in J} p_j x_j \tag{2.6}$$

subject to:

$$\sum_{j \in J} p_j x_j \leq U, \tag{2.7}$$

$$x_j \in \{0, 1\}, \quad \forall j \in J. \tag{2.8}$$

It is well-known that although being an $\mathcal{NP}$-hard problem, the SSP can be efficiently solved in a pseudo-polynomial time (see, Pisinger 2003). In the sequel, we denote by $\hat{U}_i$ ($i = 0, \ldots, 3$) the bounds that are obtained through enhancing the bounds U_i via solving an SSP, respectively.

2.3.3 A hybrid enhancement procedure

We have implemented a hybrid enhancement procedure which is based on a combination of both above-described enhancements procedures. More precisely, for each upper bound U_i ($i = 0, \ldots, 3$), we derive

$$U_i^* = \min_{1 \leq k \leq m} \hat{U}_i(I_k) \tag{2.9}$$

Hence, instead of using an upper bound $U_i(I_k)$ for instance I_k (as in 2.5), we solve an SSP for each k, and then we use the enhanced resulting value $\hat{U}_i(I_k)$.

Furthermore, for each bound U_i ($i = 1, 2, 3$) which is based on a $P||C_{\max}$ lower bound L_i, we have used in (3) the tightened value $\hat{L}_i$ which is given by

$$\hat{L}_i = \text{Min} \sum_{j \in J} p_j y_j \tag{2.10}$$

subject to:

$$\sum_{j \in J} p_j y_j \geq L_i, \tag{2.11}$$

$$x_j \in \{0, 1\}, \quad \forall j \in J. \tag{2.12}$$

where the binary variable y_j is defined for each job j ($j = 1, \ldots, n$) and takes the value 1 if job j is assigned to the machine having the maximum completion time and 0 otherwise. This hybrid enhancement strategy has been assessed through computational experiments on randomly generated instances. For these instances, the enhanced upper bounds exhibit an excellent performance. Indeed, in 92.32% of the cases U_3^* yields a proven optimal value.

3 A heuristic algorithm

Now, we turn our attention to the description of an effective heuristic which constitutes the second key feature of our branch-and-bound algorithm. This heuristic is in the spirit of the approach that has been implemented by Haouari et al. (2006) for $P \,||\, C_{\max}$. Basically, the proposed heuristic is a multi-start local search method which requires iteratively solving a sequence of $P2||C_{\min}$ instances. At each iteration, a pair of machines M_1 and M_m are selected (recall that $C_1 \leq C_2 \leq \cdots \leq C_m$) and the resulting $P2||C_{\min}$ instance which is defined on the jobset $\tilde{J} = J_1 \cup J_m$ is solved to optimality. To that aim, the $P2||C_{\min}$ is reformulated as the following subset sum problem

$$\text{Maximize} \sum_{j \in \tilde{J}} p_j z_j \tag{3.1}$$

subject to:

$$\sum_{j \in \tilde{J}} p_j z_j \leq \left\lfloor \sum_{j \in \tilde{J}} p_j / 2 \right\rfloor \tag{3.2}$$

$$x_j \in \{0, 1\}, \quad \forall j \in \tilde{J}, \tag{3.3}$$

where the binary variable z_j is defined for each job $j \in \tilde{J}$ and takes the value 1 if job j is assigned to the machine having the minimum completion time and 0 otherwise.

Let $C_{\min}^m$ denote the minimum completion time that is obtained after solving the corresponding subset sum problem. If an improved two-machine schedule has been obtained (hence, $C_{\min}^m > C_1$), then the schedules of machines M_1 and M_m are replaced by the new ones, the new values C_k ($k = 1, \ldots, m$) are computed, the machines are reindexed so that $C_1 \leq C_2 \leq \cdots \leq C_m$, and the procedure is reiterated (i.e., machines M_1 and M_m are rescheduled). Otherwise, the pair of machines (M_1, M_{m-1}) is selected and the corresponding $P2||C_{\min}$ instance is solved as a subset sum problem and so on. The procedure is stopped if no improvement has been achieved after sequentially considering all of the machine pairs (M_1, M_k) ($k = m, m-1, \ldots, 2$). A pseudo-code of the heuristic is described below.

Step 0. (*Initialization*)
Generate an initial schedule σ.
Step 1. (*Computation of the completion times*)
Compute C_k for $k = 1, \ldots, m$, Set $k = m$.

Step 2. (*Rescheduling of a machine pair*)
Solve using (13)–(15) the $P2||C_{\min}$ instance that is defined on M_1 and M_k. Let $C^k_{\min}$ denotes the obtained minimum completion time.
Step 3. (*Solution update*)
If $C^k_{\min} < C_1$ then update σ and go to Step 1.
Step 4. (*Termination test*)
If $(k > 2)$ then Set $k = k - 1$ and go to Step 2, otherwise Stop.

In order to produce an improved solution, the above described heuristic is reiterated $iter$ times (where the parameter $iter$ is set empirically) after varying the initial schedule. The starting solutions are obtained with a randomized LPT rule. This procedure iteratively selects two unscheduled jobs with the longest processing times and then randomly assigns one out of this pair to the first available machine. In our implementation, we set the maximal number of starting solutions ($iter$) to 100. However, the generation process is prematurely halted if a proven optimal solution (i.e., equal to an upper bound) is generated.

4 A branch-and-bound algorithm

4.1 Solution representation

Let S denote a feasible $P||C_{\min}$ solution with a corresponding job partition $J_1, J_2, \ldots, J_m$. Note that since the problem exhibits a natural symmetry inherent to identical machines and (possibly) some indistinguishable jobs (i.e., having identical processing times), we can associate with S a set of alternative symmetric solutions that might be generated by re-indexing machines and/or identical jobs. Actually, the number of such alternative symmetric solutions is generally huge. For instance, even for the apparently "benign" case where all jobs are different, we can associate with any feasible solution S, a set of $m! - 1$ alternative symmetric solutions. Obviously, this symmetry might significantly increase the computational burden and challenge the efficacy of an optimization algorithm. In order to avoid this serious drawback, we propose an efficient *symmetry-breaking* representation. It is noteworthy that the concept of symmetry-breaking (or -defeating) is widely used in computational integer programming for improving discrete model representations (see, Sherali and Smith 2001). We denote by n_i $(i = 1, \ldots, m)$ the number of jobs that are assigned to machine M_i (i.e., $n_i = |J_i|$). Each subset J_i $(i = 1, \ldots, m)$ is represented by a permutation $\sigma_i = (\sigma_i^1, \sigma_i^2, \ldots, \sigma_i^{n_i})$ of the n_i job indices. Hence, we associate to each solution S a permutation $\sigma(S) = \sigma_1\sigma_2 \ldots \sigma_m$. Such a permutation is said to be *valid* if it satisfies the following conditions:

(C1) $\sigma_i^k < \sigma_i^{k+1}$, for $k = 1, \ldots, n_i - 1,\ i = 1, \ldots, m$
(C2) $\sum_{k=1}^{n_i} p_{\sigma_i^k} \leq \sum_{k=1}^{n_{i+1}} p_{\sigma_{i+1}^k}$, for $i = 1, \ldots, m - 1$
(C3) if for some i $(i = 1, \ldots, m - 1)$, we have $\sum_{k=1}^{n_i} p_{\sigma_i^k} = \sum_{k=1}^{n_{i+1}} p_{\sigma_{i+1}^k}$ then $\sigma_i^1 < \sigma_{i+1}^1$

(C4) if for some j ($j = 1, \ldots, m-1$), we have $p_j = p_{j+1}$ then job $j+1$ is sequenced after job j in σ.

Condition (C1) specifies that each subsequence σ_i ($i = 1, \ldots, m$) is a nondecreasing list of the job indices. Condition (C2) requires that the machines are indexed according to nondecreasing completion times. Condition (C3) specifies that ties are broken in favor of the machine having the smallest job index. Condition (C4) requires that if two jobs j and $j+1$ are identical and j is assigned to a machine M_α then $j+1$ is necessarily assigned to machine M_β such that $\beta \geq \alpha$.

The main advantage of this representation is that each set of alternative symmetric solutions is represented by a *unique* permutation of the n jobs.

Example 4.1 Consider the instance with $n = 8$, $m = 3$, $p_1 = 15$, $p_2 = 12$, $p_3 = 12$, $p_4 = 10$, $p_5 = 9$, $p_6 = 8$, $p_7 = 5$, $p_8 = 5$. Let S denote the solution which is specified by the following assignment: jobs 1, 6, and 7 are assigned to a first machine, jobs 2, 5, and 8 are assigned to a second machine, and jobs 3 and 4 are assigned to a third machine. The only valid representation of this solution is $\sigma(S) = \sigma_1\sigma_2\sigma_3$ where $\sigma_1 = (2, 4)$, $\sigma_2 = (3, 5, 7)$, and $\sigma_3 = (1, 6, 8)$.

Conversely, given a valid permutation σ^* and a (feasible) threshold value $C_{\min}$, one can construct in $O(n)$ time a feasible solution S having a minimum completion time equal to $C_{\min}$ and satisfying $\sigma(S) = \sigma^*$. Clearly, the jobs that are assigned to M_1 are $\{\sigma^*(1), \sigma^*(2), \ldots, \sigma^*(\nu_1)\}$ where ν_1 is the smallest integer ν satisfying $\sum_{j=1}^{\nu} p_{\sigma^*(j)} \geq C_{\min}$. Next, the jobs that are assigned to M_k ($2 \leq k \leq m-1$) are $\{\sigma^*(\nu_{k-1}+1), \ldots, \sigma^*(\nu_k)\}$ where ν_k is the smallest integer ν satisfying $\sum_{j=\nu_{k-1}+1}^{\nu} p_{\sigma^*(j)} \geq C_{k-1}$. Finally, the unscheduled jobs $\{\sigma^*(\nu_{m-1}+1), \ldots, \sigma^*(n)\}$ are assigned to M_m.

4.2 Branching scheme and data representation

Since each solution is represented by a permutation of the n jobs, we have adopted the following branching scheme. Each node N_l of level l of the search tree corresponds to a partial valid permutation $\sigma(N_l) = (\sigma_1, \sigma_2, \ldots, \sigma_l)$ of l jobs. Therefore, the corresponding set of unscheduled jobs is $\bar{J}(N_l) = J \setminus \{\sigma_1, \sigma_2, \ldots, \sigma_l\}$. Obviously, the root node N_0 corresponds to the empty permutation. Each child of node N_l corresponds to appending an unscheduled job $j_0 \in \bar{J}(N_l)$ to $\sigma(N_l)$ and thus getting an extended partial permutation $(\sigma_1, \sigma_2, \ldots, \sigma_l, j_0)$. This branching strategy amounts to sequentially loading $M_1, M_2, \ldots, M_m$, in that order, while taking heed of Conditions (C1)–(C4).

With each node N at level l of the search tree is associated the following data:

- $\sigma(N)$: valid permutation of l jobs
- $J(N)$: subset of scheduled jobs ($|J(N)| = l$)
- $i(N)$: index of the last loaded machine
- $L(N)$: total workload of machine $M_{i(N)}$
- $C_1(N)$: total workload of machine M_1
- $a(N)$: lower bound on the workload of machine $M_{i(N)}$
- $b(N)$: upper bound on the workload of machine $M_{i(N)}$

- $j(N)$: index of the last scheduled job (i.e., $j(N) = \sigma_l$)
- $j_0(N)$: smallest index of the jobs scheduled on $M_{i(N)-1}$
- $j_1(N)$: smallest index of the jobs scheduled on $M_{i(N)}$
- $\bar{J}(N)$: set of unscheduled jobs that are candidate to be scheduled on $M_{i(N)}$.

The data associated with the root node N_0 is : $\sigma(N_0) = \emptyset$, $J(N_0) = \emptyset$, $i(N_0) = 1$, $L(N_0) = 0$, $C_1(N_0) = 0$, $a(N_0) = LB + 1$ (where LB denotes the value of the heuristic solution), $b(N_0) = UB$ (where UB denotes the value of an upper bound which is computed at the root node), $j(N_0) = 0$, $j_0(N_0) = 0$, $j_1(N_0) = 0$, and $\bar{J}(N_0) = J$. Now, we provide some details for a non-root node N.

Computation of $a(N)$: If $i(N) = 1$ then $a(N) = LB + 1$ (because we are seeking for a solution that strictly outperforms the heuristic one). Moreover, since only valid permutations are generated, then from conditions (C2) and (C3), we get

$$a(N) : \begin{cases} C_{i(N)-1}, \text{ if } i(N) > 1 \quad \text{and} \quad j_1(N) > j_0(N) \\ C_{i(N)-1} + 1 \ , \text{ if } i(N) > 1 \quad \text{and} \quad j_1(N) < j_0(N). \end{cases}$$

Computation of $b(N)$: An upper bound on the total workload of $M_{i(N)}$ is computed by considering a reduced $P||C_{\min}$ instance defined on $m' = m - i(N) + 1$ machines and $n' = \left|\bar{J}(N)\right| + 1$ jobs. The jobset comprises the unscheduled jobs as well as a dummy job $n+1$ corresponding to the subset of jobs that have been already assigned to $M_{i(N)}$. Thus, the processing time of this dummy job is equal to the total workload of machine $M_{i(N)}$ (i.e., $p_{n+1} = L(N)$)

Computation of $\bar{J}(N)$: The set of unscheduled jobs that are candidate to be scheduled on $M_{i(N)}$ immediately after $j(N)$ is

$$\bar{J}(N) = \left\{ j \in J \setminus J(N) : j > j(N) \quad \text{and} \quad p_j + L(N) \leq b(N) \right\}$$

Assume that node N is branched and that each child node N^+ is obtained by appending a job $j \in J \setminus J(N)$ to $\sigma(N)$. First, consider the situation where $1 < i(N) \leq m - 1$. Two cases may occur:

Case (i): $L(N) < a(N)$ and $j \in \bar{J}(N))$: in this case, additional jobs must be assigned to machine $M_{i(N)}$ and job j is a valid candidate. Therefore, for node N^+ we define $\sigma(N^+) = \sigma(N)j$, $J(N^+) = J(N) \cup \{j\}$, $i(N^+) = i(N)$, $L(N^+) = L(N) + p_j$, $C_1(N^+) = C_1(N)$, $a(N^+) = a(N)$, $j(N^+) = j$, $j_0(N^+) = j_0(N)$, $j_1(N^+) = j_1(N)$. Moreover, $b(N^+)$ and $\bar{J}(N^+)$ are computed directly.

Case (ii): $L(N) \geq a(N)$ or $j \notin \bar{J}(N)$: in this case, no further job should be assigned to machine $M_{i(N)}$ and job j should be assigned to the next machine $M_{i(N)+1}$. Therefore, the data of N^+ is : $\sigma(N^+) = \sigma(N)j$, $J(N^+) = J(N) \cup \{j\}$, $i(N^+) = i(N) + 1$, $L(N^+) = p_j$, $C_1(N^+) = C_1(N)$, $a(N^+) = a(N) + \delta$ where $\delta = 1$ if $j < j_1(N)$ and 0, otherwise, $j(N^+) = j$, $j_0(N^+) = j_1(N)$, $j_1(N^+) = j$. Here again, $b(N^+)$ and $\bar{J}(N^+)$ are computed directly.

Now, consider the special case where $i(N) = 1$. Here, three cases may occur:

Case (iii): $L(N) < a(N)$ and $j \in \bar{J}(N)$: in this case additional jobs must be assigned to machine M_1 and job j is a valid candidate. We set $C_1(N^+) = L(N) + p_j$. The remaining data are updated similarly to *Case (i).*

Case (iv): $L(N) \geq a(N)$ and $j \notin \bar{J}(N)$: in this case job j cannot be assigned to machine M_1 and requires to be therefore assigned to machine M_2. This situation is similar to *Case (ii).*

Case (v): $L(N) \geq a(N)$ and $j \in \bar{J}(N)$: in this case, job j is either assigned to machine M_1 or to machine M_2. Of course, in this latter case no further job would be assigned to M_1. It is noteworthy that this yields two *different* descendent nodes N_1^+ and N_2^+ having the same partial permutation. Therefore, the data associated with nodes N_1^+ and N_2^+ are updated similarly to Cases (ii) and (iii), respectively.

4.3 Implementation details

Implemented bounds. We have performed extensive computational experiments in order to assess different combinations of upper bounding procedures. We found that an efficient implementation is obtained when U_3^* is computed at the root node and $\bar{U}_1$ at non-root nodes. While the former upper bound is often very tight, but is (relatively) time-consuming, the latter one performs reasonably well and is very fast. Moreover, the lower bound at the root node (LB) is delivered by the heuristic which is described in Sect. 3.

Node pruning. Obviously, whenever a solution having a minimum completion time satisfying $C_{\min} > LB$ is found, the incumbent value is updated and all the active nodes N having $C_1(N) \leq LB$ are pruned. In addition, a node N is pruned if any of the following conditions holds:

- $b(N) < a(N)$
- $i(N) = m - 1$ and $L(N) \geq a(N)$.

The first condition refers to the case where N is infeasible (i.e., the corresponding permutation is not valid). The second condition refers to the case where N is a leaf of the search tree. Indeed, if $m - 1$ machines are loaded, then the unscheduled jobs are necessarily assigned to the remaining unloaded machine M_m. In this case, the incumbent value is updated by setting $LB = \min(LB, C_1(N), \sum_{j \in \bar{J}(N)} p_j)$.

Taking heed of Condition (C4). Assume that for some node N, we have $\{j, j+1, \ldots, j+h\} \subset (J \setminus J(N))$ and $p_j = p_{j+1} = \cdots = p_{j+h}$. Then, in order to generate a permutation for which Condition (C4) holds, the children nodes that are obtained by appending to $\sigma(N)$ the h jobs $j+1, \ldots, j+h$, respectively, are not created.

Search strategy. We have adopted a depth-first search strategy. In the sequel, we refer to the resulting branch-and-bound algorithm by $BB1$.

4.4 An alternative branching strategy

In order to get a better insight of the actual pertinence of the proposed symmetry-breaking branching strategy, we have implemented a second version of the proposed depth-first branch-and-bound algorithm with the same upper and lower bounds but with a different branching strategy. This latter strategy is similar to the one that has been previously implemented by Dell'Amico and Martello (1995) for solving $P||C_{\max}$. Hereafter, we refer to this second version by $BB2$. On the contrary to the above described branching strategy, the m machines are loaded simultaneously. Indeed, while the root node represents the empty schedule, at level k ($k \geq 1$) of the search tree, a node represents a partial schedule on the m machines of the job subset $\{1, 2, \ldots, k\}$. Given a node at level k, m child nodes are created by assigning job $k+1$ to machines M_1, $M_2, \ldots, M_m$, respectively. An associated upper bound is computed by considering a $Pm||C_{\min}$ instance defined on the subset $\bar{J}$ of unscheduled jobs as well as m dummy jobs having processing times $C_1, C_2, \ldots, C_m$, respectively.

Moreover, we have implemented the three following dominance rules:

Rule 1: If for some machine M_i ($i = 1, \ldots, m-1$) we have $C_i = C_{i+1}$ then job $k+1$ is only assigned to machine M_i (otherwise unnecessary equivalent solutions would be created)

Rule 2: If for some job j ($j = 1, \ldots, n-1$) we have $p_j = p_{j+1}$ and job j is assigned to machine M_i ($i \geq 2$) then job $j+1$ could not be assigned to any of machines M_1, $M_2, \ldots, M_{i-1}$ (here again, because unnecessary equivalent solutions would be created)

Rule 3: If $|\bar{J}| < m$ then job $j+1$ could only be assigned to machines M_1, $M_2, \ldots, M_{|\bar{J}|}$ (otherwise, dominated nodes would be created).

5 Computational results

We have coded the proposed upper and lower bounds as well as the branch-and-bound algorithms $BB1$ and $BB2$ in Microsoft Visual C++ (Version 6.0). All our experiments were obtained on a Pentium IV 3.2 GHz Personal Computer with 3 GB RAM. We have considered five problem classes that have been randomly generated as described in Dell'Amico and Martello (1995) who used them for testing an exact branch-and-bound algorithm for $P||C_{\max}$. The processing times were generated according to the following distributions:

- *Class 1*: discrete uniform distribution on [1,100]
- *Class 2*: discrete uniform distribution on [20,100]
- *Class 3*: discrete uniform distribution on [50,100]
- *Class 4*: normal distribution with Mean 100 and SD 50
- *Class 5*: normal distribution with Mean 100 and SD 20.

The number of jobs is ranging between 10 and 10,000, and the number of machines is ranging between 2 and 15. For each class, and each (n, m) combination, 10 instances were generated. Hence, a total number of 2,050 instances have been generated.

Table 1 displays a summary of the results that were obtained for the upper bounding procedures. In this table, each column corresponds to an (n, m) combination. For each upper bound, we provide the number of times (out of 50) it yields the minimal value over all of the bounds.

We see from this table, that the enhanced bounds exhibit a very good performance. In particular, the bound U_3^* consistently dominates all the other ones.

In Table 2, we report a summary of the results obtained with $BB1$. From this table, we see that the implemented bounds are very tight, since for most instances branching was unnecessary. Actually, we found that branching was required for only 166 instances out of 2,050. Moreover, we observe that only 6 instances remained unsolved after reaching the time limit of 800 s.

Table 3 displays the results that were obtained with $BB2$. Since we have implemented the same upper and lower bounds as in $BB1$, then here again branching was required for 166 instances out of 2,050. However, we observe that 89 instances remained unsolved after reaching the CPU time limit. This result provides evidence of the superiority of the symmetry-breaking branching strategy that has been implemented in $BB1$.

In order to get a better picture of the performance of $BB1$, we have generated an additional class of 260 instances where for each instance with n jobs ($n = 10$, 25, 50, 100, 250, 500, 1,000) the processing times are drawn from discrete uniform distribution on $[1, n]$. For these instances, it is expected that only a few jobs have equal processing times. The results are summarized in Table 4. We see that while all of these instances were optimally solved by $BB1$ (mostly at the root node), there are seven instances that remained unsolved by $BB2$.

Finally, we have run our algorithms on a class of *perfect packing* instances. This class comprises 1,520 instances that have been randomly generated as indicated in Dell'Amico and Martello (1995). A nice property peculiar to this latter class is that the optimal schedule has an equal completion time on each machine (note that in this case $P||C_{\min}$ and $P||C_{\max}$ are strictly equivalent). The results are consistent with those obtained for the other problem classes. Indeed, all of these instances were solved to optimality by $BB1$ and branching was required for only 18 instances. Among these 18 instances, $BB2$ failed to solve 15.

6 Conclusion

In this paper, we have proposed an exact branch-and-bound algorithm for $P||C_{\min}$. The proposed algorithm includes several distinctive algorithmic features. In particular, it is based on tight lower and upper bounds as well as an effective symmetry-breaking branching strategy. Computational results attest to the efficacy of the proposed algorithm, which has been able to solve to optimality all but 6 instances out of 3,830 randomly generated instances.

Table 1 Performance of the upper bounding procedures

n	10			25					50					100				250				500				1,000				2,500				5,000				10,000				Total
m	2	3	5	2	3	5	10	15	2	3	5	10	15	3	5	10	15	3	5	10	15	3	5	10	15	3	5	10	15	3	5	10	15	3	5	10	15	3	5	10	15	
U_0	48	26	4	50	50	50	27	1	50	50	50	50	48	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	1,904
U_1	48	26	16	50	50	50	28	11	50	50	50	50	48	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	1,927
U_2	48	28	18	50	50	50	29	11	50	50	50	50	48	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	1,932
U_3	48	28	33	50	50	50	31	11	50	50	50	50	48	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	1,949
$\overline{U}_0$	48	28	4	50	50	50	34	3	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	1,917
$\overline{U}_1$	48	28	16	50	50	50	37	23	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	1,952
$\overline{U}_2$	48	27	18	50	50	50	38	23	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	1,954
$\overline{U}_3$	48	28	33	50	50	50	46	23	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	1,978
$\hat{U}_0$	50	38	15	50	50	50	27	1	50	50	50	50	48	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	1,929
$\hat{U}_1$	50	38	31	50	50	50	28	11	50	50	50	50	48	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	1,956
$\hat{U}_2$	50	42	34	50	50	50	29	11	50	50	50	50	48	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	1,964
$\hat{U}_3$	50	42	50	50	50	50	31	11	50	50	50	50	48	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	1,982
U_0^*	50	48	15	50	50	50	39	21	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	1,973
U_1^*	50	50	32	50	50	50	42	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	2,024
U_2^*	50	50	35	50	50	50	43	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	2,028
U_3^*	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	2,050

Table 2 Performance of the branch and bound algorithm $BB1$

n	m	Class 1		Class 2		Class 3		Class 4		Class 5	
		NN	Time	*NN*	Time	*NN*	Time	*NN*	Time	*NN*	Time
10	2	1	–	1	–	1	–	1	–	1	–
	3	90	0.001	137	0.001	48	0.002	215	0.002	100	0.002
	5	128	0.002	152	0.002	188	–	150	0.003	140	0.002
25	2	1	–	1	–	1	0.002	1	–	1	–
	3	1	–	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–	1	–
	10	19649828	85	31675367	129 (9)	235178	1.380	4294944.9	19.625	10014760	38.675
	15	24572119	128 (9)	268948	1.900	42	0.006	14537221	77.878	48	0.005
50	2	1	–	1	–	1	–	1	–	1	–
	3	1	–	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–	1	–
	10	1	–	1	–	1	–	1	–	1	–
	15	17	0.003	21	0.001	1	0.001 (8)	4498586.8	20.335	1	0.001 (8)
100	3	1	–	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–	1	–
	10	1	–	1	–	1	–	1	–	1	–
	15	1	–	1	–	1	–	1	–	1	–
250	3	1	–	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–	1	–
			–		–		–		–		–
	10	1	–	1	–	1	–	1	–	1	–
	15	1	–	1	–	1	–	1	–	1	–
500	3	1	–	1	0.001	1	0.001	1	–	1	0.001
	5	1	–	1	–	1	–	1	–	1	–
	10	1	–	1	–	1	–	1	–	1	0.001
	15	1	–	1	0.001	1	0.002	1	–	1	0.002
1,000	3	1	–	1	0.004	1	0.006	1	–	1	0.004
	5	1	–	1	–	1	–	1	–	1	0.003
	10	1	–	1	–	1	–	1	–	1	0.004
	15	1	–	1	0.006	1	0.008	1	0.002	1	0.006
2,500	3	1	0.002	1	0.016	1	0.021	1	0.002	1	0.016
	5	1	0.002	1	0.002	1	0.002	1	0.002	1	0.016
	10	1	0.002	1	0.002	1	0.002	1	0.002	1	0.016
	15	1	0.002	1	0.02	1	0.021	1	0.003	1	0.019
	3	1	0.006	1	0.07	1	0.084	1	0.005	1	0.062
	5	1	0.006	1	0.004	1	0.005	1	0.005	1	0.050

Table 2 continued

n	m	Class 1		Class 2		Class 3		Class 4		Class 5	
		NN	Time	*NN*	Time	*NN*	Time	*NN*	Time	*NN*	Time
5,000	10	1	0.006	1	0.005	1	0.005	1	0.005	1	0.047
	15	1	0.005	1	0.08	1	0.098	1	0.005	1	0.070
10,000	3	1	0.016	1	0.519	1	0.520	1	0.016	1	0.294
	5	1	0.015	1	0.016	1	0.016	1	0.016	1	0.177
	10	1	0.015	1	0.016	1	0.016	1	0.016	1	0.177
	15	1	0.016	1	1.588	1	2.605	1	0.016	1	0.575

(-) means that the average CPU time is less than 0.001 s
The figures in brackets indicate the number of solved instances if less than 10

Table 3 Performance of the branch and bound algorithm $BB2$

n	m	Class 1		Class 2		Class3		Class 4		Class 5	
		NN	Time	*NN*	Time	*NN*	Time	*NN*	Time	*NN*	Time
10	2	1	–	1	–	1	–	1	–	1	—
	3	48	0.001	117	0.001	45	0.001	207	0.001	97	0.002
	5	1,366	0.007	1,019	0.005	582	0.004	2,159	0.013	645	0.004
	2	1	–	1	–	1	0.002	1	–	1	–
	3	1	–	1	–	1	–	1	–	1	–
25	5	1	–	1	–	1	–	1	–	1	–
	10	2,9754,513	181.901 (7)	4,6047,433	(2)	1	0.007 (2)	43,02,480	30.422 (2)	–	(0)
	15	–	(0)	1	(1)	1	0.003 (3)	(0)		1	0.003 (3)
50	2	1	–	1	–	1	–	1	–	1	–
	3	1	–	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–	1	–
	10	1	–	1	–	1	–	1	–	1	–
	15	1	– (8)	1	0.001	1	0.002 (9)	1	0.002 (6)	1	– (8)
100	3	1	–	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–	1	–
	10	1	–	1	–	1	–	1	–	1	–
	15	1	–	1	–	1	–	1	–	1	–
250	3	1	–	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–	1	–
	10	1	–	1	–	1	–	1	–	1	–
	15	1	–	1	–	1	–	1	–	1	–
500	3	1	–	1	0.001	1	0.001	1	–	1	0.001
	5	1	–	1	–	1	–	1	–	1	–
	10	1	–	1	–	1	–	1	–	1	0.001
	15	1	–	1	0.001	1	0.002	1	–	1	0.002
1,000	3	1	–	1	0.004	1	0.006	1	–	1	0.004
	5	1	–	1	–	1	–	1	–	1	0.003
	10	1	–	1	–	1	–	1	–	1	0.004
	15	1	–	1	0.006	1	0.008	1	0.002	1	0.006
	3	1	0.002	1	0.016	1	0.021	1	0.002	1	0.016
	5	1	0.002	1	0.002	1	0.002	1	0.002	1	0.016

Table 3 continued

n	m	Class 1		Class 2		Class3		Class 4		Class 5	
		NN	Time	*NN*	Time	*NN*	Time	*NN*	Time	*NN*	Time
2,500	10	1	0.002	1	0.002	1	0.002	1	0.002	1	0.016
	15	1	0.002	1	0.02	1	0.021	1	0.003	1	0.019
5,000	3	1	0.006	1	0.07	1	0.084	1	0.005	1	0.062
	5	1	0.006	1	0.004	1	0.005	1	0.005	1	0.050
	10	1	0.006	1	0.005	1	0.005	1	0.005	1	0.047
	15	1	0.005	1	0.080	1	0.098	1	0.005	1	0.070
10,000	3	1	0.016	1	0.519	1	0.520	1	0.016	1	0.294
	5	1	0.015	1	0.016	1	0.016	1	0.016	1	0.177
	10	1	0.015	1	0.016	1	0.016	1	0.016	1	0.177
	15	1	0.016	1	1.588	1	2.605	1	0.016	1	0.575

(-) means that the average CPU time is less than 0.001 s
The figures in brackets indicate the number of solved instances if less than 10

Table 4 Performance of $BB1$ and $BB2$ on instances with processing times drawn from $[1, n]$

n	m	*BB1*		*BB2*	
		NN	Time	*NN*	Time
10	3	1	–	1	–
	5	101	–	66	–
25	3	1	–	1	–
	5	1	–	1	–
	10	3954509	21.165	2545768	15.335
	15	12301985	73.786	242984	2.549(3)
50	3	1	–	1	–
	5	1	–	1	–
	10	1	–	1	–
	15	1	–	1	–
100	3	1	–	1	–
	5	1	–	1	–
	10	1	–	1	–
	15	1	–	1	–
250	3	1	–	1	–
	5	1	–	1	–
	10	1	–	1	–
	15	1	–	1	–
500	3	1	–	1	–
	5	1	0.001	1	0.001
	10	1	–	1	–
	15	1	0.001	1	0.001
1,000	3	1	0.002	1	0.002
	5	1	0.003	1	0.003
	10	1	0.004	1	0.004
	15	1	0.005	1	0.005

(-) means that the average CPU time is less than 0.001 s.
The figures in brackets indicate the number of solved instances if less than 10

We expect that some of the ideas that have been developed in this paper would prove useful for the exact solution of similar parallel machine scheduling problems, but this would require further investigation.

References

Azar Y, Epstein L (1997) On-line machine covering. Lect Notes Comput Sci 1284:23–36

Csirik J, Kellerer H, Woeginger G (1992) The exact LPT-bound for maximizing the minimum completion time. Oper Res Lett 11:281–287

Dell'Amico M, Martello S (1995) Optimal scheduling of tasks on identical parallel processors. ORSA J Comput 7:191–200

Deuermeyer BL, Friesen DK, Langston MA (1982) Scheduling to maximize the minimum processor finish time in a multiprocessor system. SIAM J Algorithms Discret Methods 3:190–196

Haouari M, Gharbi A (2004) Lower bounds for scheduling on identical parallel machines with heads and tails. Ann Oper Res 129:187–204

Haouari M, Gharbi A, Jemmali M (2006) Tight bounds for the identical parallel machine scheduling problem. Int Trans Oper Res 13:529–548

Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys D (1993) Sequencing and scheduling: algorithms and complexity. In: Graves SS, Rinnooy Kan AHG, Zipkin P (eds) Handbooks in operations research and management science, vol 4, pp 445–522

Pisinger D (2003) Dynamic programming on the word RAM. Algorithmica 35:437–459

Sherali HD, Smith JC (2001) Improving discrete model representations via symmetry considerations. Manage Sci 47:1396–1407

Tan ZY, He Y (2002) Ordinal on-line scheduling for maximizing the minimum machine completion time. J Comb Optim 6:199–206

Woeginger GJ (1997) A polynomial time approximation scheme for maximizing the minimum completion time. Oper Res Lett 20:149–154