Intl. Trans. in Op. Res. 15 (2008) 19–34

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Tight bounds for the identical parallel machine-scheduling problem: Part II

Mohamed Haouari[a,b] and Mahdi Jemmali[a]

[a]*Combinatorial Optimization Research Group – ROI, Ecole Polytechnique de Tunisie, La Marsa, Tunisia*
E-mail: mh6368@yahoo.com,
[b]*Faculty of Business Administration, Bilkent University, Ankara, Turkey*

Received 12 January 2007; received in revised form 28 May 2007; accepted 15 July 2007

Abstract

A companion paper introduces new lower bounds and heuristics for the problem of minimizing makespan on identical parallel machines. The objective of this paper is threefold. First, we describe further enhancements of previously described lower bounds. Second, we propose a new heuristic that requires solving a sequence of 0–1 knapsack problems. Finally, we show that embedding these newly derived bounds in a branch-and-bound procedure yields a very effective exact algorithm. Moreover, this algorithm features a new symmetry-breaking branching strategy. We present the results of computational experiments that were carried out on a large set of instances and that attest to the efficacy of the proposed algorithm. In particular, we report proven optimal solutions for some benchmark problems that have been open for some time.

Keywords: scheduling; identical parallel machines; lower bounds; heuristics; branch-and-bound

1. Introduction

We address the problem of scheduling a set of n jobs on m identical parallel machines with the objective of minimizing the makespan. This very basic combinatorial optimization problem, which is denoted by $P||C_{\max}$, is probably one of the most intensely investigated deterministic $\mathcal{NP}$-hard machine-scheduling problems (Lawler et al., 1993). Recent contributions to this fundamental scheduling problem are the hybrid bin-packing-based heuristic by Alvim and Ribeiro (2004), the multi-exchange neighborhood search heuristic by Frangioni et al. (2004), and the iterated local search algorithm by Tang and Luo (2006). In a previous companion paper (Haouari et al., 2006), the authors have proposed tight lower and upper bounding strategies for $P||C_{\max}$. These latter upper and lower bounds strategies have proved to often outperform the best bounds from the literature. The contribution of this paper is threefold. First, we describe a new improved variant of the so-called *multi-start subset-sum-based improvement heuristic*, which was first

Published by Blackwell Publishing, 9600 Garsington Road, Oxford, OX4 2DQ, UK and 350 Main St, Malden, MA 02148, USA.

introduced in Haouari et al. (2006). Second, we propose an enhancement procedure for strengthening previously developed lower bounds. Third, we present an exact branch-and-bound algorithm for solving $P||C_{\max}$. This algorithm includes two distinctive features: (i) strong lower and upper bounds and (ii) a symmetry-breaking representation of a schedule (on parallel machines) as a permutation of jobs.

We present the results of extensive computational experiments that were carried out on both randomly generated as well as benchmark instances. These results provide strong empirical evidence that the proposed algorithm consistently solves $P||C_{\max}$ instances in moderate CPU time. In particular, we report proven optimal solutions for 13 benchmark problems that have been open for some time.

The paper is organized as follows: in Section 2, we describe a new improved heuristic. In Section 3, we introduce a new lower bound enhancement procedure. The details of our branch-and-bound algorithm are provided in Section 4. In Section 5, the performance of the proposed exact procedure is assessed through an extensive computational study. Finally, some concluding remarks are provided.

Throughout the paper, we shall conform to the following notation. J: set of jobs, n: number of jobs, m: number of machines, p_j: processing time of job $j \in J$, J_k: subset of jobs that are assigned to machine M_k ($k = 1, \ldots, m$), C_k: completion time of machine M_k ($k = 1, \ldots, m$), $L_{TV} = \max(p_1, p_m + p_{m+1}, \lceil \sum_{j=1}^{n} p_j/m \rceil)$, L_{AR} and U_{AR} are the lower and upper bounds that have been proposed by Alvim and Ribeiro (2004), respectively, $\tilde{L}_{FS}$: the lower bound based on Fekete and Schepers' bin-packing lower bound that is described in Haouari et al. (2006), U_{MSS}: the value of the solution provided by the MSS heuristic that is described in Haouari et al. (2006). Moreover, w.l.o.g. it is assumed that $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_n$ and $C_1 \geqslant C_2 \geqslant \cdots \geqslant C_m$.

2. An improvement of the multi-start subset-sum-based improvement heuristic

In this section, we present a new $P||C_{\max}$ heuristic that is in the same vein as the multi-start subset-sum-based improvement heuristic (MSS), which was first introduced in the companion paper. For the sake of clarity, we briefly recall the basic ideas underlying this heuristic. Given an initial feasible solution σ, MSS requires iteratively selecting two machines M_1 and M_k (with $2 \leqslant k \leqslant m$ and $C_k < C_1$ and optimally solving the $P_2||C_{\max}$ instance that is defined on the job subset $\tilde{J} = J_1 \cup J_k$, this latter problem being reformulated as the following subset-sum problem (SSP):

$$\text{SSP}: \text{Maximize} \sum_{j \in \tilde{J}} p_j y_j \tag{1}$$

subject to:

$$\sum_{j \in \tilde{J}} p_j y_j \leqslant \left\lfloor \sum_{j \in \tilde{J}} p_j/2 \right\rfloor \tag{2}$$

$$y_j \in \{0, 1\}, \quad \forall j \in J_k \cup J_1 \tag{3}$$

Let $S = \{j \in \tilde{J} : y_j = 1\}$. Then, the jobs belonging to S and $\tilde{J}\backslash S$ are assigned to M_k and M_1, respectively. Hence, a possibly better reassignment of the jobs on the selected pair of machines is achieved and therefore an improved solution is derived. Then, a new pair of machines is selected and the process is continued until no further improvement can be achieved. This search process is reiterated a preset number of iterations (*n_iter*) starting from randomly generated initial solutions. We refer to Haouari et al. (2006) for a more detailed description of *MSS*.

Now, we observe that it might be preferable to assign on the machine having the largest completion time (i.e. machine M_1), a subset of jobs having short processing times (or equivalently, a maximal number of jobs). In this way, in a subsequent iteration it would be easier to reassign these short jobs. Let $Q \subset \{0, 1\}^{|\tilde{J}|}$ denote the set of optimal solutions of the *SSP* model. Instead of selecting *any* optimal solution $y \in Q$, we solve the following problem:

$$\text{Find } y \in Q \text{ such that } \sum_{j=1}^{|\tilde{J}|} y_j \text{ is minimal.} \tag{P}$$

Define a modified processing time $\pi_j = |\tilde{J}|p_j - 1$ for $j \in \tilde{J}$. Consider the following knapsack problem (KP):

$$\text{KP} : \text{Maximize} \sum_{j \in \tilde{J}} \pi_j y_j \tag{4}$$

subject to (2) and (3).

Lemma 1 *If $y^* \in \{0, 1\}^{|\tilde{J}|}$ is an optimal solution of (KP), then y^* solves (P).*

Proof. First, we show that if y^* is an optimal solution of (KP) then $y^* \in Q$.

Obviously, y^* satisfies (2)–(3). Assume that $\bar{y} \in Q$ and $\sum_{j\in\tilde{J}} p_j\bar{y}_j > \sum_{j\in\tilde{J}} p_j y_j^*$. We have $\sum_{j\in\tilde{J}} \pi_j\bar{y}_j - \sum_{j\in\tilde{J}} \pi_j y_j^* = |\tilde{J}|(\sum_{j\in\tilde{J}} p_j\bar{y}_j - \sum_{j\in\tilde{J}} p_j y_j^*) + (\sum_{j\in\tilde{J}} y_j^* - \sum_{j\in\tilde{J}} \bar{y}_j)$. Since $\sum_{j\in\tilde{J}} p_j\bar{y}_j - \sum_{j\in\tilde{J}} p_j y_j^* \geqslant 1$ and $\sum_{j\in\tilde{J}} y_j^* - \sum_{j\in\tilde{J}} \bar{y}_j \geqslant 1 - (|\tilde{J}| - 1)$, then $\sum_{j\in\tilde{J}} \pi_j\bar{y}_j - \sum_{j\in\tilde{J}} \pi_j y_j^* > 0$, which contradicts the hypothesis that y^* is an optimal solution of (KP) Thus, $y^* \in Q$.

Second, we show that $\sum_{j=1}^{|\tilde{J}|} y_j^*$ is minimal. Assume that $\bar{y} \in Q$ and $\sum_{j\in\tilde{J}} \bar{y}_j < \sum_{j\in\tilde{J}} y_j^*$. Because $\sum_{j\in\tilde{J}} \pi_j y_j^* \geqslant \sum_{j\in\tilde{J}} \pi_j\bar{y}_j$ then $|\tilde{J}|(\sum_{j\in\tilde{J}} p_j y_j^* - \sum_{j\in\tilde{J}} p_j\bar{y}_j) + (\sum_{j\in\tilde{J}} \bar{y}_j - \sum_{j\in\tilde{J}} y_j^*) \geqslant 0$. Moreover, $\sum_{j\in\tilde{J}} p_j\bar{y}_j = \sum_{j\in\tilde{J}} p_j y_j^*$, thus we get $\sum_{j\in\tilde{J}} \bar{y}_j - \sum_{j\in\tilde{J}} y_j^* \geqslant 0$ which contradicts the assumption. Thus, $\sum_{j\in\tilde{J}} \bar{y}_j \geqslant \sum_{j\in\tilde{J}} y_j^* \quad \forall \bar{y} \in Q$. ■

Hence, (*P*) can be solved in pseudopolynomial time by solving a knapsack problem. Consequently, we propose a new variant of *MSS* where at each iteration a KP instead of a SSP is solved. It is well known that KP could be efficiently solved in pseudopolynomial time (see Martello et al. (1999) and Chapter 5 of Kellerer et al. (2004)). In our implementation, we have solved the KPs using Pisinger's code (available at http://www.diku.dk/∼pisinger/.). Obviously, if an improvement is achieved, then a test is carried out in order to check whether the current solution is proven optimal (i.e. equal to a lower bound). In this case, the search procedure is stopped before reaching the maximal number of iterations *n_iter*. In our experiments, we set *n_iter* = 500. We have carried out exhaustive computational experiments and have found that the new proposed KP-based

heuristic outperforms *MSS* and makes it possible to deliver new improved solutions to hard benchmark instances (the details are provided in Section 5). At this point, it is noteworthy that as the KP is $\mathcal{NP}$-hard, we might expect that an exact approach would fail in some instances. However, we have observed that all of the thousands of KP instances that were generated in our computational study were optimally solved within very short CPU times using Pisinger's code.

Hereafter, we refer to this heuristic as the multi-start knapsack-based improvement heuristic (*MSK*).

3. Deriving enhanced lower bounds

In Haouari et al. (2006), a general lifting procedure for enhancing $P||C_{\max}$ lower bounds has been proposed. This procedure has been shown to yield lower bounds that outperform previously proposed bounds. For the sake of completeness, we briefly recall the basic idea of the lifting procedure (here again, we refer to Haouari et al., 2006 for a detailed description). Define $J_l \subseteq J$ as the subset of jobs that contains the l jobs of J with largest processing times, and $J_l^k \subseteq J_l$ as the subset of jobs that is obtained by considering the $\lambda_k(l) = k\lfloor l/m \rfloor + \min(k, l - \lfloor l/m \rfloor m)$ jobs of J_l with the smallest processing times. Also, let $L(J_l^k)$ denote a lower bound on the makespan of the auxiliary $P||C_{\max}$ instance I_k $(1 \leqslant k \leqslant m)$ that is defined on k machines and the jobset J_l^k. Hence, the lifted version of the lower bound $L(J_l^k)$ is

$$\tilde{L}(J) = \max_{1 \leqslant k \leqslant m} \left\{ \max_{1 \leqslant l \leqslant n} L(J_l^k) \right\} \tag{5}$$

Haouari et al. (2006) provide empirical evidence that this lifting procedure yields tight lower bounds. In particular, the lifted version of a bin-packing-based lower bound (denoted by L_{FS}) has a very good performance.

Now, we show that, based on a simple observation, it is still possible to derive stronger lower bounds in the following way. Given a lower bound L, we can derive a possibly better valid lower bound $\hat{L}$ by solving the following problem:

$$\hat{L} = \text{Min} \sum_{j \in J} p_j x_j \tag{6}$$

subject to:

$$\sum_{j \in J} p_j x_j \geqslant L \tag{7}$$

$$x_j \in \{0, 1\}, \quad \forall j \in J \tag{8}$$

By setting $y_j = 1 - x_j \cdot \forall j \in J$, the problem defined by (6)–(8) could easily be reformulated as a SSP and is therefore solvable in pseudopolynomial time. Hereafter, we refer by $\hat{L}_\xi$ to the lower bound that is obtained after enhancing a lower bound L_ξ (for e.g., $L_\xi \in \{L_{TV}, L_{FS}\}$) through the SSP-based enhancement procedure.

In order to obtain strong lower bounds, we have combined the SSP-based enhancement procedure with the lifting procedure by computing for a given lower bound $L(.)$ an enhanced lifted

bound $\vec{L}(.)$ defined by

$$\vec{L}(J) = \max_{1 \leqslant k \leqslant m} \left\{ \max_{1 \leqslant l \leqslant n} \hat{L}(J_l^k) \right\} \tag{9}$$

Hence, $\vec{L}(.)$ requires computing for each auxiliary instance I_k $(1 \leqslant k \leqslant m)$ a lower bound $L(J_l^k)(1 \leqslant l \leqslant n)$ and then solving an SSP to derive an enhanced lower bound $\hat{L}(J_l^k)$.

Our computational experiments provide evidence that even the best-lifted lower bound (namely the bound that is denoted by $\tilde{L}_{FS}$ in Haouari et al. (2006)) could be improved using (9).

Example. Consider the 8 jobs–3 machines instance with the following processing times: {40, 41, 46, 71, 85, 86, 88, 92}. We have $L_{TV} = \max(92,\ 86 + 85,\ \lceil \frac{549}{3} \rceil) = 183$. After applying the lifting procedure, we obtain $\tilde{L}_{TV} = 185$. This value is obtained by considering the values $k = 2$ and $l = 8$, which yield $\lambda_2(8) = 2\lfloor 8/3 \rfloor + \min(2, 8 - 3\lfloor 8/3 \rfloor) = 6$. Thus, $J_8^2 = \{40, 41, 46, 71, 85, 86\}$ and

$$L_{TV}(J_8^2) = \max\left(86,\ 71 + 85,\ \left\lceil \frac{40 + 41 + 46 + 71 + 85 + 86}{2} \right\rceil\right) = 185$$

$L_{TV}(J_8^2)$ could be enhanced by solving the following SSP:

$$\hat{L}_{TV}(J_8^2) = \min\ 40x_1 + 41x_2 + 46x_3 + 71x_4 + 85x_5 + 86x_6$$

subject to:

$$40x_1 + 41x_2 + 46x_3 + 71x_4 + 85x_5 + 86x_6 \geqslant 185,$$

$$x_j \in \{0, 1\}, \quad \forall j \in \{1, \ldots, 6\}.$$

The optimal solution of this problem is $x_1 = 1$, $x_4 = 1$, $x_5 = 1$, $x_2 = 0$, $x_3 = 0$, and $x_6 = 0$. The value of this solution is $\vec{L}_{TV} = \hat{L}_{TV}(J_8^2) = 196$. It is noteworthy that, for this instance, we have $\tilde{L}_{FS} = 185$. Hence, this example shows the remarkable result that even the trivial bound L_{TV} could yield an enhanced lower bound $\vec{L}_{TV}$ that dominates $\tilde{L}_{FS}$.

4. An exact branch-and-bound algorithm

The $P||C_{\max}$ is known to be $\mathcal{NP}$-hard in the strong sense and while several efforts have been devoted so far to designing efficient approximate solutions, the research on exact approaches has remained rather scant. A first contribution in this field has been achieved by Rothkopf (1966), who proposed a dynamic programming procedure that solves problem $P||C_{\max}$ in $O(nC^m)$ where C denotes an upper bound on the optimal makespan. Clearly, this latter exponential algorithm could only solve instances having very few machines and a small value of C. However, an effective branch-and-bound algorithm has been proposed by Dell'Amico and Martello (1995). This branch-and-bound constitutes an algorithmic breakthrough as it provides evidence that large $P||C_{\max}$ can be solved exactly. Recently, Mokotoff (2004) has formulated $P||C_{\max}$ as a mixed-integer program (MIP) and solved it using a cutting-plane scheme based on the generation of valid inequalities. However, computational experiments carried out by Dell'Amico and Martello (2005) demonstrate that this latter MIP-based approach is consistently outperformed by Dell'Amico and Martello's combinatorial branch-and-bound algorithm. We describe the components of a new

branch-and-bound algorithm that not only embeds the newly proposed lower and upper bounds but also includes a new solution representation and branching scheme.

4.1. Solution representation

We propose to represent a feasible $P||C_{\max}$ schedule as a permutation of the n jobs. A similar representation has been described recently by Haouari and Jemmali (2007) for the problem of maximizing the minimum completion time on identical parallel machines. This new solution encoding is based on the concept of symmetry-breaking (or -defeating), which has been previously implemented in integer programming for improving discrete model representations (see Sherali and Smith, 2001). Before describing this representation, we first observe that given a feasible $P||C_{\max}$ schedule S, a set $\Gamma(S)$ containing several equivalent symmetric schedules can be obtained by simply reindexing the identical machines (clearly, $|\Gamma(S)| = m!$). Moreover, if the instance includes some indistinguishable jobs (i.e. having identical processing times), then additional equivalent solutions might be generated through the interchange of similar jobs. Consequently, and as noticed by Sherali and Smith (2001), "the branch-and-bound algorithm can get hopelessly mired by being forced to explore and fathom symmetric reflections of various solutions during the search process". In order to reduce the computational burden that might be caused by the natural symmetry inherent in $P||C_{\max}$, we propose to represent each set of alternative symmetric solutions by a *unique* permutation of the n jobs. To this end, 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 $n_i = |J_i|$ job indices (recall that J_i refers to the subset of jobs that are assigned to M_i). 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) Each subsequence σ_i ($i = 1, \ldots, m$) is a nondecreasing list of the job indices. Thus, $\sigma_i^k < \sigma_i^{k+1}$, for $k = 1, \ldots, n_i - 1$, $i = 1, \ldots, m$.
(C2) The machines are indexed according to nonincreasing completion times.
(C3) If two machines M_i and M_{i+1} have the same completion time (i.e. $C_i = C_{i+1}$), then $\sigma_i^1 < \sigma_{i+1}^1$.
(C4) If two jobs j and $j+1$ have the same processing time and j is assigned to a machine M_α, then $j+1$ is necessarily assigned to a machine M_β such that $\beta \geqslant \alpha$. Thus, job $j+1$ is sequenced after job j in σ.

Example. Consider the instance with $n = 6$, $m = 3$, $p_1 = 9$, $p_2 = 6$, $p_3 = 6$, $p_4 = 5$, $p_5 = 4$, $p_6 = 3$. An application of the LPT rule yields the following job assignment: $J_1 = \{1, 6\}$, $J_2 = \{2, 4\}$, and $J_3 = \{3, 5\}$. The completion times of machines M_1, M_2, and M_3 are 12, 11, and 10, respectively. Clearly, a first symmetric solution can be obtained simply by interchanging the jobs assigned to machines M_1 and M_2, respectively. This yields the solution $J_1 = \{2, 4\}$, $J_2 = \{1, 6\}$, and $J_3 = \{3, 5\}$. Thus, in this way 3! symmetric solutions could be obtained by reindexing the three identical machines. In addition, we observe that as jobs 2 and 3 have the same processing time, then we can obtain an additional symmetric solution by interchanging these two identical jobs. This yields the following solution: $J_1 = \{1, 6\}$, $J_2 = \{3, 4\}$, and $J_3 = \{2, 5\}$. Consequently, by combining the symmetric solutions that could be obtained by reindexing the machines as well as those obtained by interchanging identical jobs, we can obtain as many as twelve symmetric solutions. These

Table 1
Set of symmetric solutions that are represented by $\sigma = (1, 6, 2, 4, 3, 5)$

Solution	J_1	J_2	J_3
S1	{1, 6}	{2, 5}	{3, 4}
S2	{1, 6}	{3, 4}	{2, 5}
S3	{2, 5}	{1, 6}	{3, 4}
S4	{3, 4}	{1, 6}	{2, 5}
S5	{2, 5}	{3, 4}	{1, 6}
S6	{3, 4}	{2, 5}	{1, 6}
S7	{1, 6}	{3, 5}	{2, 4}
S8	{1, 6}	{2, 4}	{3, 5}
S9	{3, 5}	{1, 6}	{2, 4}
S10	{2, 4}	{1, 6}	{3, 5}
S11	{3, 5}	{2, 4}	{1, 6}
S12	{2, 4}	{3, 5}	{1, 6}

solutions are listed in Table 1. However, in our branch-and-bound, there is a unique permutation that represents all these twelve symmetric solutions and satisfies conditions (C1) – (C4). This permutation is $\sigma = \sigma_1\ \sigma_2\ \sigma_3$ where $\sigma_1 = (1, 6)$, $\sigma_2 = (2, 4)$, and $\sigma_3 = (3, 5)$.

4.2. Branching scheme and data representation

We have implemented a new branching strategy that amounts to *sequentially* loading machines $M_1, M_2, \ldots, M_m$, in that order, while taking heed of Conditions (C1) – (C4). Hence, this branching strategy differs from that implemented in Dell'Amico and Martello (1995) and where the machines are loaded *simultaneously*.

The root node N_0 corresponds to the empty permutation and each node N_l of level l ($l \geqslant 1$) of the search tree corresponds to a partial valid permutation $\sigma(N_l) = (\sigma_1, \sigma_2, \ldots, \sigma_l)$ of l jobs. Each child of node N_l is derived by appending an unscheduled job $j_0 \in \bar{J}(N_l) = J \backslash \{\sigma_1, \sigma_2, \ldots, \sigma_l\}$ to $\sigma(N_l)$ and thus obtaining an extended partial permutation $(\sigma_1, \sigma_2, \ldots, \sigma_l, j_0)$.

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 $M_{i(N)}$.
- $a(N)$: lower bound on the total workload of machine $M_{i(N)}$.
- $b(N)$: upper bound on the total 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 candidates to be scheduled on $M_{i(N)}$.

The data associated with the root node N_0 are: $\sigma(N_0) = \emptyset$, $J(N_0) = \emptyset$, $i(N_0) = 1$, $L(N_0) = 0$, $C_1(N_0) = 0$, $b(N_0) = UB - 1$ (where UB denotes the value of the heuristic solution), $a(N_0) = LB$

(where LB denotes the value of a lower bound that 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 $b(N)$: As only valid permutations are generated, then from conditions (C2) and (C3), we obtain

$$b(N): \begin{cases} UB - 1, & \text{if } i(N) = 1 \\ C_{i(N)-1,} & \text{if } i(N) > 1 \text{ and } j_1(N) > j_0(N) \\ C_{i(N)-1} - 1, & \text{if } i(N) > 1 \text{ and } j_1(N) < j_0(N) \end{cases}$$

Computation of $a(N)$: A lower bound on the total workload of $M_{i(N)}$ is computed by considering a reduced $P||C_{\max}$ instance defined on $m' = m - i(N)+1$ machines and $n' = |J\backslash J(N)| + 1$ jobs. The jobset comprises the unscheduled jobs as well as a dummy job $n'+1$ having a processing time equal to $L(N)$.

Computation of $\bar{J}(N)$: For each $j \in J\backslash J(N)$, let $J_j = \{h \in J\backslash J(N) : h \geqslant j\}$ and $\rho_j = \sum_{h \in J_j} p_h$. The set of unscheduled jobs that are candidates to be scheduled on $M_{i(N)}$ immediately after $j(N)$ is

$$\bar{J}(N) = \{j \in J\backslash J(N) : j > j(N),\ p_j + L(N) \leqslant b(N),\ \text{and } L(N) + \rho_j \geqslant a(N)\} \tag{10}$$

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

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

Case (ii): $\bar{J}(N) = \emptyset$: in this case j is assigned to a new machine $M_{i(N)+1}$. Therefore, the data of N^+ are $\sigma(N^+) = \sigma(N)j$, $J(N^+) = J(N) \cup \{j\}$, $i(N^+) = i(N) + 1$, $L(N^+) = p_j$, $C_1(N^+) = C_1(N)$, $b(N^+) = b(N) - \delta$, 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, $a(N^+)$ 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, job j is assigned to machine M_1. We set $C_1(N^+) = L(N) + p_j$. The remaining data are updated similar to case (i).

Case (iv): $L(N) \geqslant a(N)$ and $j \notin \bar{J}(N)$: in this case job j is assigned to machine M_2. This situation is similar to case (ii).

Case (v): $L(N) \geqslant a(N)$ and $j \in \bar{J}(N)$: in this case, job j is either assigned to machine M_1 or to machine M_2. 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 similar to case (iii) and case (ii), respectively.

4.3. Node pruning

Obviously, whenever a solution having a maximum completion time satisfying $C_{\max} < UB$ is found, the incumbent value is updated and all the active nodes N having $C_1(N) \geqslant UB$ are pruned. In addition, a node N is pruned if any of the following conditions holds:

- $b(N) < a(N)$
- $i(N) = m - 2$ and $\bar{J}(N) = \emptyset$.

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 $m-2$ machines are loaded, then the unscheduled jobs are necessarily assigned to the remaining unloaded machine M_{m-1} and M_m. In this case, an SSP is solved in order to solve the residual $P2||C_{\max}$ optimally. Let C denote the makespan of the complete schedule. The incumbent value is updated by setting $UB = \min(UB, C)$.

4.4. Implemented lower and upper bounds

At the root node, we start by computing the trivial lower bound L_{TV} as well as the simple upper bound delivered by the LPT rule. If these two bounds are equal, then the algorithm is stopped. Otherwise, we successively run the *MSK* heuristic in order to obtain a tight initial upper bound U_{MSK}, and the enhanced lifted bin-packing-based lower bounding procedure for computing an initial lower bound $\vec{L}_{FS}$. While this latter lower bound is very often extremely tight, it is, however, rather time-consuming. Therefore, at each non-root node, the enhanced version of the lifted trivial lower bound $\vec{L}_{TV}$ is computed. This latter lower bound has a very good performance and can be computed very efficiently.

5. Computational results

We have coded our branch-and-bound algorithm in Microsoft Visual C++ (Version 6.0) and have run it on a Pentium IV 3.2 GHz Personal Computer with 1 GB RAM.

5.1. Performance on Dell'Amico and Martello's instances

First, we have run our branch-and-bound algorithm on a class of *perfect packing* instances that have been randomly generated as indicated in Dell'Amico and Martello (1995). This class comprises 1520 instances where the processing times are generated from an interval $[1, Q]$ (with $Q \in \{50, 100, 200, 400\}$) in such a way that the optimal schedule has an equal completion time on each machine. The results are displayed in Table 2. In this table, we have provided for each combination (n, m) and for each class the mean number of explored nodes (*NN*) as well as the mean CPU time in seconds (*Time*), computed over 10 problem instances. We observe that all the instances were solved to optimality. At this point, it is worth mentioning that Dell'Amico and

Table 2
Performance of the branch-and-bound algorithm on the class of perfect packing instances

		$Q = 50$		$Q = 100$		$Q = 200$		$Q = 400$	
n	*m*	*NN*	*Time*	*NN*	*Time*	*NN*	*Time*	*NN*	*Time*
10	3	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–
25	3	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–
	10	1	0.010	1	0.027	1	0.027	1	0.008
	15	1	0.003	1	0.002	1	0.001	1	0.004
50	3	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–
	10	1	–	1	–	1	0.001	1	0.004
	15	1	0.003	1	0.010	75	0.664	110	0.766
100	3	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–
	10	1	–	1	–	1	–	1	–
	15	1	–	1	–	1	–	1	0.001
250	3	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–
	10	1	–	1	–	1	–	1	–
	15	1	–	1	–	1	–	1	–
500	3	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–
	10	1	–	1	–	1	–	1	–
	15	1	–	1	–	1	–	1	–
1000	3	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–
	10	1	–	1	–	1	–	1	–
	15	1	–	1	–	1	–	1	–
2500	3	1	0.001	1	0.001	1	0.001	1	0.001
	5	1	0.001	1	0.001	1	0.001	1	0.001
	10	1	0.001	1	0.001	1	0.001	1	0.001
	15	1	0.001	1	0.001	1	0.001	1	0.002
5000	3	1	0.003	1	0.003	1	0.003	1	0.003
	5	1	0.003	1	0.003	1	0.003	1	0.003
	10	1	0.003	1	0.003	1	0.003	1	0.003
	15	1	0.003	1	0.003	1	0.004	1	0.004
10,000	3	1	0.011	1	0.011	1	0.011	1	0.011
	5	1	0.011	1	0.012	1	0.013	1	0.011
	10	1	0.014	1	0.014	1	0.013	1	0.013
	15	1	0.013	1	0.014	1	0.013	1	0.013

–, average CPU time is <0.001 s.

Martello (1995) report that their branch-and-bound algorithm, which has been run on a Digital VAXstation 3100, failed to solve 13 instances. Interestingly, we conclude from Table 2 that the implemented bounds are very tight, as almost all instances were solved at the root node. Indeed, we found that branching was required for only three instances out of 1520.

Moreover, we have run our algorithm on five additional problem classes that have been randomly generated as described in Dell'Amico and Martello (1995). 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 standard deviation 50.
- Class 5: normal distribution with mean 100 and standard deviation 20.

The number of jobs ranged between 10 and 10,000, and the number of machines ranged between 2 and 15. For each class, and each (n, m) combination, 10 instances were generated. Hence, a total of 1900 instances have been generated.

A summary of the results is displayed in Table 3. We observe that all instances, except five, were solved to optimality (we set the maximum CPU time to 1200 seconds). Surprisingly, the five unsolved instances are relatively small ($n = 50, m = 15$). These lattermost results are consistent with those reported by Dell'Amico and Martello (1995). Indeed, their branch-and-bound algorithm failed to solve eight small-sized instances. Moreover, we see that the proposed bounds U_{MSK} and $\vec{L}_{FS}$ exhibit a very good performance because here again most instances were optimally solved at the root node, and branching was required for only 77 instances out of 1900.

5.2. Performance on benchmark instances

In order to test our algorithm on harder problems, we considered the benchmark instances proposed by França et al. (1994), which include 390 uniform distribution instances, and those proposed by Frangioni et al. (2004), which include 390 non-uniform distribution instances. The results are displayed in Table 4. In this table, in addition to the mean number of explored nodes (*NN*) and the mean CPU time in seconds (*Time*), we provide *Gap* and *Max_Gap*, which represent the average and maximum percentage deviation (computed over 10 problem instances) between U_{MSK} and $\vec{L}_{FS}$, respectively. From this table, we see that our algorithm produced proven optimal solutions for 759 instances. Here again, branching was scarcely required. Interestingly, the algorithm produced optimal solutions for 13 instances that have been open for some time. The detailed results for these 13 instances are shown in Table 5.

The set of unsolved instances includes three non-uniform distribution instances and 18 uniform distribution instances. However, among these 18 latter hard instances, the *MSK* heuristic produced new improved upper bounds for six instances, inferior solutions for six instances, and similar solutions for six instances. Moreover, for all these 18 instances, $\vec{L}_{FS}$ was found equal to the best-known lower bound. The details are displayed in Table 6. In this table $L_{old} \equiv max(L_{AR}, \tilde{L}_{FS})$ and $U_{old} \equiv \min(U_{AR}, U_{MSS})$ refer to the best-known lower and upper bounds, respectively.

Actually, this very good performance is largely due to the tightness of the proposed bounds. In Table 7, we report the results of pairwise comparisons between $\vec{L}_{FS}$ and L_{AR} and $\tilde{L}_{FS}$, respectively. We see that for all the 780 instances, $\vec{L}_{FS}$ is either strictly better than or equal to L_{AR} and $\tilde{L}_{FS}$, respectively.

Table 3
Performance of the branch-and-bound algorithm on Dell'Amico and Martello's instances

		Class 1		Class 2		Class 3		Class 4		Class 5	
n	*m*	*NN*	*Time*	*NN*	*Time*	*NN*	*Time*	*NN*	*Time*	*NN*	*Time*
10	3	150	0.094	127	0.093	142	0.097	131	0.093	136	0.101
	5	1	0.065	174	0.156		0.129	225	0.172	1	0.089
25	3	1	–	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–	1	–
	10	4,750,793	36.828	14,814,146	121.328	383,810	3.648	1,434,262	12.816	1,124,031	11.005
	15	1	0.128	1,082,607	12.177	1	0.611	1	0.140	1	0.668
50	3	1	–	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–	1	–
	10	1	–	1	–	1	–	1	0.001	1	–
	15	149	0.891	1	0.009	1	0.142	218	0.891 (7)	1	0.277 (8)
100	3	1	–	1	–	1	–	1	–	1	–
	5	1	–	1	–	1	–	1	–	1	–
	10	1	–	1	–	1	–	1	0.001	1	0.001
	15	1	0.001	1	0.001	1	0.002	1	0.002	1	0.002
250	3	1	–	1	–	1	0.001	1	–	1	0.002
	5	1	–	1	–	1	–	1	0.001	1	0.001
	10	1	–	1	–	1	–	1	0.002	1	0.002
	15	1	0.001	1	0.002	1	0.004	1	0.002	1	0.004
500	3	1	–	1	0.003	1	0.004	1	–	1	0.003
	5	1	–	1	–	1	–	1	–	1	0.004
	10	1	–	1	–	1	–	1	0.002	1	0.005
	15	1	–	1	0.004	1	0.008	1	0.003	1	0.010
1000	3	1	–	1	0.008	1	0.014	1	–	1	0.016
	5	1	–	1	–	1	–	1	–	1	0.012
	10	1	–	1	–	1	–	1	0.001	1	0.014
	15	1	0.001	1	0.015	1	0.017	1	0.006	1	0.017
2500	3	1	0.001	1	0.023	1	0.032	1	0.003	1	0.034
	5	1	0.001	1	0.001	1	0.001	1	0.001	1	0.021
	10	1	0.002	1	0.001	1	0.001	1	0.001	1	0.021
	15	1	0.001	1	0.028	1	0.035	1	0.001	1	0.032
5000	3	1	0.004	1	0.075	1	0.101	1	0.004	1	0.089
	5	1	0.004	1	0.004	1	0.004	1	0.004	1	0.071
	10	1	0.004	1	0.004	1	0.004	1	0.004	1	0.064
	15	1	0.004	1	0.087	1	0.123	1	0.005	1	0.103
10,000	3	1	0.016	1	0.240	1	0.318	1	0.015	1	0.245
	5	1	0.016	1	0.015	1	0.015	1	0.015	1	0.167
	10	1	0.016	1	0.015	1	0.015	1	0.015	1	0.170
	15	1	0.015	1	0.311	1	0.381	1	0.014	1	0.284

–, average CPU time is <0.001 s.
The figures in brackets indicate the number of solved instances, if <10.

Moreover, we report in Table 8 the results of pairwise comparisons between U_{MSK} and U_{AR} and U_{MSS}, respectively. We see that MSK consistently outperforms MSS. Moreover, there are only 10 instances for which $U_{MSK} > U_{AR}$. For the remaining instances, we observe that U_{MSK} is either strictly better than (19 instances) or equal to U_{AR}.

Table 4
Performance of the branch-and-bound algorithm on the benchmark instances

			Uniform instances				Non-uniform instances			
Interval	n	m	*Gap*	*Max_Gap*	*NN*	*Time*	*Gap*	*Max_Gap*	*NN*	*Time*
[1, 100]	10	5	0	0	1	0.004	0	0	1	0.010
	50	5	0	0	1	0.014	0	0	1	0.015
		10	0	0	1	0.039	0	0	1	0.018
		25	0.181	0.909	1	0.295 (8)	0	0	1	0.078
	100	5	0	0	1	0.010	0	0	1	0.017
		10	0	0	1	0.061	0	0	1	0.046
		25	0	0	1	0.228	0	0	1	0.093
	500	5	0	0	1	–	0	0	1	0.107
		10	0	0	1	0.041	0	0	1	0.175
		25	0	0	1	0.411	0	0	1	0.581
	1000	5	0	0	1	0.001	0	0	1	0.348
		10	0	0	1	0.001	0	0	1	0.422
		25	0	0	1	0.001	0	0	1	0.928
[1, 1000]	10	5	0	0	1	0.006	0	0	1	0.013
	50	5	0	0	1	0.018	0	0	1	0.018
		10	0	0	1	0.076	0	0	1	0.038
		25	0.310	2.198	1	0.669 (8)	0	0	1	0.278
	100	5	0	0	1	0.031	0	0	1	0.059
		10	0	0	1	0.071	0	0	1	0.095
		25	0.010	0.051	14,364	1.278	0	0	1	0.312
	500	5	0	0	1	0.084	0	0	1	0.475
		10	0	0	1	0.195	0	0	1	0.634
		25	0	0	1	0.837	0	0	1	1.148
	1000	5	0	0	1	0.162	0	0	1	1.574
		10	0	0	1	0.328	0	0	1	2.117
		25	0	0	1	1.206	0	0	1	2.832
[1, 10,000]	10	5	0	0	1	0.009	0	0	1	0.018
	50	5	0	0	1	0.037	0	0	1	0.064
		10	0.004	0.007	23,607,013	146.411 (7)	0	0	1	0.332
		25	0.030	0.300	1	0.369 (9)	0	0	1	0.417
	100	5	0	0	1	0.054	0	0	1	0.820
		10	0	0	1	0.117	0	0	1	0.265
		25	0.028	0.034		(0)	0.002	0.003	10,399,656	246.803 (7)
	500	5	0	0	1	0.384	0	0	1	7.412
		10	0	0	1	0.585	0	0	1	9.140
		25	0	0	1	1.481	0	0	1	9.664
	1000	5	0	0	1	1.439	0	0	1	29.414
		10	0	0	1	2.046	0	0	1	36.964
		25	0	0	1	3.698	0	0	1	43.501

–, average CPU time is less than 0.001 s.
The figures in brackets indicate the number of solved instances, if <10.

5.3. Is $P||C_{max}$ an "easy" $\mathcal{NP}$-hard problem?

In summary, we ran our branch-and-bound algorithm on a total of 4200 instances, and we found that 4174 instances (99.38%) were optimally solved. Hence, our computational experiments raise

Table 5
Optimal solutions of benchmark instances

	Interval	n	m	Instance	L_{old}	U_{old}	C^*_{max}	NN	$Time$
Uniform	[1, 1000]	100	25	3	1941	1942	1941	140,710	4.032
instances	[1, 10,000]	10	5	5	10,789	10,828	10,828	1	0.015
				7	11,385	11,575	11,575	1	0.015
		50	10	1	26,662	26,663	26,662	4,788,878	31.344
				3	23,674	23,675	23,674	82,100,804	505.672
				5	27,205	27,207	27,205	1	0.922
				6	25,296	25,298	25,296	1,312,001	8.140
				8	25,479	25,481	25,479	71,676,209	439.453
				9	32,266	32,270	32,266	4,741,157	34.328
				10	26,707	26,709	26,707	630,045	5.016
Non-uniform	[1, 10,000]	50	10	10	47,336	47,337	47,336	1	0.188
instances		100	25	2	37,881	37,882	37,881	4,933,732	207.704
				3	37,668	37,669	37,668	6,336,782	233.734

Table 6
New improved upper bounds for benchmark instances

	Interval	n	m	Instance	L_{old}	U_{old}	U_{MSK}	$\vec{L}_{FS}$
Uniform instances	[1, 100]	50	25	4	110	111	111	110
				5	111	111	112	111
	[1, 1000]	50	25	4	1092	1105	1109	1092
				9	995	995	1004	995
	[1, 10,000]	50	10	2	26,233	26,235	26,235	26,233
				4	27,764	27,767	**27,765**	27,764
				7	23,388	23,389	23,389	23,388
			25	2	9659	9688	9688	9659
		100	25	1	21,169	21,174	21,175	21,169
				2	17,197	17,203	17,203	17,197
				3	21,572	21,578	**21,577**	21,572
				4	20,842	20,850	**20,847**	20,842
				5	20,568	20,576	**20,575**	20,568
				6	20,695	20,704	**20,702**	20,695
				7	20,021	20,026	20,026	20,021
				8	19,272	19,276	19,277	19,272
				9	20,598	20,604	**20,602**	20,598
				10	19,124	19,129	19,130	19,124
Non-uniform instances	[1, 10,000]	100	25	1	37,881	37,882	37,882	37,881
				4	37,929	37,930	37,930	37,929
				8	37,942	37,943	37,943	37,942

Figures in bold indicate new improved solutions.

the legitimate question of whether problem $P||C_{\max}$ could be considered as rather easy from the practical computational point of view, and whether the performance of our algorithm would deteriorate on some problem class. To this end, we have run our algorithm on the problem class where the number of machines m is set equal to $2n/5$ and the processing times are drawn from the

Table 7
Comparison between $\vec{L}_{FS}$ and L_{AR} and $\tilde{L}_{FS}$

$\vec{L}_{FS} > \tilde{L}_{FS}$	$\vec{L}_{FS} = \tilde{L}_{FS}$	$\vec{L}_{FS} < \tilde{L}_{FS}$	$\vec{L}_{FS} > \tilde{L}_{FS}$	$\vec{L}_{FS} = \tilde{L}_{FS}$	$\vec{L}_{FS} < \tilde{L}_{FS}$
2	778	0	15	765	0

Table 8
Comparison between U_{MSK} and U_{AR} and U_{MSS}

$U_{MSK} < U_{MSS}$	$U_{MSK} = U_{MSS}$	$U_{MSK} > U_{MSS}$	$U_{MSK} < U_{AR}$	$U_{MSK} = U_{AR}$	$U_{MSK} > U_{AR}$
28	751	1	19	751	10

Table 9
Performance of the branch-and-bound algorithm on the class where $m = 0.4\ n$

(n, m)	(10, 4)	(20, 8)	(30, 12)	(40, 16)	(50, 20)	(60, 24)	(70, 28)	(80, 32)	(90, 36)	(100, 40)	(150, 60)	(200, 80)
NN	1	456,470	1	1	1	9,609,214	1	403,164	1	1	1	1
Time	0.039	2.445	0.551	0.867	1.200	76.031	1.739	5	3.352	3.305	7.970	18.072
US	0	0	5	4	8	5	10	7	11	6	9	12

discrete uniform distribution on $[\frac{n}{5}, \frac{n}{2}]$. For each value of $n \in 10, 20, 30, 40, 50, 60, 70, 80, 100, 150, 200$, a set of 20 instances were randomly generated. A summary of the results that were obtained for this challenging class is displayed in Table 9. In this table, US represents the number of instances (out of 20) that remained unsolved after 1200 seconds.

We see that, except for small-sized instances ($n \leqslant 20$), the performance of our algorithm deteriorates as the number of unsolved instances is relatively high. These results suggest that an alternative approach would be more suited for this class of instances.

6. Conclusion

In this paper, we have proposed an improvement of the lifting procedure for generating very strong $P||C_{\max}$ lower bounds. Moreover, we have implemented a new enhanced KP-based heuristic, which was found to consistently yield optimal or very near-optimal solutions. These new lower and upper bounding strategies have been embedded into an exact branch-and-bound algorithm. This algorithm includes an additional distinctive feature that consists in a new symmetry-breaking-based solution representation of a schedule as a permutation of jobs. Computational results on a large set of 4430 instances attest to the tightness of the proposed lower and upper bounds and provide evidence of the efficacy of the branch-and-bound algorithm. In particular, we have reported the optimal solution of 13 hard benchmark instances that have been open for some time. Furthermore, we have provided new improved upper bounds for six additional benchmark instances.

Future research effort needs to be focused on the class of problems that has been characterized to be the most intractable. This class refers to instances with a large processing time range and a very large number of machines.

Acknowledgment

The authors would like to thank an anonymous referee for kindly suggesting the idea underlying the improvement of the *MSS* heuristic.

References

Alvim, A.C.F., Ribeiro, C.C., 2004. A hybrid bin packing heuristic to multiprocessor scheduling. In Ribeiro, C.C., Martins, S.L. (eds) *Lecture Notes in Computer Science*, Vol. 3059. Springer-Verlag, Berlin, pp. 1–13.

Dell'Amico, M., Martello, S., 1995. Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing* 7, 191–200.

Dell'Amico, M., Martello, S., 2005. A note on exact algorithms for the identical parallel machine scheduling problem. *European Journal of Operational Research* 160, 576–578.

França, P.M., Gendreau, M., Laporte, G., Müller, F.M., 1994. A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective. *Computers and Operations Research* 21, 205–210.

Frangioni, A., Necciari, E., Scutellà, M.G., 2004. A multi-exchange neighborhood for minimum makespan machine scheduling problems. *Journal of Combinatorial Optimization* 8, 195–220.

Haouari, M., Jemmali, M., 2007. Maximizing the minimum completion time on parallel machines. *40R A Quarterly Journal of Operations Research* (in press).

Haouari, M., Gharbi, A., Jemmali, M., 2006. Tight bounds for the identical parallel machine scheduling problem. *International Transactions in Operations Research* 13, 529–548.

Kellerer, H., Pferschy, U., Pisinger, D., 2004. *Knapsack Problems*. Springer, Berlin.

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D., 1993. Sequencing and Scheduling: Algorithms and Complexity. In Graves, S.S., Rinnooy Kan, A.H.G., Zipkin, P. (eds) *Handbooks in Operations Research and Management Science 4*, pp. 445–522.

Martello, S., Pisinger, D., Toth, P., 1999. Dynamic programming and strong bounds for the 0–1 knapsack problem. *Management Science* 45, 414–424.

Mokotoff, E., 2004. An exact algorithm for the identical parallel machine scheduling problem. *European Journal of Operational Research* 152, 758–769.

Rothkopf, M.H., 1966. Scheduling independent tasks on parallel processors. *Management Science* 12, 437–447.

Sherali, H.D., Smith, J.C., 2001. Improving discrete model representations via symmetry considerations. *Management Science* 47, 1396–1407.

Tang, L., Luo, J., 2006. A new ILS algorithm for parallel machine scheduling problems. *Journal of Intelligent Manufacturing* 17, 609–619.