Available online at www.sciencedirect.com

SCIENCE DIRECT®

European Journal of Operational Research 152 (2004) 758–769

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Production, Manufacturing and Logistics

An exact algorithm for the identical parallel machine scheduling problem

Ethel Mokotoff [*]

Department of Economics, Universidad de Alcalá, Plaza Victoria 3, 28802 Alcalá de Henares, Spain

Received 7 May 1999; accepted 21 August 2002

Abstract

The *NP-hard* classical deterministic scheduling problem of minimizing the makespan on identical parallel machines is considered. The polyhedral structure of the problem is analyzed, and strong valid inequalities are identified for fixed values of the makespan. Thus, partial linear descriptions of the associated polytope are obtained and used to build an exact algorithm. Linear programming relaxations are solved iteratively until the integer solution is reached. The algorithm includes a preprocessing phase which reformulates the problem and simplifies it. Different constructive rules have been tried out in the computation of upper bound values and the incidence of this choice is reported. Computational results show that the proposed algorithm yields, in a short computational time, optimal solutions in almost all tested cases.

Keywords: Optimization; Polyhedral combinatorics; Scheduling

1. Introduction

In the classical deterministic identical parallel machine problem, there are a number of independent jobs to be processed on a range of identical machines. Each job has to be carried out on one of the machines during a fixed processing time, without preemption. The problem of finding the schedule that optimizes the makespan is considered.

Where can this model be applied to real-life situations? There are many examples in industry, social sciences and computer science, where the problem of job-allocation and total completion time fit in with this model: production lines, patients in hospitals, share-time systems, to give but three examples. In fact, any situation that deals with a series of processing units and has a set of jobs to be carried out on them are real-life examples of this mathematical model.

This problem is known to be *NP-complete* since PARTITION polynomially reduces to the special case with two machines [5]. So over the years there has been a great deal of research to develop approximation algorithms which can be applied. The only way to secure optimal solutions is by enumerative techniques. Dynamic programming and branch and bound (B&B) techniques have been

[*] Tel.: +34-91-885-4202; fax: +34-91-885-4239.
E-mail address: ethel.mokotoff@uah.es (E. Mokotoff).

0377-2217/$ - see front matter © 2002 Elsevier B.V. All rights reserved.
doi:10.1016/S0377-2217(02)00726-9

used to find optimal solutions. (For a survey of parallel machine scheduling problems, see [7]).

Here we are presenting an exact cutting plane algorithm built from the identification of valid inequalities for fixed values of the maximum completion time. These valid inequalities have been successfully applied to develop an approximate algorithm [8] and corresponding ones have been applied to develop an exact and an approximate algorithm for the unrelated parallel machine case [9]. The algorithm is a cutting plane method, which includes a specially-developed preprocessing phase. From the analysis of applying different constructive rules we have refined the value of the upper bound. Computational results show that, in almost all tested cases, the problem can be solved exactly, and confirm the intuitive idea of it being much more efficient than the corresponding algorithm when it is applied to the unrelated parallel machines.

After briefly introducing the notation, assumptions and formulation, we present the algorithm with examples and the results obtained in our computational experiment. We conclude the paper with a summary discussion on the obtained results and further research directions.

1.1. Notation, assumptions and formulation

In the rest of the paper, we will use the following notation:

J_i	job i, $i \in N = \{1, \ldots, n\}$
M_j	machine j, $j \in M = \{1, \ldots, m\}$
p_i	processing time of job J_i
C_i	completion time of job J_i
$C_{\max}$	makespan, the maximum completion time of all jobs J_i, $C_{\max} = \max\{C_1, C_2, \ldots, C_n\}$
x_{ij}	assignment variable

Consider a set J of n independent and non-preemptive jobs, J_i, which must be carried out on a set M of m machines, M_j, which have the same speed, which we consider to be identical. Therefore, each job is to be carried out on any one of the machines and the processing times of each job, p_i, are not affected by the machine processing it. Independent jobs means that there is no precedence among them, and non-preemption requires that jobs must be completed once they have begun, without interruption. The aim is to find an assignment of the n jobs to machines from set M that minimizes the maximum completion time criterion, called *makespan.*

The problem in hand is then denoted by $P//C_{\max}$, where P represents identical parallel machines, the jobs are not constrained, and the objective is to produce the minimum length schedule.

A mixed integer programming (MIP) formulation of the minimum makespan problem is as follows:

$$\min \quad y$$

$$\text{s.t.} \quad \sum_{j=1}^{m} x_{ij} = 1, \quad 1 \leqslant i \leqslant n, \tag{1}$$

$$y - \sum_{i=1}^{n} p_i x_{ij} \geqslant 0, \quad 1 \leqslant j \leqslant m, \tag{2}$$

where the optimal value of y is $C_{\max}$ and

$$x_{ij} = \begin{cases} 1 & \text{if job } i \text{ is assigned to machine } j \\ 0 & \text{if job } i \text{ is not assigned to machine } j \end{cases}$$

2. Cutting plane scheme

The proposed algorithm (PA) presents a cutting plane scheme (see [4] for a survey of this field) built from the identification of valid inequalities that apply to the subset of the solutions constrained by a maximum value of the makespan. The main part of this algorithm consists in generating constraints with the valid inequalities identified for $P//C_{\max}$ in [8]. The separation procedure checks whether or not a given point, the optimum solution of the last linear programming (LP) relaxation solved, is an integral vector. If so, the algorithm stops, otherwise some new valid inequalities are added to truncate the last LP polyhedron.

For an algorithm such as that being presented the computation of fitting initial tight lower and upper bounds is critical. Because refined bounds may reduced the number of iterations, and consequently the computational time, remarkably. (In Section 4 we report about the benefit of computing the upper bound by means of the algorithms based on list scheduling [3]).

Since the number of binary variables and inequalities is rather large even for moderately sized multiple machine scheduling problems, we have implemented a special preprocessing procedure in order to keep the problem in a manageable size, as well as to tighten the lower bound value.

The solution set F for the stated problem is defined by (x, y)

$$\begin{cases} x \in \{0,1\}^{n\times m}, \\ y \in \mathbb{R}^+, \\ \sum_{j=1}^m x_{ij} = 1 & \forall i \in \{1,\dots,n\}, \\ y - \sum_{i=1}^n p_i x_{ij} \geqslant 0 & \forall j \in \{1,\dots,m\}. \end{cases}$$

The polyhedron P that defines the linear relaxation of F is obtained by substituting $x \in \{0,1\}^{n\times m}$ by $x \in \mathbb{R}_+^{n\times m}$. Due to the fact that

$$\min\{y : (x,y) \in F\} = \min\{y : (x,y) \in P,\ Ax + Dy \leqslant b\},$$

where $Ax + Dy \leqslant b$ are valid linear inequalities that together with the initial inequalities of P define the so-called linear description of F (see e.g. [11]), a method for identifying a valid inequality, which is not satisfied by the solution of the LP relaxation (should one exist), is the core of cutting plane techniques. Iteratively, valid inequalities are added and LP relaxations are solved, until a feasible solution to the combinatorial optimization problem is obtained. If, after a number of iterations without finding an integral solution, there is no new cut, the algorithm makes use of B&B to complete the search, taking the last solved LP relaxation as initial solution, which may be good enough to efficiently reduce the number of nodes to be explored in the B&B procedure.

2.1. Valid inequalities

For a value y^0 of y, we denote by $F(y^0)$ the subset of $(x, y^0) \in F$. Let $\mathscr{I}$ be a family of valid inequalities for $\text{conv}(F(y^0))$ and let $P(\mathscr{I})$ be the current relaxation of $\text{conv}(F(y^0))$ defined by the inequalities of P and the inequalities of $\mathscr{I}$. Let (x^0, y^0) be a point of $P(\mathscr{I})$ that does not belong to $F(y^0)$. We have to find a valid inequality I such that $\text{conv}(F(y^0)) \subset P(\mathscr{I} \cup \{I\})$ and $(x^0, y^0) \notin P(\mathscr{I} \cup \{I\})$. To manage it we have drawn inspiration from the ideas developed in [12] for the upper bound flow model.

For each machine M_j, let $S_j = \{i \in N | x_{ij}^0 > 0\}$ be the set of jobs assigned (at least in part) to machine M_j. If the machine constraint corresponding to M_j is satisfied at equality by (x^0, y^0), that constraint is said to be active. Let $\Delta_j = \sum_{i\in S_j} p_i - y^0$ be the excess charge of machine M_j and let $S_j' = \{k \in S_j | p_k > \Delta_j\}$ be the subset of the jobs in S_j whose processing time on M_j is greater than Δ_j. Notice that the excess charge of M_j is positive if $\sum_{i\in S_j} p_i > y^0$.

The following proposition shows that for every machine M_j, corresponding to an active constraint, with positive excess charge Δ_j, and such that $S_j' \neq \{\phi\}$, a valid inequality is generated.

For any $x \in \mathbb{R}$, we let x^+ denote the value $\max\{0, x\}$.

Proposition 1. *Let (x^0, y^0) be a point of $P(\mathscr{I})$ and assume that the machine constraint for M_j is active for (x^0, y^0)* [1]. *Then the linear inequality*:

$$\sum_{i\in S_j} p_i x_{ij} \leqslant y^0 - \sum_{i\in S_j} (p_i - \Delta_j)^+ (1 - x_{ij}) \qquad (3)$$

is a valid inequality for $\text{conv}(F(y^0))$. *Moreover if there is a job $J_k \in S_j'$ for which $x_k^0 < 1$, then the inequality is not satisfied by (x^0, y^0).*

Proof

1. Since there is a job $k \in S_j$ such that $p_k > \Delta_j$, and $0 < x_{kj}^0 < 1$,

$$\sum_{i\in S_j} (p_i - \Delta_j)^+ (1 - x_{ij}^0) > 0.$$

[1] This will occur when there are no added constraints. Then, at any iteration, after having added $h \geqslant 0$ constraints of type 3 (added constraints), the linear programming problem has $m + n + h$ constraints in addition to the non-negativity conditions. Therefore, an optimal LP solution has $m + n + h$ basic variables, including the slack variables corresponding to the constraints of type 2 and 3, which may take positive values while the other non-basic variables take the value zero. If the solution is not integral, more than n assigning variables take positive values. Thus, at least one constraint of type 2 or 3 are satisfied as equality.

Thus, if $\sum_{i\in S_j} p_i x_{ij}^0 = y^0$, the LP solution (x^0, y^0) does not satisfy the inequality (3).

2. Let (x^*, y^*) be a solution for which x^* is binary and $y^* = y^0$.

Let $B_j = \{i \in S_j / x_{ij}^* = 0\}$,

$$\sum_{i\in S_j} p_i x_{ij}^* = \sum_{i\in S_j\setminus B_j} p_i = \sum_{i\in S_j} p_i - \sum_{i\in B_j} p_i$$
$$= y^0 + \Delta_j - \sum_{i\in B_j} p_i = y^0 - \left(\sum_{i\in B_j} p_i - \Delta_j\right).$$

Since $\sum_{i\in S_j} p_i x_{ij}^*$ cannot be greater than $y^* = y^0$, it is necessary that $\sum_{i\in B_j} p_i - \Delta_j \geqslant 0$.

Thus, $\sum_{i\in S_j} p_i x_{ij}^* = y^0 - \left(\sum_{i\in B_j} p_i - \Delta_j\right)^+ \leqslant y^0 - \sum_{i\in B_j} (p_i - \Delta_j)^+$.

On the other hand, since $1 - x_{ij}^* = 1$ if $i \in B_j$, and $1 - x_{ij}^* = 0$ if $i \in S_j - B_j$, we have

$$y^0 - \sum_{i\in B_j} (p_i - \Delta_j)^+ = y^0 - \sum_{i\in B_j} (p_i - \Delta_j)^+ (1 - x_{ij}^*)$$
$$- \sum_{i\in S_j - B_j} (p_i - \Delta_j)^+ (1 - x_{ij}^*)$$
$$= y^0 - \sum_{i\in S_j} (p_i - \Delta_j)^+ (1 - x_{ij}^*).$$

Thus, we can conclude

$$\sum_{i\in S_j} p_i x_{ij}^* \leqslant y^0 - \sum_{i\in S_j} (p_i - \Delta_j)^+ (1 - x_{ij}^*). \qquad \square$$

2.1.1. Example

Consider the set J of six jobs, having the processing times $\{5,5,3,3,1,1\}$. They have to be processed by a set M of 2 identical machines.

Solving the LP relaxation the following solution is obtained:

$C_{\max} = 9$

$i\setminus j$	M_1	M_2
J_1	1	0
J_2	0	1
J_3	0.33	0.67
J_4	1	0
J_5	0	1
J_6	0	1
Δ	2	1

$S_1 = \{1,3,4\}$ and $S_2 = \{2,3,5,6\}$

$S_1' = \{1,3,4\}$ and $S_2' = \{2,3\}$

S_1 yields $5x_{11} + 3x_{31} + 3x_{41} \leqslant C_{\max} - [3(1 - x_{11}) + 1(1 - x_{31}) + 1(1 - x_{41})]$

S_2 yields $5x_{22} + 3x_{32} + 1x_{52} + 1x_{62} \leqslant C_{\max} - [4(1 - x_{22}) + 2(1 - x_{32})]$

Added these new constraints, the solution to the LP relaxation is:

$i\setminus j$	M_1	M_2
J_1	0	1
J_2	1	0
J_3	1	0
J_4	0	1
J_5	0	1
J_6	1	0
Δ	0	0

that is an integral vector. While the previous non-integral solution does not satisfy the valid inequalities, this integral vector does.

3. Proposed algorithm

The results of the previous sections have led us to derive an exact cutting plane algorithm for the $P//C_{\max}$ problem. The algorithm essentially consists in the iterative addition of valid inequalities, starting from the solution obtained by successive lineal programming relaxations. The computation of initial lower and upper bounds are crucial for the fast convergence of the algorithm, therefore we have focused attention on preprocessing and we have tried different approximative algorithms to compute upper bound values.

The first step is to sort the jobs according to

$p_1 \geqslant p_2 \geqslant, \ldots, \geqslant p_n.$

3.1. Lower bound

The most simple lower bound for $P//C_{\max}$ is the following solution value of the relaxation we obtain by assuming that preemption is allowed [6]

$$\mathrm{LB}(C_{\max}) = \max\left\{\frac{1}{m}\sum_{i=1}^{n} p_i; \max_i\{p_i\}\right\}.$$

This lower bound can be further improved by considering that

$$C_{\max} \geqslant p_m + p_{m+1}.$$

Then, it is possible to tighten the bound as follows:

$$\mathrm{LB}(C_{\max}) = \max\left\{\frac{1}{m}\sum_{i=1}^{n} p_i; \max_i\{p_i\}; p_m + p_{m+1}\right\}.$$

Later, this bound can only be increased when the linear program ensues unfeasible.

3.2. Upper bound

A feasible initial solution is built by means of an approximative algorithm (AA). The obtained value of $C_{\max}$ is taken as the upper bound, i.e., $\mathrm{UB}(C_{\max}) = C_{\max}(\mathrm{AA})$.

The classical longest processing time first (LPT) heuristic [2] consists in making a list according to LPT order and then, as with any list scheduling algorithm, assigning the uppermost job from the list to a free or to the least loaded machine, until the list is exhausted. The MultiFit algorithm takes into account the duality existing between the $P//C_{\max}$ and the bin packing problem [1]. The MultiFit algorithm performs better than the previous one by increasing the computational complexity. We have also implemented other more recent heuristics like those presented in [10] and [3] based on list scheduling, some of which provide similar results to MultiFit without pushing the computational effort so much, and even taking less time than LPT in many cases.

3.3. Preprocessing

In the preprocessing phase, an essential part of cutting plane algorithms, we try to reformulate the problem to narrow the difference between the value of the objective function in the LP relaxation and that of the MIP as much as possible. It consists in the analysis of the given problem instance in order to find a structure that helps to break down the instance, to reduce its size, or to adjust and tighten the MIP formulation. This can be done by, for example, turning some inequalities into equations, or fixing certain variables.

In the PA, by only using techniques specially developed for the $P//C_{\max}$ problem, we implemented identification of infeasibilities and redundancies, improving bounds and coefficients, and fixing variables. The preprocessing task presents three stages that are described in the following.

3.3.1. Fixing variables

As the machines are identical, we can fix in advance a job to any machine. By doing this, we save m binary variables, and do not lose any feasible solution. Furthermore, we have eliminated one constraint of type (1), and the right hand side of the constraints of type (2) is subsequently improved by this tightening adjustment.

3.3.2. Assigning variables

When we take into account the value of the lower bound of $C_{\max}$, $\mathrm{LB}(C_{\max})$, we can identify unfeasible and redundant solutions. We have to check whether

$$p_k + p_l \leqslant \mathrm{LB}(C_{\max})$$

for a determined value of $\mathrm{LB}(C_{\max})$, to looking into the possibility of two jobs with processing times p_k and p_l being assigned to the same machine. Beginning with the first two jobs, $p_1 + p_2$ is compared with $\mathrm{LB}(C_{\max})$, if the sum is greater than $\mathrm{LB}(C_{\max})$, the binary variable x_{21} is fixed at 0, and following the same idea as that described in the above paragraph, job J_2 is assigned to machine M_2, and the x_{2j} variables, for $j = 1, \dots, m$ are fixed. This procedure is applied to $J_2, J_3, \dots, J_m$. The corresponding constraints of type (1) are eliminated, and the constraints of type (2) are also improved by tightening their bounds.

3.3.3. Adding constraints

Having completed the first two stages, the actual $\mathrm{LB}(C_{\max})$ for each machine is not necessary the same. If job J_i was assigned to machine M_j as a consequence of having applied previous stages, then the lower bound $\mathrm{LB}(C_{\max})$ for this machine has decreased by p_i. Thus, by analysis of each machine's new capacity value, it could be the case that a machine cannot process a certain job, J_k without exceeding LB($C_{\max}$). If this is indeed the case, we add the restriction

$x_{kj} = 0.$

The same analysis applies to the remaining jobs and machines.

3.4. Core of the algorithm

If the initial lower and upper bounds (obtained by Section 3.1 and 3.2 respectively) coincide, the algorithm stops, since the heuristic used to obtain an upper bound has found the optimal solution. Otherwise, the preprocessing makes a tightness adjustment in the MIP formulation presented in Section 2. In the following we will refer to the formulation after preprocessing as adjusted MIP formulation to distinguish from the original MIP formulation. An iterative process of convergence starts by solving the linear relaxation of the adjusted MIP formulation in which y is fixed at the current lower bound value. From the solution obtained, whenever it is not integer, the cuts

Table 1
Comparison between total time algorithm required using different upper bound values. This table gives average time values over 10 problems in seconds ($p_{ij} \in$ uniform in range [1–100])

m	n	LPT	FGH$_1$	AP$_{10}$(0.80)	MF(17)
3	5	0.0277	0.0279	0.0279	0.0094
3	6	0.0196	0.0160	0.0169	0.0156
3	7	0.0152	0.0120	0.0144	0.0108
3	8	0.0319	0.0297	0.0309	0.0276
3	9	0.0405	0.0349	0.0427	0.0420
3	10	0.0543	0.0361	0.0377	0.0465
3	15	0.0992	0.1277	0.997	0.0879
3	25	0.0235	0.0294	0.0673	0.0383
3	50	0.0828	0.0556	0.0841	0.0612
3	100	0.0902	0.0560	0.0942	0.0505
3	250	0.2425	0.1328	0.0988	0.1276
3	500	0.0336	0.0010	0.0008	0.0224
3	1000	0.0017	0.0176	0.0011	0.0228
5	6	0.0050	0.0050	0.0050	0.0040
5	7	0.0097	0.0046	0.0046	0.0046
5	8	0.0076	0.0049	0.0067	0.0048
5	9	0.0453	0.0311	0.0384	0.0364
5	10	0.0247	0.0246	0.0246	0.0066
5	15	27.2198	20.5630	19.4565	22.6658
5	25	0.2181	0.3874	0.3511	0.3569
5	50	0.3918	0.0892	0.2970	0.0777
5	100	0.3397	0.2097	0.3090	0.1824
5	250	1.4025	0.0065	0.3362	0.0183
5	500	0.0003	0.0063	0.0003	0.0172
5	1000	0.0071	0.0168	0.0010	0.0325
10	25	1178.2777	1201.1674	1163.1956	1200.9186
10	50	662.4469	567.4265	624.2728	704.8083
10	100	24.5514	205.0813	20.8812	205.2585
10	250	17.3885	0.0217	14.1583	0.0552
10	500	0.0445	0.0554	0.0554	0.0734
10	1000	0.0772	0.0989	0.0552	0.1324
15	25	0.0061	0.0003	0.0003	0.0067
15	50	1646.8634	1813.5872	1391.8967	1761.2515
15	100	568.0953	121.6604	105.8862	171.3333
15	250	124.9129	0.0243	82.4432	0.0271
15	500	53.2511	0.0161	68.3381	0.0101
15	1000	88.9632	0.1321	0.0281	0.0511

described in Section 2.1 are used to truncate the polyhedron of the adjusted MIP formulation where y is fixed. If the resultant linear relaxation is not feasible, the lower bound is increased by one unit. Unless the lower and upper bound values coincide, the original MIP formulation is then retaken, with the updated lower bound value, and the process, including the preprocessing phase, restarts. The algorithm stops when a linear relaxation gives an integral solution or lower and upper bound values coincide. If no new cut is found, we apply the B&B algorithm to the original MIP formulation with the latest lower bound value.

3.4.1. Example

Consider set J of five jobs, with processing times $p_1 = 9$, $p_2 = 7$, $p_3 = 7$, $p_4 = 5$ and $p_5 = 2$. They have to be processed on set M of two identical machines.

Applying the definitions above, the lower bound is 15 and the upper bound obtained by LPT is 16. The preprocessing phase fixes $x^0_{11} = 1$, $x^0_{12} = 0$, $x^0_{21} = 0$, $x^0_{22} = 1$, and $x^0_{31} = 0$.

Table 2
Comparison between time required by cutting plane phase using different upper bound values. This table gives average time values over 10 problems in seconds ($p_{ij} \in$ uniform in range [1–100])

m	n	LPT	FGH$_1$	AP$_{10}$(0.80)	MF(17)
3	5	0.0277	0.0277	0.0277	0.0094
3	6	0.0192	0.0153	0.0163	0.0150
3	7	0.0146	0.0114	0.0136	0.0102
3	8	0.0310	0.0290	0.0304	0.0269
3	9	0.0402	0.0345	0.0362	0.0418
3	10	0.0540	0.0358	0.0362	0.0460
3	15	0.0988	0.1271	0.997	0.0874
3	25	0.0227	0.0289	0.0282	0.0379
3	50	0.0821	0.0548	0.0562	0.0606
3	100	0.0895	0.0554	0.0611	0.0497
3	250	0.2421	0.1322	0.0008	0.1273
3	500	0.0333	0.0005	0.0004	0.0062
3	1000	0.0008	0.0008	0.0008	0.0003
5	6	0.0045	0.0045	0.0045	0.0034
5	7	0.0032	0.0032	0.0032	0.0032
5	8	0.0073	0.0044	0.0062	0.0044
5	9	0.0451	0.0308	0.0380	0.0311
5	10	0.0239	0.0239	0.0239	0.0058
5	15	27.2195	20.5626	19.4559	22.6537
5	25	0.2178	0.3870	0.3507	0.3569
5	50	0.3913	0.0885	0.2773	0.0772
5	100	0.3342	0.2094	0.2529	0.1821
5	250	1.3963	0.0063	0.3142	0.0056
5	500	0.0002	0.0002	0.0002	0.0060
5	1000	0.0005	0.0005	0.0005	0.0055
10	25	1178.2544	1201.1671	1163.1946	1200.9180
10	50	662.4461	567.4261	450.4103	704.8078
10	100	24.5451	205.0811	14.5190	205.2582
10	250	17.3821	0.0104	14.1501	0.0211
10	500	0.0442	0.0442	0.0542	0.0112
10	1000	0.04989	0.0499	0.0498	0.0103
15	25	0.0001	0.0001	0.0001	0.0063
15	50	1646.8629	1813.5859	1390.4502	1761.2514
15	100	568.0898	121.6596	92.7394	171.3311
15	250	124.8892	0.0176	77.9013	0.0055
15	500	53.2450	0.0001	68.3230	0.0001
15	1000	88.9630	0.0220	0.0280	0.0001

Solving the first LP relaxation, which includes the constraint $y = 15$, the following solution is obtained:

$y^0 = 15$

and

$i \setminus j$	M_1	M_2
1	1	0
2	0	1
3	0	1
4	0.8	0.2
5	1	0
Δ	1	4

The sets S_j and S'_j are the following:

$S_1 = \{1,4,5\}, \quad S_2 = \{2,3,4\},$

$S'_1 = \{1,4,5\} \quad \text{and} \quad S'_2 = \{2,3,4\}.$

S_1 yields the new inequality

$$9x_{11} + 5x_{41} + 2x_{51} \leqslant y - 8(1 - x_{11}) - 4(1 - x_{41}) - (1 - x_{51}) \times x_{11} + x_{41} + x_{51} - y \leqslant -13$$

and S_2 yields the new inequality

$$7x_{22} + 7x_{32} + 5x_{42} \leqslant y - 3(1 - x_{22}) - 3(1 - x_{32}) - (1 - x_{42}) \times 4x_{22} + 4x_{32} + 4x_{42} - y \leqslant -7.$$

When these new constraints are added, the LP relaxation is unfeasible for $y = 15$. Increasing the

Table 3
Comparison between the PA and B&B, $p_{ij} \in$ uniform in range [1–100]. This table gives average performance values over 10 problems

m	n	Iterations		Time	
		PA	B&B	PA	B&B
3	5	0.00	0.00	0.06	0.37
3	6	0.00	0.00	0.01	0.36
3	7	12.20	3.60	0.09	0.95
3	8	17.40	21.60	0.10	0.03
3	9	48.40	50.40	0.05	0.06
3	10	15.80	34.40	0.06	0.06
3	11	131.40	460.80	0.14	0.28
3	12	28.40	29.40	0.08	0.05
3	13	98.00	1721.60	0.14	0.77
3	14	46.60	43.00	0.08	0.07
3	15	205.60	5318.40	0.22	1.51
4	5	0.00	0.00	0.01	0.03
4	6	0.00	0.00	0.03	0.03
4	7	0.00	0.00	0.02	0.02
4	8	2.60	0.00	0.06	0.02
4	9	1.20	196.40	0.04	0.18
4	10	185.40	231.60	0.03	0.18
4	11	277.40	275.20	0.09	0.18
4	12	784.60	2803.60	0.57	1.80
4	13	4052.40	6594.80	2.98	3.44
4	14	945.40	4492.60	0.62	3.85
4	15	520.80	109108.40	0.57	39.79
5	6	0.00	0.00	0.01	0.03
5	7	0.00	0.00	0.01	0.03
5	8	0.00	26.60	0.01	0.05
5	9	0.00	0.00	0.02	0.03
5	10	2.40	48.60	0.05	0.10
5	11	237.60	231.80	0.37	0.23
5	12	9.20	1581.80	0.09	1.01
5	13	1637.40	25771.40	1.68	17.00
5	14	11658.00	11653.80	5.39	9.73
5	15	4496.00	110108.40	2.85	41.79

lower bound to $y = 16$, the lower and upper bound reach equality. Thus, the solution to the LPT procedure:

$y^0 = 16$

and

$i \setminus j$	M_1	M_2
J_1	1	0
J_2	0	1
J_3	0	1
J_4	1	0
J_5	1	0
Δ	0	0

is optimal, and the algorithm stops.

4. Computational results

We carried out computational experiments on a Pentium 500, using the simplex and B&B methods, both included in the CPLEX Callable Library 6.5. To avoid the CPLEX routine building a huge tree, because of memory space constraints, we fixed the parameter determining the maximum tree memory at 50 Mb and the parameter determining the maximum time at 2000 seconds (given the high computational complexity of the problem in hand, the CPLEX routine could stop without guaranteeing optimality).

We have investigated the influence of different constructive algorithms for upper bound. In our

Table 4
Comparison between the PA and B&B, $p_{ij} \in$ uniform in range [10–100]. This table gives average performance values over 10 problems

m	n	Iterations		Time	
		PA	B&B	PA	B&B
3	5	0.00	0.00	0.02	0.02
3	6	0.20	0.00	0.02	0.04
3	7	44.40	40.20	0.08	0.03
3	8	39.80	36.80	0.09	0.06
3	9	69.10	123.40	0.12	0.10
3	10	96.60	92.60	0.12	0.10
3	11	314.70	422.40	0.22	0.25
3	12	70.00	66.60	0.12	0.07
3	13	113.00	109.60	0.19	0.13
3	14	85.40	300.40	0.15	0.21
3	15	148.40	5748.10	0.19	1.86
4	5	0.00	0.00	0.01	0.03
4	6	0.00	0.00	0.01	0.02
4	7	0.00	0.00	0.02	0.02
4	8	2.90	27.60	0.08	0.05
4	9	72.90	198.00	0.12	0.15
4	10	445.60	803.00	0.30	0.51
4	11	1251.60	1248.20	0.90	0.86
4	12	1771.00	1767.80	1.17	1.22
4	13	8492.90	14352.60	5.00	7.59
4	14	322.00	318.40	0.14	0.32
4	15	444.90	81034.20	0.49	32.35
5	6	0.00	0.00	0.02	0.02
5	7	0.00	0.00	0.01	0.03
5	8	0.00	12.40	0.02	0.05
5	9	49.50	0.00	0.08	0.05
5	10	2.30	1.80	0.06	0.04
5	11	314.60	312.60	0.30	0.31
5	12	2375.70	13637.40	1.81	8.05
5	13	6505.20	30973.20	5.06	18.95
5	14	57743.60	57740.80	30.98	34.56
5	15	322.00	318.40	0.29	0.32

experiment we have tried with LPT [2], MultiFit with 7 and 17 iterations [1], improvement heuristics with fixed gap with one iteration (FGH_1) [10] and the constructive algorithms with partial sets of machines presented in [3].

The test problems obtained, randomly generating the p_i values according to uniform distributions in the range [1–100] for instances of varying size, have been solved by the PA using the different constructive algorithms to compute the upper bound. Results for MultiFit with 7 iterations was notably dominated by MultiFit with 17 iterations (MF_{17}). Among the constructive algorithms presented in [3] similar results were attained, so we report the results corresponding to the partition into 10 subsets and where $\delta = 80\%$ ($AP_{10}(0.80)$). For different values of n and m, the entries in Table 1 give the average CPU time in seconds required by the PA when using each of the algorithms, computed over 10 problem instances. To evaluate the incidence of refined upper bound in the cutting plane phase, we present Table 2 which gives the average over 10 problem instances of CPU time in seconds required by the PA when using each of the algorithms without taking into account the time required to compute the upper bound. The results shows that the $AP_{10}(0.80)$ gives slightly better upper bound values than the others with difficult

Table 5
Comparison between the PA and B&B, $p_{ij} \in$ uniform in range [50;100]. This table gives average performance values over 10 problems

m	n	Iterations		Time	
		PA	B&B	PA	B&B
3	5	0.00	0.00	0.06	0.02
3	6	2.60	0.00	0.06	0.01
3	7	56.40	52.80	0.10	0.03
3	8	76.00	72.40	0.12	0.09
3	9	34.00	40.20	0.11	0.07
3	10	317.20	313.60	0.22	0.16
3	11	952.50	1124.00	0.54	0.53
3	12	251.70	591.80	0.38	0.37
3	13	2788.30	4000.80	1.38	1.56
3	14	3095.10	5675.60	1.48	2.54
3	15	188.50	95750.10	0.27	7.64
4	5	0.00	0.00	0.01	0.02
4	6	0.00	0.00	0.07	0.03
4	7	1.50	0.00	0.05	0.04
4	8	4.60	0.00	0.12	0.04
4	9	432.40	429.20	0.25	0.20
4	10	1175.50	1357.80	0.74	0.78
4	11	1428.00	1424.20	0.99	0.96
4	12	957.00	971.60	0.95	0.88
4	13	32442.40	32437.60	15.74	15.55
4	14	79954.30	109046.60	39.84	52.24
4	15	80362.70	154953.10	47.06	87.77
5	6	0.00	0.00	0.01	0.03
5	7	0.00	0.00	0.03	0.03
5	8	2.00	0.60	0.04	0.04
5	9	4.00	0.00	0.02	0.02
5	10	5.80	1.00	0.06	0.06
5	11	10174.80	10171.40	3.61	3.61
5	12	31845.80	31841.60	17.96	17.96
5	13	115404.60	115400.20	74.23	74.55
5	14	63308.80	63304.60	50.04	53.34
5	15	109050.80	119063.60	52.73	95.20

Table 6
Performance of PA and B&B for larger size instances, $p_{ij} \in$ uniform in range [1–100]. This table gives average performance values over 10 problems

m	n	PA			B&B		
		Iterations	Time	Unsolved inst.	Iterations	Time	Unsolved inst.
3	20	25.40	0.12	0	3826073.50	1729.20	8
3	50	5.80	0.10	0	1451551.50	1203.65	6
3	100	17.00	0.22	0	777414.30	1210.26	6
3	200	4.60	0.12	0	133795.50	638.43	3
3	500	0.00	0.01	0	385.90	15.43	0
5	20	344.80	0.57	0	560053.90	431.09	2
5	50	29.20	0.35	0	1849177.30	1617.32	8
5	100	44.40	0.67	0	879344.70	1802.66	9
5	200	34.80	1.23	0	326452.70	1643.67	8
5	500	0.00	0.01	0	77960.20	1050.35	5
15	20	0.00	0.01	0	498004.30	918.36	4
15	50	103490.40	864.47	2	–	–	10
15	100	40153.00	329.66	0	–	–	10
15	200	11824.60	348.79	0	37489.60	1964.59	9
15	500	320.60	83.54	0	6411.60	1847.71	9
100	200	27.00	2042.08	0	–	–	10
100	500	292.20	2768.05	2	–	–	10
100	1000	182.00	5098.40	0	–	–	10

instances like 5–15, 10–25 and 15–50, while MF_{17} is known to be more efficient solving easy instances.

To analyse the performance of PA, we have compared it with the B&B algorithm which forms part of the CPLEX Callable Library 6.5. We have considered three classes of test problems obtained by randomly generating the p_i values according to uniform distributions in the ranges [1–100], [10–100] and [50–100]. The upper bound for PA has been computed by LPT in this experiment. For each class, and for different values of n and m, the entries in Tables 3–5 give the average number of iterations and of CPU time in seconds, required by each of the two algorithms, computed over 10 problem instances. For PA the number of iterations is the sum of the number of solved LP relaxations and of the nodes of the B&B procedure (when PA makes use of B&B to complete the search). For the B&B an iteration is every solved node.

The computational performance of PA was in general satisfactory for all tested cases. Given the high computational complexity of the problem in hand PA and B&B could stop without guaranteeing that the obtained solution is optimal. In the tested cases, that we present in Tables 3–5, both were always able to converge to the optimal integral solution. Table 6 presents the performance of the PA and B&B for larger size problems. It is important to notice that the B&B was not capable of optimizing a high percentage of generated instances (the number of unsolved instances is indicated when it is greater than zero). Furthermore, we can see that there is an important saving in the average computational effort of the PA.

As expected, the PA reaches the optimal solution in many more cases for the same capacity of memory and in less time when we apply it to $P//C_{\max}$ than when we apply it to $R//C_{\max}$. Therefore application to $P//C_{\max}$ is even more efficient than the results presented in [9].

5. Concluding remarks

In this paper we propose a new algorithm based on cutting plane techniques for a hard scheduling problem. We have followed the trend of recent CO literature, which applies Polyhedral Theory to optimize NP-hard combinatorial problems.

The obtained results show that this kind of approach is quite a powerful tool for effectively producing quality feasible solutions. They also give support to the hypothesis stating that specially-developed algorithms for specific combinatorial problems work better than general methods like B&B.

After having applied this technique to $P//C_{\max}$ and $R//C_{\max}$, and having refined the computation of initial lower and upper bound, we have been able to prove the efficiency of this algorithm and thus we are now considering the development of similar approaches to solve even harder scheduling problems such as job-shop problems.

References

[1] E.G. Coffman Jr., M.R. Garey, D.S. Johnson, An application of Bin-Packing to multiprocessor scheduling, SIAM Journal on Computing 7 (1978) 1–17.

[2] R.L. Graham, Bounds on the performance of scheduling algorithms, SIAM Journal on Applied Mathematics 17 (1969) 263–269.

[3] J.L. Jimeno, E. Mokotoff, J. Pérez, A constructive algorithm to minimise the makespan on identical parallel machine, in: Eighth International Workshop on Project Management and Scheduling, Valencia, 2002.

[4] M. Jünger, G. Reinelt, S. Thienel, Practical problem solving with cutting plane algorithms in combinatorial optimization, in: W. Cook, L. Lovász, P. Seymour (Eds.), Combinatorial Optimization, Dimacs, Vol. 20, 1995, 111–152.

[5] J.K. Lenstra, A.H.G. Rinnooy Kan, Computational complexity of discrete optimization, in: J.K. Lenstra, A.H.G. Rinnooy Kan, P. Van Emde Boas (Eds.), Interfaces between Computer Science and Operations Research, Proceedings of a Symposium held at the Matematisch Centrum, Amsterdam, 1979, 64–85.

[6] R. McNaughton, Scheduling with deadlines and loss function, Management Science 6 (1959) 1–12.

[7] E. Mokotoff, Parallel machines scheduling problems: A survey, Asia-Pacific Journal of Operational Research 18 (2001) 193–242.

[8] E. Mokotoff, Scheduling to minimize the makespan on identical parallel machines: An LP-based algorithm, Investigación Operativa 8 (1999) 97–108.

[9] E. Mokotoff, P. Chrétienne, A cutting plane algorithm for the unrelated parallel machine scheduling problem, European Journal of Operational Research 141 (2002) 517–527.

[10] E. Mokotoff, J.L. Jimeno, A. Gutiérrez, List scheduling algorithms to minimize the makespan on identical parallel machines, Top 9 (2001) 243–269.

[11] W.R. Pulleyblank, Polyhedral Combinatorics, in: J.L. Nemhauser, A.H.G. Rinnooy Kan, M.J. Todd (Eds.), Handbooks in Operations Research and Management Science 1: Optimization, 1989, pp. 371–446.

[12] T. Van Roy, L. Wolsey, Valid inequalities for mixed 0-1 programs, Discrete Applied Mathematics 4 (1986) 199–213.