Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR TP 96251

Parallel machine scheduling by column generation

J.M. van den Akker, J.A. Hoogeveen, S.L. van de Velde

DOCUMENT CONTROL SHEET

	ORIGINATOR'S REF.		SECURITY CLASS.
	NLR TP 96251 U		**Unclassified**

ORIGINATOR
National Aerospace Laboratory NLR, Amsterdam, The Netherlands

TITLE
Parallel machine scheduling by column generation

PRESENTED AT
Fifth international workshop on project management en scheduling held at 11-13 April 1996, in Poznan, Poland

AUTHORS	DATE	pp	ref
J.M. van den Akker, J.A. Hoogeveen, S.L. van de Velde	**960506**	**28**	**20**

DESCRIPTORS

Air traffic control
Air traffic
Algorithms
Arrivals
Cost reduction
Linear programming
Parallel processing (computers)
Run time (computers)
Scheduling
Tasks
Weighting functions

ABSTRACT
Parallel machine scheduling problems concern the scheduling of n jobs on m machines to minimize some function of the job completion times. If preemption is not allowed, then most problems are not only NP-hard, but also very hard from a practical point of view. In this paper, we show that strong and fast linear programming lower bounds can be computed for an important class of machine scheduling problems with additive objective functions. Characteristic of these problems is that on each machine the order of the jobs in the relevant part of the schedule is obtained through some priority rule. To that end, we formulate these parallel machine scheduling problems as a set covering problems with an exponential number of binary variables, n covering constraints, and a single side constraint. We show that the linear programming relaxation can be solved efficiently by column generation, since the pricing problem is solvable in pseudo-polynomial time. We display this approach on the problem of minimizing total weighted completion time on m identical machines. Our computational results show that the lower bound is singularly strong and that the outcome of the linear program is often integral. Moreover, they show that our branch-and-bound algorithm that uses the linear programming lower bound outperforms the previously best algorithm. We elaborate on the application of the presented approach to parallel machine scheduling problems other than that of minimizing total weighted completion time on identical machines. Moreover, we discuss the occurrence of parallel machine scheduling problems in the field Air Traffic Management (ATM) and investigate the applicability of the approach to problems in this field.

Summary

Parallel machine scheduling problems concern the scheduling of n jobs on m machines to minimize some function of the job completion times. If preemption is not allowed, then most problems are not only $\mathcal{NP}$-hard, but also very hard from a practical point of view. In this paper, we show that strong and fast linear programming lower bounds can be computed for an important class of machine scheduling problems with additive objective functions. Characteristic of these problems is that on each machine the order of the jobs in the relevant part of the schedule is obtained through some priority rule. To that end, we formulate these parallel machine scheduling problems as a set covering problems with an exponential number of binary variables, n covering constraints, and a single side constraint. We show that the linear programming relaxation can be solved efficiently by column generation, since the pricing problem is solvable in pseudo-polynomial time. We display this approach on the problem of minimizing total weighted completion time on m identical machines. Our computational results show that the lower bound is singularly strong and that the outcome of the linear program is often integral. Moreover, they show that our branch-and-bound algorithm that uses the linear programming lower bound outperforms the previously best algorithm. We elaborate on the application of the presented approach to parallel machine scheduling problems other than that of minimizing total weighted completion time on identical machines. Moreover, we discuss the occurrence of parallel machine scheduling problems in the field Air Traffic Management (ATM) and investigate the applicability of the approach to problems in this field.

Contents

(28 pages in total)

List of tables

This page is intentionally left blank.

1 Introduction

Parallel machine scheduling problems concern the scheduling of n jobs on m parallel machines to minimize some function of the job completion times. Problems in which preemption of jobs is not allowed decompose into two subproblems: *assigning* jobs to machines and then *sequencing* the jobs on each machine. We consider the class of problems with additive objective functions that have all jobs in the relevant part of the schedule sequenced according to some priority rule – we refer to it as class (A). The difficult part lies then mainly in the assignment of jobs to machines, because for a given assignment we can find the optimal schedule by sequencing the jobs in the relevant part of the schedule according to the priority rule, after which the non-relevant jobs are scheduled after the other jobs.

Class (A) contains important objective functions like total weighted completion time, for which the whole schedule is relevant, and objective functions like the weighted number of tardy jobs and total weighted late work, for which only the on-time part of the schedule is relevant. All these problems are unary $\mathcal{NP}$-hard but solvable in pseudo-polynomial time for a ***fixed*** number of machines m by applying the dynamic programming techniques of Rothkopf [15] and Lawler and Moore [11]. These pseudo-polynomial algorithms, however, are impractical unless $m = 2$ for the processing times are (very) small.

Additive objective functions pose a computational challenge, since it is difficult to compute strong lower bounds. This is nicely witnessed by the research effort that the problem of minimizing the total weighted completion on m identical parallel machines has attracted since the early days of machine scheduling research; see for instance Eastman, Evan, and Issacs [7], Elmaghraby and Park [8], Barnes and Brennan [1], Sarin, Ahn, and Bishop [16]. Using the notation scheme of Graham, Lawler, Lenstra, and Rinnooy Kan [9], we refer to this problem as $P||\sum_{j=1}^{n} w_j C_j$ or as $P_m||\sum_{j=1}^{n} w_j C_j$ when the number of machines m is fixed. The lower bounds developed by Webster [19, 20], which require pseudo-polynomial time, and Belouadah and Potts [3] are a big leap forward. Belouadah and Potts also report on the performance of a branch-and-bound algorithm that uses their bound; it is capable of solving instances with up to 20 jobs and 5 machines.

It is also remarkable that other parallel machine scheduling problems with additive objective functions have not received any attention yet. For instance, no one ventured at the parallel machine problem of minimizing the weighted number of tardy jobs or total late work, although their single-machine counterparts, which are binary $\mathcal{NP}$-hard, attracted considerable interest; see for instance Potts and Van Wassenhove [13, 14] and Hariri, Potts, and Van Wassenhove [10].

In this paper, we present a methodology that can be used to deal with the problem $P||\sum_{j=1}^{n} w_j C_j$ and the other problems in class (A); we describe the approach in detail for the former problem and indicate how it can be modified to deal with the latter problems. The approach is based on formulating the problem $P||\sum_{j=1}^{n} w_j C_j$ as a set covering problem with an exponential number of binary variables, n covering constraints, and a single side constraint. We then solve the linear programming relaxation of this formulation by a column generation approach that uses an $O(n\sum_{j=1}^{n} p_j)$ algorithm to solve the corresponding pricing problem. Obviously, if the optimal solution for the linear programming relaxation happens to be integral, then we have identified an optimal solution for the problem $P||\sum_{j=1}^{n} w_j C_j$. If not, then we apply a branch-and-bound algorithm to determine an optimal solution. Our computational results show the compelling quality of the linear programming bound, which makes branching often unnecessary, and the superiority of our branch-and-bound algorithm to the algorithms presented before.

We also study the application of the presented approach to problems in the area of Air Traffic Management. We show that the scheduling of arrival aircraft on runways of an airport can be viewed as a parallel machine scheduling problem and we discuss the application of the presented column generation algorithm to this problem.

When we were conducting this research, Chan, Kaminsky, Muriel, and Simchi-Levi [5] as well as Chen and Powell [6] independently proposed and analyzed the column generation approach to this formulation of the problem $P||\sum_{j=1}^{n} w_j C_j$. Chan, Kaminsky, Muriel, and Simchi-Levi emphasized on worst-case performance analysis and probabilistic analysis. They have established two main results. The first one is that the linear programming bound is asymptotically optimal for any number of machines; if $m = 2$, then these values always coincide. Their second main result is that the value of the optimal solution is at most equal to $(1+\sqrt{2})/2$ times the value of the linear programming bound; this bound is strengthened to 1.04 in case $w_j = p_j$ for all $j = 1, \ldots, n$. Chen and Powell show how the formulation can be obtained by Dantzig Wolfe decomposition. They also propose a branch-and-bound algorithm in which lower bounds are computed by solving the linear programming relaxation through column generation. They do not branch on the completion times, but on the original x_{ij} variables that indicate that job i is processed immediately before job j on some machine. As a consequence, they cannot use the $O(n\sum p_j)$ algorithm to solve the pricing problem after the branching has started, but have to resort to an $O(n^2\sum p_j)$ time algorithm.

This paper is organized as follows. In Chapter 2, we present the column generation approach for the problem $P||\sum_{j=1}^{n} w_j C_j$. In Chapter 3, we report on our computational experiments for this

problem. In Chapter 4, we discuss the adaptations of the formulation and the pricing algorithm necessary to apply them to other prominent problems in the class (A). In Chapter 5, we discuss an application in the field of Air Traffic Management. Chapter 6 concludes the paper.

2 Column generation for $P||\sum_{j=1}^{n} w_j C_j$

2.1 Problem description

There are m identical machines, $M_1, \ldots, M_m$, available for processing n independent jobs, $J_1, \ldots, J_n$. Job J_j $(j = 1, \ldots, n)$ has a processing requirement of length p_j and a weight w_j. Each machine is available from time zero onwards and can handle no more than one job at a time. Preemption of jobs is not allowed. A *feasible schedule* is a specification of the job completion times $C_1, \ldots, C_n$ such that no machine processes more than one job at a time and $C_j - p_j \geq 0$. The objective is to find a schedule with minimum total weighted completion time $\sum_{j=1}^{n} w_j C_j$. The problem is $\mathcal{NP}$-hard in the strong sense when the number of machines is part of the problem instance.

There are two important observations that we can make with respect to the form of any optimal schedule. First, on each machine the jobs need to be processed contiguously from time zero onwards, and no machine should be idle before all jobs have been started. From this, it follows that the last job on any machine is completed between time $H_{\text{min}} = \sum_{j=1}^{n} p_j/m - (m-1)p_{\text{max}}/m$ and $H_{\text{max}} = \sum_{j=1}^{n} p_j/m + (m-1)p_{\text{max}}/m$, where $p_{\text{max}} = \max_{1 \leq j \leq n} p_j$. Second, jobs that are processed by the same machine are scheduled in order of non-increasing w_j/p_j ratios in any optimal schedule; this follows directly from Smith's rule for the single-machine version (Smith [17]). Hence, the problem belongs to the class (A). In the remainder, we assume that the jobs have been reindexed in order of non-increasing ratios, and to avoid trivialities, we also assume that $n > m$. There exists a dynamic programming algorithm based on the observations made above that runs in $O(n(\sum_{j=1}^{n} p_j)^{m-1})$ time and space. Especially the space requirement becomes unmanageable when m increases.

2.2 Mathematical formulation

The $P||\sum_{j=1}^{n} w_j C_j$ problem can be mathematically formulated as a set covering problem with an exponential number of binary variables, n covering constraints, and a single side constraint. We define a *machine schedule* as a string of jobs that can be assigned together to any single machine. Let a_{js} be a constant that is equal to 1 if job J_j is included in machine schedule s and 0 otherwise. Accordingly, the column $(a_{1s}, \ldots, a_{ns})^T$ represents the jobs in machine schedule s. Let $C_j(s)$ be the completion time of job J_j in s; $C_j(s)$ is defined only if $a_{js} = 1$. Note that since the jobs in s appear in order of their indices we have that $C_j(s) = \sum_{k=1}^{j} a_{ks} p_k$. Hence, the cost c_s of machine

schedule s is readily computed as

$$c_s = \sum_{j=1}^{n} w_j C_j(s) = \sum_{j=1}^{n} w_j a_{js} \left[\sum_{k=1}^{j} a_{ks} p_k \right].$$

Let S be the set containing all feasible machine schedules. We introduce variables x_s ($s = 1, \ldots, |S|$) that assume value 1 if machine schedule s is selected and 0 otherwise. The problem is then to select m machine schedules, one for each machine, such that together they contain each job exactly once and minimize total cost. Mathematically, the problem is then to determine values x_s that minimize

$$\sum_{s \in S} c_s x_s$$

subject to

$$\sum_{s \in S} x_s = m, \tag{1}$$

$$\sum_{s \in S} a_{js} x_s = 1, \text{ for each } j = 1, \ldots, n, \tag{2}$$

$$x_s \in \{0, 1\}, \text{ for each } s \in S. \tag{3}$$

Condition (1) together with the integrality conditions (3) ensure that exactly m machine schedules are selected. The conditions (2) ensure that each job is executed exactly once. Note that the equality sign in the conditions (1) and (2) can be changed into 'smaller than' and 'larger than', respectively, without loosing the validity of the formulation.

The number of columns involved in this formulation is $|S| = \sum_{k=1}^{n-m+1} \binom{n}{k}$. Neither the set covering problem, nor its linear programming relaxation, which is obtained by replacing conditions (3) by the conditions $x_s \geq 0$ for all $s \in S$, can therefore be solved by a method that first generates all feasible columns explicitly. Instead, we resort to a method that considers the feasible columns implicitly: column generation. Starting with a restricted linear programming problem in which only a subset of the variables is available, the column generation method solves the linear programming relaxation of the set covering formulation by adding new columns that may decrease the solution value, if the optimal solution has not been determined yet; these new columns are not obtained through enumeration, but by solving an optimization problem, which is called the

pricing problem. We discuss this problem in the next section.

2.3 The pricing algorithm

From the theory of linear programming, we know that a solution to a minimization problem is optimal if the *reduced cost* of each variable is non-negative. In our problem, the reduced cost c'_s of any machine schedule s is given by

$$c'_s = c_s - \lambda_0 - \sum_{j=1}^{n} \lambda_j a_{js},$$

where λ_0 is the given value of the dual variable corresponding to condition (1) and $\lambda_1, \ldots, \lambda_n$ are the given values of the dual variables corresponding to conditions (2). To test whether the current solution is optimal, we determine if there exists a machine schedule $s \in S$ with negative reduced cost. To that end, we solve the *pricing problem* of finding the machine schedule in S with minimum reduced cost. Since λ_0 is a constant that is included in the reduced cost of each machine schedule, we essentially have to minimize

$$c_s - \sum_{j=1}^{n} \lambda_j a_{js} = \sum_{j=1}^{n} [w_j (\sum_{k=1}^{j} a_{ks} p_k) - \lambda_j] a_{js}.$$

Our algorithm, which we call the pricing algorithm, tests whether a feasible solution to the linear programming relaxation is optimal; if the outcome is negative, then it outputs a set of machine schedules s with $c'_s < 0$ among which the machine schedule with minimum reduced cost.

Our pricing algorithm is based on dynamic programming and uses a forward recursion that exploits the property that on each machine the jobs are sequenced in order of increasing indices. Let $F_j(t)$ denote the minimum reduced cost for all machine schedules that consist of jobs from the set $\{J_1, \ldots, J_j\}$ and complete their last job at time t. Furthermore, let $P(j) = \sum_{k=1}^{j} p_k$. For the schedule that realizes $F_j(t)$, there are two decisions possible: either leave J_j out of the machine schedule, or include it. As to the first possibility, we then select the best machine schedule with respect to the first $j-1$ jobs that finishes at time t; the value of this solution is $F_{j-1}(t)$. As to the second possibility, we add J_j to the best machine schedule for the first $j-1$ jobs that finishes at time $t - p_j$; the value of this solution is $F_{j-1}(t - p_j) + w_j t - \lambda_j$. The initialization is then

$$F_j(t) = \begin{cases} -\lambda_0, & \text{if } j = 0 \text{ and } t = 0, \\ \infty, & \text{otherwise.} \end{cases}$$

The recursion is then for $j = 1, \ldots, n$, $t = 0, \ldots, \min\{P(j), H_{\max}\}$

$$F_j(t) = \min\{F_{j-1}(t), F_{j-1}(t - p_j) + w_j t - \lambda_j\}. \tag{4}$$

The optimal solution value is then found as

$$F^* = \min_{H_{\min} \leq t \leq H_{\max}} F_n(t).$$

Accordingly, if $F^* \geq 0$, then the current linear programming solution is optimal. If $F^* < 0$, then it is not, and we need to introduce new columns to the problem. Candidates are associated with those t for which $F_n(t) < 0$; they can be found by backtracing. Note that the pricing algorithm requires $O(n \sum_{j=1}^{n} p_j)$ time and space. This means that our column generation approach is not sensible if $m = 2$: the $P_2||\sum_{j=1}^{n} w_j C_j$ problem is better solved directly through dynamic programming.

2.4 The branch-and-bound algorithm

Let x^* denote the optimal solution to the linear programming relaxation of the set covering formulation and let S^* denote the set containing all columns s for which $x_s^* > 0$. If x^* is integral, then x^* constitutes an optimal solution for $P||\sum_{j=1}^{n} w_j C_j$. If not, then we have fractional machine schedules. We first discuss a special case of a fractional solution, namely the case in which for each job the completion time is equal in each machine schedule in S^* in which it occurs. This special case is less esoteric than it may seem on first sight; it occurred quite often in our computational experiments. The next result is then a powerful tool to fathom the corresponding node in the branch-and-bound tree.

Theorem 1 *If $C_j(s) = C_j$ for each job J_j $(j = 1, \ldots, n)$ and for each s with $x_s^* > 0$, then the schedule obtained by processing J_j in the time interval $[C_j - p_j, C_j]$ $(j = 1, \ldots, n)$ is feasible and has minimum cost.*

Proof. The schedule in which job J_j $(j = 1, \ldots, n)$ is processed from time $C_j - p_j$ to time C_j is feasible if and only if at most m jobs are processed at the same time and no job starts before time zero. The second condition is obviously satisfied, since the C_j values originate from feasible machine schedules. The first constraint is satisfied if we show that at most m jobs are started at time zero and that the number of jobs started at any point in time $t \in [1, T]$ is no more than the number of jobs completed at that point in time, where T denotes the latest point in time that a job is started. Let $A(t) \subseteq S^*$ be the set of all machine schedules in which at least one job starts at time t; similarly, let $B(t) \subseteq S^*$ be the set of all machine schedules in which at least one job completes at time t. As $C_j(s) = C_j$, for any machine schedule containing J_j, the number of jobs started at time t is equal to $\sum_{s \in A(t)} x_s^*$; similarly, the number of jobs completed at time t is equal to $\sum_{s \in B(t)} x_s^*$. Because of condition (1), we know that at most m jobs are started at time zero. Since each machine schedule s is constructed such that there is no idle time between the jobs, there can only be a job in s that starts at time t if some other job in s is completed at time t. Hence, $A(t) \subset B(t)$, which means that the indicated schedule is feasible. It is readily checked that the condition $C_j(s) = C_j$ implies that the cost of this schedule is equal to the cost of the fractional

solution, and hence minimal. □

If the optimal solution to the linear program neither is integral, nor satisfies the conditions of Theorem 1, then a branch-and-bound algorithm is required to find an optimal solution. From other applications, we know that the branching strategy of fixing a variable at either zero or one does not work in combination with column generation, as the pricing algorithm may come up with this column again, even though we fixed the variable at zero. Our branching strategy is based upon splitting the set of possible completion times.

If we have a fractional optimal solution that does not satisfy the conditions of Theorem 1, then there is at least one job J_j for which

$$\sum_{s \in S^*} C_j(s) x_s^* > \min\{C_j(s) \mid x_s^* > 0\};$$

we call such a job J_j a *fractional job*. Our partitioning strategy reflects this property. We design a binary branch-and-bound tree for which in each node we first identify the fractional job with smallest index, and, if any, then create two descendant nodes: one for the condition that $C_j \leq \min\{C_j(s) \mid x_s^* > 0\}$ and one for the condition that $C_j \geq \min\{C_j(s) \mid x_s^* > 0\} + 1$. The first condition essentially specifies a deadline $\bar{d}_j$ at which J_j must be completed; the second condition specifies a release date $r_j = \min\{C_j(s) \mid x_s^* > 0\} + 1 - p_j$ before which J_j cannot be started.

The nice thing of this partitioning strategy is that either type of condition can easily be incorporated in the pricing algorithm without increasing its time or space requirement. In fact, we simply have to replace equation (4) by

$$F_j(t) = \begin{cases} \min\{F_{j-1}(t), F_{j-1}(t-p_j) + w_j t - \lambda_j\}, & \text{if } r_j + p_j \leq t \leq \bar{d}_j, \\ F_{j-1}(t), & \text{otherwise.} \end{cases}$$

An additional convenient feature is that this partitioning strategy facilitates fast reoptimization, because we can temporarily discard the columns in S^* that violate the new cut. Observe that it is possible that the remaining columns do not constitute a feasible solution to the linear programming relaxation anymore, although this did not occur in our computational experiments. This problem can be overcome by generating some additional columns that satisfy all constraints used to describe the current node. We therefore have to find a solution to a parallel machine scheduling problem in which all jobs scheduled on the same machine are in order of nonincreasing w_j/p_j ratio and all release dates and deadlines specified by the node are met. To find such a solution we use a greedy heuristic. Before starting, we first compare r_j to $\sum_{i=1}^{j-1} p_i/m$ for any job j with a release date, and we compare $\bar{d}_j$ to $\sum_{i=1}^{j} p_i/m$ for any job j that has a deadline. This comparison provides us with an indication of whether we have to take special precautions in the form of machines with a very

high of very low workload as far as the distribution of the first jobs is concerned. We then add the jobs to the current partial schedule in order of nonincreasing w_j/p_j ratio, where we try to meet the release dates and deadlines. Jobs without release date or deadline are added to create a highly loaded or underloaded machine, if such precautions are wanted.

If the greedy heuristic does not succeed in finding a feasible solution, then we apply a branch-and-bound algorithm similar to the algorithm proposed by Carlier [4] for $P|r_j|L_{\max}$.

3 Computational experiments

3.1 Implementation

In this section, we report on our computational results for the problem $P||\sum_{j=1}^{n} w_j C_j$. The algorithms were coded in the computer language C; the experiments were conducted on an HP9000/710 machine. To solve the linear programs, we used the package CPLEX.

To start the column generation method, we need some initial columns to compute the initial dual variables. We use a simple randomized list scheduling heuristic, which is run several times. The heuristic generates a schedule by assigning the jobs in order of non-increasing values w_j/p_j randomly to the machines, where an earlier available machine has a higher probability of getting the next job on the list.

We incorporated some speed-ups in the pricing algorithm to reduce the empirical running time. Since these speed-ups are not of interest for the red line of the paper, we have included the details in Appendix A.

3.2 Computational results

We first applied our algorithm to the fifteen instances given in Barnes and Brennan [1]. In these instances, the number of jobs n varies form 5 to 20 and the number of machines m varies from 2 to 5. For thirteen of them, the linear program had an integral optimal solution, and hence no branching was required. Problems #10 and #14 required one and six nodes, respectively, while the solution value of the linear program was equal to the optimal solution value; it was hence only a question of finding an integral solution. The computation time was negligible for each instance.

We then tested our algorithm on three classes of randomly generated instances:

(i) instances with processing times drawn from the uniform distribution $[1, 10]$ and weights from the uniform distribution $[10, 100]$;

(ii) instances with processing times and weights both drawn from the uniform distribution $[1, 100]$;

(iii) instances with processing times and weights both drawn from the uniform distribution $[10, 20]$.

We tested our algorithm on instances with $n = 20, 30, 40, 50$ jobs and $m = 3, 4, 5$ machines. As we will see, the performance of our algorithm increases with the number of machines *for fixed* n, which contrasts with other branch-and-bound algorithms, including the one by Belouadah and Potts [3]. For this reason, we did not consider instances with more than five machines. Accordingly, problems with three machines are the hardest to solve for our algorithm, which is

confirmed by our computational experiments. Recall that problems with $m = 2$ are better directly solved by dynamic programming than by our algorithm.

We divide these instances into 'easy' instances, for which no branching was required, and 'hard' instances, for which the branch-and-bound algorithm needed to be invoked. For each combination of n and m, we report on the number of easy instances out of 20 and the average time to solve the linear programming problem for them. For the 'hard' instances, we report on the *maximum* number of nodes in the branch-and-bound tree, the *maximum* computation time to solve a hard instance, and the *maximum gap* between linear programming solution value and optimal solution value in absolute terms. The maximum percent excess of the optimal solution value over the linear programming solution was always smaller than 0.05%, and goes unreported. Finally, we report the number of instances for which the gap is zero.

Tables 1-3 summarize our computational results. The headers of the columns are:

n	=	number of jobs;
m	=	number of machines;
NB	=	number of instances out of 20 for which branching was not required;
ACT	=	average computation time in seconds for the 'easy' instances;
MNN	=	maximum number of search tree nodes;
MCT	=	maximum computation time in seconds for the 'hard' instances;
MGAP	=	maximum gap between optimal solution value and lower bound;
ILP=LP	=	number of instances out of 20 for which the optimal solution value and lower bound concur.

Table 1 displays our results for instances belonging to class (i), which is used by Belouadah and Potts [3] as well to test the performance of their branch-and-bound algorithm. They report that their algorithm fails to solve instances within one minute of computation time on a CDC 7600 computer, which is about twice as slow as our machine, if $n = 30$ and $m = 3$ or $m = 4$, and if $n = 20$ and $m = 5$.

First of all, Table 1 shows the quality of the linear programming lower bound: branching is often not required, particularly for the instances with $n \leq 30$, and the lower bound is tight for all 240 instances but 7. Accordingly, if the linear programming solution is not integral, then it is often a

		'easy' instances		'hard' instances			
n	m	NB	ACT	MNN	MCT	MGAP	ILP=LP
20	3	20	1.13	-	-	-	20
20	4	17	0.72	4	1.59	3	18
20	5	18	0.45	2	0.53	0	20
30	3	14	8.67	2	20.83	0	20
30	4	11	4.99	2	7.33	0	20
30	5	13	2.90	5	5.13	0	20
40	3	6	55.31	2	73.60	0	20
40	4	9	23.84	12	70.20	3	16
40	5	4	14.60	3	21.17	0	20
50	3	3	243.91	5	425.76	36	19
50	4	3	101.25	7	181.43	0	20
50	5	2	49.73	6	81.89	0	20

Table 1 Results for randomly generated instances.

question of finding an integral solution of the same value; this is an important reason that the rule stipulated in Theorem 1 is so useful. If branching is required, then we need few nodes only to find and verify an optimal solution. Table 1 confirms the claim that we made earlier: for *fixed* n, the instances grow easier with increasing m. This is plausible: the more machines involved, the fewer jobs we may expect to appear in the optimal machine schedules, and accordingly the smaller the relevant solution space will be. Also note that for $n = 40$ and $n = 50$ the linear programming solution is fractional more often than for $n \leq 30$, although the corresponding value is about equally often tight. A likely reason for this phenomenon is that the number of optimal fractional solutions grows with n and the chances of 'hitting' an integral one decreases accordingly.

Since the pricing algorithm requires pseudo-polynomial time, we may expect that the performance of our algorithm deteriorates with the size of the processing times of the jobs. Table 2 displays our results for instances belonging to class (ii), where the processing times are drawn from the uniform distribution $[1, 100]$.

Table 2 indicates that these instances are indeed harder to solve. Not only do we need more time to solve the linear programs, which is entirely attributable to the pricing algorithm, but also slightly more search nodes. The extra time effort is modest, however. Note that the maximum gap is slightly bigger in comparison with Table 1. As a whole, the results remain satisfactory.

		'easy' instances		'hard' instances			
n	m	NB	ACT	MNN	MCT	MGAP	ILP=LP
20	3	19	2.82	1	2.70	0	20
20	4	20	1.87	-	-	-	20
20	5	20	1.34	-	-	-	20
30	3	17	15.87	6	27.17	4	17
30	4	18	9.79	10	18.85	9	19
30	5	18	6.80	14	15.07	10	18
40	3	18	71.74	4	104.30	6	19
40	4	12	36.42	10	72.09	11	14
40	5	15	25.59	18	63.27	8	17
50	3	16	314.33	10	462.47	5	17
50	4	13	131.53	12	259.64	11	15
50	5	13	78.89	18	211.92	18	15

Table 2 Results for instances with large processing times.

Finally, we may also expect that instances with fairly homogeneous ratios w_j/p_j are hard to solve as well. After all, if all ratios w_j/p_j are close to each other, then the number of relevant columns will be large. This intuition is confirmed by our computational results for the instances belonging to class (iii).

Indeed, Table 3 shows that the solution to the linear programming problem is seldom integral, although it is often tight. Apparently, many optimal solutions are fractional. At the same time, it is more difficult to find an integral solution: we need more search nodes than before. This is quite plausible: since the ratios w_j/p_j are close to each other, the position of a job in the final schedule is not so predetermined, and accordingly our branching rule will be less effective. Note that for $n = 50$ and $m = 3$, there is only a single instance for which the linear programming bound is tight, which contrasts with the other results.

Finally, we note that our algorithm is able to deal with instances with $n > 50$ as well, as long as the number of machines is not too small. Also, but this is a subjective feeling, the column generation approach gives the idea that solving the parallel machine scheduling problem comes down to cracking the linear programming relaxation: once you have done that, you only need a small number of search nodes to find an optimal solution. This would imply that you can really benefit from faster computers. This is opposite to our experience with most standard branch-and-bound

		'easy' instances		'hard' instances			
n	m	NB	ACT	MNN	MCT	MGAP	ILP=LP
20	3	13	1.11	2	1.29	0	20
20	4	13	0.74	2	1.07	0	20
20	5	11	0.51	4	0.89	0	20
30	3	4	7.53	9	19.17	0	20
30	4	2	3.54	9	10.69	0	20
30	5	2	3.53	9	6.64	0	20
40	3	0	-	15	170.75	0	20
40	4	2	16.96	17	51.69	0	20
40	5	0	-	20	34.25	0	20
50	3	0	-	32	622.46	41	1
50	4	0	-	40	584.15	1	18
50	5	0	-	56	306.90	2	19

Table 3 Results for instances with homogeneous w_j/p_j ratios.

algorithms, which may require an enormous number of nodes.

4 Extensions

In this section, we briefly discuss two other types of problems in the class (A) to which the column generation approach applies:

- problems with additive objective functions for which only a part of the schedule is relevant, like the weighted number of tardy jobs and total weighted late work;
- parallel machine problems with non-identical machines.

We indicate the major differences with the approach for the $P||\sum_{j=1}^{n} w_j C_j$ problem only.

4.1 Complementary objectives

The weighted number of late jobs and total weighted late work are two important objective functions in which jobs are not penalized as long as they are completed at or before their due dates. These objective functions are defined as follows. A job is called *late* if $C_j > d_j$, where d_j is the due date of J_j. The weighted number of late jobs is denoted by $\sum_{j=1}^{n} w_j U_j$, where $U_j = 1$ if J_j is late, and $U_j = 0$ otherwise. Total weighted late work is denoted by $\sum_{j=1}^{n} w_j V_j$ where V_j, the late work of J_j, is defined as the portion of work of J_j that is performed after its due date d_j. Accordingly, we have that $V_j = \min\{p_j, \max\{0, C_j - d_j\}\}$. In both cases, the objective functions are to be minimized.

For either problem, there is an optimal schedule in which each machine first performs the on-time jobs in order of non-decreasing due dates and then the late jobs in any sequence (Potts and Van Wassenhove [14]). The late jobs appear thus in the irrelevant part of the schedule, and in fact it does not matter if, when, and by what machine the late jobs are executed. These problems are therefore equivalent to *maximizing* $\sum_{j=1}^{n} w_j(1 - U_j)$, the weighted number of on-time jobs, and $\sum_{j=1}^{n} w_j(p_j - V_j)$, total weighted on-time work. These problems lend themselves much better for the column generation approach, since the pricing algorithm needs to focus then only on the on-time jobs. These complementary problems can then be formulated as maximizing

$$\sum_{s \in S} c_s x_s$$

subject to

$$\sum_{s \in S} a_{js} x_s \leq 1, \text{ for each } j = 1, \ldots, n, \tag{5}$$

and conditions (1) and (3). Note the sense of condition (5): machine schedules contain on-time jobs only. The pricing algorithm maximizes the reduced cost, which means that the initialization

of the recursion is different; apart from that, the recursion is essentially the same as the one described in Section 2.3, but a job will only be included in a machine schedule if it is (partially) on-time.

4.2 Non-identical machines

If the machines are not identical, then the processing time of J_j on M_i is p_{ij}, not p_j, for each i and j. Hence, the cost of a machine schedule then depends on the choice of the machine as well, and we may even have that the priority rule is not equal for all machines. This can be easily overcome by associating a different set of machine schedules to each machine. Let $S(i)$ denote the set of feasible machine schedules for machine M_i $(i = 1, \ldots, n)$. We need to adjust the formulation given in Section 2.2 only slightly to accommodate non-identical machine problems. The problem is formulated as minimizing

$$\sum_{i=1}^{m} \sum_{s \in S(i)} c_s x_s$$

subject to

$$\sum_{s \in S(i)} x_s = 1, \text{ for each } i = 1, \ldots, m, \tag{6}$$

$$\sum_{i=1}^{m} \sum_{s \in S(i)} a_{js} x_s = 1, \text{ for each } j = 1, \ldots, n, \tag{7}$$

$$x_s \in \{0, 1\}, \text{ for each } s \in S(i), i = 1, \ldots, m. \tag{8}$$

For the pricing algorithm, we need to perform the recursion m times, one time for each machine separately. Accordingly, the pricing algorithm runs in $O(n \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij})$ time.

In addition, the partitioning strategy that we proposed in Section 2.4 does not apply in case of non-identical machines. An effective alternative is available though: simply use a forward partitioning strategy where jobs are assigned to machines; this partitioning strategy can easily be combined with column generation.

5 Application to Air Traffic Management

Due to the continuing expansion of air traffic, Air Traffic Management (ATM) is confronted with increasing problems in the safe and efficient handling of traffic. This causes a growth of delays, especially during peak hours or bad weather. Improvement of ATM planning can help to make better use of the existing capacity of the air space and airports and therefore to reduce delays. For this reason a lot of research is performed on the application of well-known optimization methods to ATM planning problems, see for example Van Kemenade et al. [18] and Maugis [12].

As ATM planning includes scheduling, the application of optimization methods for scheduling problems can be very beneficial in the area. In this chapter, we show that the scheduling of arrival aircraft on runways of an airport is a parallel machine scheduling problem, and we discuss the application of the column generation approach to this problem.

Consider the runways of an airport as parallel machines. The jobs that have to be processed by the machines are the aircraft arriving at the airport. Since a landing aircraft causes turbulence, we have that after the landing of each aircraft it takes some time before the runway is free for the next aircraft. We consider the lengths of these waiting periods as processing times, and denote by p_j the time until the runway is free after the landing of aircraft j. Now the landing time of an aircraft corresponds to the start time of a job. Consequently, we denote the landing time of aircraft j by S_j. Each aircraft has a planned time of arrival. We consider this time as a due date d_j, and say that aircraft j is late if $S_j > d_j$. Note that this notion of lateness differs from the one in Section 4.1. It is not hard to see that with this notion of lateness the problem of minimizing the weighted number of late jobs, i.e., $\sum_{j=1}^{n} w_j U_j$, can be solved in the same way as the problem in Section 4.1, and hence by the column generation algorithm. The weighted number of late, i.e., delayed, aircraft can hence be minimized with the column generation algorithm.

However, since aircraft approach the airport at different times, we have to deal with release dates, which complicates the problem. The algorithm can only be applied to groups of aircraft with approximately the same release date. Consider a group of aircraft that are in the neighbourhood of their arrival airport, i.e., that are either flying in circles in the holding area or are approaching the airport. These aircraft can land within a short time from now, i.e., they have approximately the same release date. For such a group of aircraft, the weighted number of delayed aircraft can be minimized by the column generation algorithm.

6 Conclusion

Column generation algorithms have been shown to be useful for many intractable combinatorial optimization problems; see for an overview Barnhart, Johnson, Nemhauser, Savelsbergh, and Vance [2]. This paper shows that column generation is computationally attractive for parallel machine scheduling problems as well.

References

1. J.W. BARNES AND J.J. BRENNAN (1977). An improved algorithm for scheduling jobs on identical machines. *AIIE Transactions 9*, 25-31.
2. C. BARNHART, E.L. JOHNSON, G.L. NEMHAUSER, M.W.P. SAVELSBERGH, AND P.H. VANCE (1994). *Branch-and-price: column generation for solving huge integer programs*, Report COC-9403, Georgia Institute of Technology, Atlanta.
3. H. BELOUADAH AND C.N. POTTS (1994). Scheduling identical parallel machines to minimize total weighted completion time. *Discrete Applied Mathematics 48*, 201-218.
4. J. CARLIER (1987). Scheduling jobs with release dates and tails on identical machines to minimize the makespan. *European Journal of Operational Research 29*, 298-306.
5. L.M.A. CHAN, P. KAMINSKY, A. MURIEL, AND D. SIMCHI-LEVI (1995). *Machine scheduling, linear programming and list scheduling heuristics*, Working paper, Northwestern University, Chicago.
6. Z. CHEN AND W.B. POWELL (1995). *Solving Parallel Machine Total Weighted Completion Time Problems by Column Generation*, Working paper, Princeton University.
7. W.L. EASTMAN, S. EVAN, AND I.M. ISSACS (1964). Bounds for the optimal scheduling of N jobs on M processors. *Management Science 11*, 268-279.
8. S.E. ELMAGHRABY AND S.H. PARK (1974). Scheduling jobs on a number of identical machines. *AIIE Transactions 6*, 1-13.
9. R.L. GRAHAM, E.L. LAWLER, J.K. LENSTRA, AND A.H.G. RINNOOY KAN (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics 5*, 287-326.
10. A.M.A. HARIRI, C.N. POTTS AND L.N. VAN WASSENHOVE (1995). Single machine scheduling to minimize total weighted late work. *ORSA Journal on Computing 7*, 232-242.
11. E.L. LAWLER AND J.M. MOORE (1969). A functional equation and its application to resource allocation and sequencing problems. *Management Science 16*, 77-84.
12. L. MAUGIS *Mathematical Programming for the Air Traffic Flow Management Problem with en-route capacity.* Technical Report CENA.
13. C.N. POTTS AND L.N. VAN WASSENHOVE (1988). Algorithms for scheduling a single machine to minimize the weighted number of late jobs. *Management Science 34*, 843-858.
14. C.N. POTTS AND L.N. VAN WASSENHOVE (1992). Single machine scheduling to minimize total late work. *Operations Research 40*, 586-595.
15. M.H. ROTHKOPF (1966). Scheduling independent tasks on parallel processors. *Management Science 12*, 437-447.

16. S.C. SARIN, S. AHN, AND A.B. BISHOP (1988). An improved branching scheme for the branch-and-bound procedure of scheduling n jobs on m parallel machines to minimize total weighted flowtime. *International Journal of Production Research 26*, 1183-1191.
17. W.E. SMITH (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly 31*, 325-333.
18. C.H.M. VAN KEMENADE, C.F.W. HENDRIKS, J.N. KOK, AND H.H. HESSELINK (1995). Evolutionary computation in air traffic control planning. *Proceedings of the sixth International Conference of Genetic Algorithms*, 611-616, editor: S. Forrest. Morgan, Kaufmann, San Francisco, California, also as NLR Technical Publication 94565.
19. S.T. WEBSTER (1992). New bounds for the identical parallel processor weighted flow time problem. *Management Science 38*, 124-136.
20. S.T. WEBSTER (1995). Weighted flow time bounds for scheduling identical processors. *European Journal of Operational Research 80*, 103-111.

Appendices

A Implementation of the pricing algorithm

In the previous, we took advantage of the property that there is an optimal schedule in which no machine schedule finishes its last job before time $H_{\min}$. Accordingly, we solved the pricing problem by choosing the machine schedule s with smallest $F_n(t)$ value, where $H_{\min} \leq t \leq H_{\max}$. To satisfy this lower bound on t, we may need to add a job J_j to s, although $w_j C_j(s) - \lambda_j > 0$ and we are minimizing.

If we ignore the above property, then we know that there is a machine schedule s with minimum cost in which all jobs J_j have $w_j C_j(s) < \lambda_j$. We exploit this observation to reduce the empirical running time of the pricing algorithm. Define $\Delta_j = \lceil \lambda_j / w_j \rceil$ for each j $(j = 1, \ldots, n)$; we have that $\Delta_j \geq 1$, since we can discard all jobs J_j with $\lambda_j = 0$. Accordingly, if $\Delta_j \leq \min\{H_{\max}, P(j)\}$, then equation (4) can be replaced by

$$F_j(t) = \begin{cases} \min\{F_{j-1}(t), F_{j-1}(t-p_j) + w_j t - \lambda_j\}, & \text{for } t = 0, \ldots, \Delta_j - 1, \\ F_{j-1}(t), & \text{for } t = \Delta_j, \ldots, \min\{H_{\max}, P(j)\}. \end{cases}$$

Moreover, the optimal solution value is now found as

$$F^* = \min_{0 \leq t \leq H_{\max}} F_n(t).$$

We like to avoid the explicit computation and storage of all values $F_j(t)$ for $t \geq \Delta_j$, since they are all the same. On the other hand, the recurrence relation needs the value $F_j(t)$ when computing $F_{j+1}(t)$ and $F_{j+1}(t + p_{j+1})$, so we need a procedure to retrieve the proper value of $F_j(t)$ when it is needed. Note now that for any $t \geq \Delta_j$ we have that $F_j(t) = F_{j-a(j,t)}(t)$, where $j - a(j,t)$ is the index of the last job before J_j that cannot be discarded from the machine schedule with maximum value beforehand, that is, $a(j,t)$ is equal to the smallest value such that $t \leq \Delta_{j-a(j,t)} - 1$. Hence, we know that we did not exclude $F_{j-a(j,t)}(t)$ from the computation. In the same fashion, we have for any $t \geq \Delta_j + p_j$ that $F_j(t - p_j) = F_{j-b(j,t)}(t - p_j)$, where $j - b(j,t)$ is the index of the last job before J_j that may be included in the machine schedule with maximum value. Therefore, $b(j,t)$ is the smallest value such that $t - p_j \leq \Delta_{j-b(j,t)} - 1$. Accordingly, the recurrence relation is then for $j = 1, \ldots, n$, $t = 0, \ldots, \min\{\Delta_j - 1, P(j), H_{\max}\}$

$$F_j(t) = \min\{F_{j-a(j,t)}(t), F_{j-b(j,t)}(t - p_j) + w_j t - \lambda_j\}. \tag{9}$$

In order to make it work and to gain from this adjustment, we need to establish an efficient

procedure to find the $a(j,t)$ and $b(j,t)$ values for all j and the appropriate times t. Note we must have $a(j,0) = 1$ for any j $(j = 1,\ldots,n)$ and $a(j,t) \geq a(j,t-1)$. Hence, when computing $F_j(t)$ we check first if $a(j,t) = a(j,t-1)$; if it is not, then we increase the value $a(j,t)$ in steps of size one until $t \leq \Delta_{j-a(j,t)}$. Accordingly, the computation of $a(j,t)$ requires one check each time we perform computation (9) plus $O(n^2)$ operations altogether to find the values $a(j,t)$ if $a(j,t) \neq a(j,t-1)$. We can design a similar procedure to compute the values $b(j,t)$. Hence, the worst-case running time of the pricing algorithm remains the same. The average running time has been reduced, however, since we have restricted the range of the state variable t for which the recursion needs to be performed.