Parallel machine scheduling through column generation: minimax objective functions, release dates, deadlines, and/or generalized precedence constraints

Marjan van den Akker

Han Hoogeveen

Jules van Kempen

institute of information and computing sciences, utrecht university

technical report UU-CS-2005-027

www.cs.uu.nl

Parallel machine scheduling through column generation: minimax objective functions, release dates, deadlines, and/or generalized precedence constraints

J.M. van den Akker J.A. Hoogeveen J.W. van Kempen
Department of Information and Computing Sciences
Utrecht University
P.O. Box 80089, 3508 TB Utrecht, The Netherlands
marjan@cs.uu.nl, slam@cs.uu.nl, jwkempen@cs.uu.nl

June 27, 2005

Abstract

In this paper we describe a solution framework for a number of scheduling problems in which the goal is to find a feasible schedule that minimizes some objective function of the minimax type on a set of parallel, identical machines, subject to release dates, deadlines, and/or generalized precedence constraints. We determine a lower bound on the objective function in the following way. We first turn the minimization problem into a decision problem by putting an upper bound on the outcome value; the question is then 'Does there exist a feasible schedule for the given instance?'. The question of feasibility is identical to 'Are m machines enough to feasibly accommodate all jobs?'. We turn this decision problem into a minimization problem again by asking for the minimum number of machines that we need. This can be formulated as an integer linear programming problem that resembles the ILP-formulation of the cutting stock problem. For this problem we determine a high quality lower bound by applying column generation to the LP-relaxation; if this lower bound is more than m, then we can conclude infeasibility. To speed up the process, we compute an intermediate lower bound based on the outcome of the pricing problem. As the pricing problem is intractable for many variants of the original scheduling problem, we mostly solve it approximately by applying local search, but once in every 50 iterations, we solve it to optimality by formulating it as an integer linear programming problem using a time-indexed formulation that we solve using CPLEX such that we can compute our intermediate lower bound.

After having derived the lower bound on the objective function of the original scheduling problem, we try to find a matching upper bound by identifying

a feasible schedule with objective function value equal to this lower bound. Our computational results show that our lower bound is so strong that this is almost always possible. We are able to solve problems with up to 160 jobs and 10 machines in 10 minutes on average.

1980 Mathematics Subject Classification (Revision 1991): 90B35.
Keywords and Phrases: parallel machine scheduling, set covering formulation, linear programming, column generation, maximum lateness, release dates, precedence constraints, intermediate lower bounds, time-indexed formulation.

1 Introduction

In this paper we consider one of the basic problems in scheduling and project management; we refer to the book by Pinedo (2002) for an introduction to scheduling theory. We are given m parallel, identical machines, which are continuously available from time zero onwards and can process no more than one job at a time; these machines have to process n jobs, which are denoted by $J_1, \ldots, J_n$. Processing J_j requires one, arbitrary processor during an uninterrupted period of length p_j, which period must start at or after the given release date r_j and must be completed by the given deadline $\bar{d}_j$. Given a schedule σ, we denote the completion time of job J_j by $C_j(\sigma)$, and hence, we need for all jobs J_j that $r_j + p_j \leq C_j(\sigma) \leq \bar{d}_j$ for σ to be feasible. Moreover, the jobs may be subject to generalized precedence constraints, which prescribe that for a pair of jobs J_i and J_j the difference in completion time $C_j(\sigma) - C_i(\sigma)$ should be at least (at most, or exactly) equal to some given value q_{ij}. The quality of the schedule is measured by some objective function of minimax type, which is assumed to be nondecreasing in the completion times, like maximum lateness or maximum cost. Here, the maximum lateness is defined as $\max_j L_j(\sigma)$, where $L_j(\sigma) = C_j(\sigma) - d_j$; d_j signals the due date, by which the job preferably should be completed. A special case occurs when all due dates are equal to zero; in this case, the objective function becomes equal to minimizing the maximum completion time, that is, the makespan of the schedule.

We solve these problems by applying the technique of column generation. This approach has been shown to work very well for the problem of minimizing total weighted completion time on a set of identical parallel machines (see Van den Akker, Hoogeveen, and Van de Velde (1999) and Chen and Powell (1999)), and since the appearance of these papers, the method of applying column generation has been applied to many parallel machine problems with a sum type criterion in which the jobs are known to follow a specific order on the individual machines; we refer to Van den Akker, Hoogeveen, and Van de Velde (2005) for an overview. One notable exception is due to Brucker and Knust (2000, 2002, 2003), who apply column generation to a number of resource constraint project scheduling problems in which the goal is to minimize the makespan. Here they first formulate the problem as a decision problem and then use linear programming to check whether it is possible to execute all jobs in a feasible preemptive schedule; here the decision variables refer to the length of a time slice during which a given set of jobs is executed simultaneously.

We use the three-field notation scheme introduced by Graham, Lawler, Lenstra, and Rinnooy Kan (1979) to denote scheduling problems. The remainder of this chapter is organized as follows. In Section 2, we describe the basic approach for the relatively simple problem of minimizing $L_{\max}$ without release dates and generalized precedence constraints. We explain the column generation approach and the derivation of an intermediate lower bound in Sections 3 and 4. In Sections 5 and 6, we add release dates and generalized precedence constraints. In Section 7, we describe our local search algorithm to solve the pricing problem approximately, and in Section 8 we formulate the pricing problem as a time-indexed integer linear programming problem that can be used to find (an upper

bound on) the solution of the pricing algorithm. Finally, in Section 9 we draw some conclusions.

Our contribution. We give the first algorithm for solving this kind of problems using column generation. Our approach is a bit complementary to the approach by Brucker and Knust, since they check the existence of a preemptive schedule for a given set of resources, whereas we let the number of machines vary. Moreover, we describe an efficient way to use an intermediate lower bound to be able to make a decision without having to solve the LP-relaxation to optimality.

2 The basic approach

In this section, we sketch the basic approach, which we illustrate on the $P||L_{\max}$ problem, that is, there are m parallel, identical machines to execute n jobs, where the objective is to minimize maximum lateness; there are no release dates and precedence constraints, but there can be deadlines.

It is well-known that this optimization problem can be solved by solving a set of decision problems, which are obtained by putting an upper bound L on the value of the objective function. Since the restriction $L_{\max} \leq L$ is equivalent to the constraint that $L_j = C_j - d_j \leq L$ for each job J_j, we find that $C_j \leq d_j + L \equiv \bar{d}_j$; the decision problem is then to determine whether there exists a feasible schedule meeting all deadlines, where we take the minimum of the original deadline and the deadline induced by the constraint $L_{\max} \leq L$. Hence, we can solve the optimization problem by determining the smallest value L that allows a feasible schedule.

Since the machines are identical, the decision problem can be reformulated as: *is it possible to partition the jobs in at most m subsets such that for each subset we can find a feasible single-machine schedule that meets all deadlines*? Checking the feasibility of a subset is easy by executing the jobs in order of earliest deadline order and see whether these are all met (Jackson, 1955); hoping not to confuse the reader, we call this the ED-order . Note that it is identical to the earliest due date (EDD) order if the original deadlines are not restrictive. We solve this decision problem by answering the question: what is the minimum number of machines that we need to get a feasible schedule? Or equivalently, into how many feasible single-machine schedules do we have to partition the jobs?

Given a subset of jobs, it is easy to find the corresponding single-machine schedule by putting the jobs in ED-order. Therefore, we also call a subset of jobs that allows a feasible single-machine schedule a *machine schedule*. The above question is then to find the minimum number of mutually distinct machine schedules that contain all jobs. We can formulate the above problem as an integer linear programming problem as follows. Let S be the set containing all machine schedules. We introduce binary variables x_s ($s = 1, \ldots, |S|$) that take value 1 if machine schedule s is selected and 0 otherwise. Each machine schedule s is encoded by a vector $a_s = (a_{1s}, \ldots, a_{ns})$, where $a_{js} = 1$ if machine schedule s contains job J_j and $a_{js} = 0$, otherwise. We have to minimize the number

of machine schedules that we select, such that each job is contained in one machine schedule. Hence, we have to determine values x_s that solve the problem

$$\min \quad \sum_{s \in S} x_s$$

subject to

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

$$x_s \in \{0, 1\}, \text{ for each } s \in S. \tag{2}$$

We obtain the linear programming relaxation by replacing conditions (2) by the conditions $x_s \geq 0$ for all $s \in S$; we do not need to enforce the upper bound of 1 for x_s, since this follows immediately from the conditions (1). We solve the LP-relaxation using column generation.

3 Column generation

We first solve the linear programming relaxation for a small initial subset of the columns. Given the solution to the linear programming problem with the current set of variables, it is well-known from the theory of linear programming (see for instance Bazaraa, Jarvis, and Sherali (1990)) that the reduced cost of a variable x_s is given by

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

where $\lambda_1, \ldots, \lambda_n$ are the dual multipliers corresponding to the constraints (1) of the solution of the current LP. If for each variable x_s we have that $c'_s \geq 0$, then the solution with the current set of variables solves the linear programming problem with the complete set of variables as well. To check whether this condition is fulfilled, we minimize the reduced cost over all machine schedules. Therefore, we must pick the subset of the jobs with maximum total dual multiplier value among all subsets of jobs that lead to a feasible single-machine schedule, that is, we must solve the problem

$$\max_{s \in S} \sum_{j=1}^{n} \lambda_j a_{js} \tag{3}$$

We use $\hat{c}$ to denote the outcome value of this problem; hence, we have that the minimum reduced cost, which we denote by c^*, is equal to

$$c^* = 1 - \hat{c}$$

This maximization problem is equivalent to the problem of minimizing the total weight of the jobs that are not selected, which is known as the problem of minimizing the weighted

number of tardy jobs, where the weight of a job is equal to the dual multiplier λ_j and the due date for each job is equal to the deadline $\bar{d}_j$. Note here that the constraint that each weight is nonnegative in this scheduling problem is not restrictive, since a job with negative weight will never be selected in the maximization problem. This problem, which is denoted as $1||\sum w_j U_j$ in the three-field notation scheme, is solvable in $O(n\sum p_j)$ time by the dynamic programming algorithm of Lawler and Moore (1969). Hence, in this situation we solve the pricing problem to optimality. If $c^* \geq 0$, then we have solved the linear programming relaxation; otherwise, we add the variable with minimal reduced cost value to the LP and solve it again. In this way, we solve the linear programming relaxation to optimality. If the outcome value is more than m, then we know that the answer to the decision problem is 'no'; if the outcome value is no more than m, and we have not identified a feasible solution yet that uses m (or fewer) machines, then we solve the integer linear programming problem to optimality using the branch-and-bound algorithm developed by Van den Akker, Hoogeveen, and Van de Velde (1999) for the problem $P||\sum w_j C_j$.

4 An intermediate lower bound

In the above implementation we have to apply column generation to the bitter end, that is, until we have concluded that the linear programming relaxation has been solved to optimality, before we have found a valid lower bound. Since we only need to know whether the outcome value is more than m or no more than m, we are not interested in the exact outcome value, as long as it allows us to decide the decision problem. Fortunately, it is possible to compute an *intermediate lower bound*. This procedure works as follows (see also Bazaraa, Jarvis, and Sherali (1990) for a general description of this principle).

Since c^* is the outcome value of the pricing problem, we know that for each $s \in S$ we have

$$1 = c_s = c'_s + \sum_{j=1}^{n} \lambda_j a_{js} \geq c^* + \sum_{j=1}^{n} \lambda_j a_{js}$$

We use this expression to find a lower bound for the objective function of the linear programming relaxation as follows

$$\sum_{s\in S} x_s \geq \sum_{s\in S} (c^* + \sum_{j=1}^{n} \lambda_j a_{js}) x_s = c^* \sum_{s\in S} x_s + \sum_{s\in S} \sum_{j=1}^{n} \lambda_j a_{js} x_s =$$

$$c^* \sum_{s\in S} x_s + \sum_{j=1}^{n} \lambda_j \left[\sum_{s\in S} a_{js} x_s \right] = c^* \sum_{s\in S} x_s + \sum_{j=1}^{n} \lambda_j$$

where we use constraint (1). Therefore, we find that

$$(1 - c^*) \sum_{s\in S} x_s \geq \sum_{j=1}^{n} \lambda_j$$

Since $1 - c^* = \hat{c}$, which value is larger than 1, since $c^* < 0$, we find that

$$\sum_{s \in S} x_s \geq \sum_{j=1}^{n} \lambda_j / \hat{c}$$

This gives the desired intermediate lower bound, which we can use to decide whether m machines are sufficient. If this lower bound has value smaller than or equal to m, then we continue with solving the LP-relaxation.

Solving the problem $P||L_{\max}$ was relatively simple, since each machine schedule can be represented by just listing the indices of the jobs that it contains, and since the column generation problem can be solved by applying dynamic programming. We can use the same methodology to solve any problem for which putting an upper bound on the objective function results in a set of deadlines. Hence, we can solve the more general $P||f_{\max}$ problem in the same fashion, where $f_{\max}$ denotes maximum cost, which is defined as $\max_j f_j(C_j)$, where $f_j(t)$ is the cost function of job J_j, which is assumed to be nondecreasing in t. In this case, the deadline $\bar{d}_j$ for each job J_j $(j = 1, \ldots, n)$ is the largest integral value of t such that $f_j(t) \leq F$. This value t can easily be computed if the function $f_j(t)$ has an inverse, and we can find it using binary search otherwise. Since the problem $P|r_j|C_{\max}$ is the mirror problem of $P||L_{\max}$, we can solve this problem through our basic approach as well.

5 Including release dates: $P|r_j|L_{\max}$

In this section we assume that next to the deadlines, which may come from putting an upper bound on $L_{\max}$, there are release dates. When we follow the basic approach that we worked out for $P||L_{\max}$, then we need to find a set of at most m machine schedules that contain all jobs and that obey all release dates and deadlines. Since we have both release dates and deadlines, we cannot easily define the corresponding machine schedule if we know the set of jobs it consists of. Therefore, we represent a machine schedule s by listing the completion times of the jobs that it contains next to the vector a_s, which has $a_{js} = 1$ if job J_j is contained and $a_{js} = 0$, otherwise.

The problem of minimizing the number of machines that we need can then be formulated as an integer linear programming problem, which is the same as the one in Section 2. Hence, when we solve the problem using column generation, we get the same pricing problem, but now we have to construct a machine schedule with maximum weight that obeys the release dates and deadlines. This boils down to the well-known scheduling problem $1|r_j|\sum w_j U_j$, which is $\mathcal{NP}$-hard in the strong sense. We do not want to solve this to optimality, but we use local search to find an approximately optimal solution to the pricing problem, which we use in our column generation; see Section 7 for a description. Note that the derivation of the intermediate lower bound is still valid, but to compute it, we need the value $\hat{c}$, which is the optimum of the maximization problem (3), and not just a lower bound on this, which we find using local search. To this end, we compute an upper bound on or the exact value of the outcome of the maximization

problem in Section 8, which we can plug in for $\hat{c}$ to find a feasible lower bound on the number of machines that we need to feasibly accommodate all jobs. Before showing the details of the local search and the upper bound, we first discuss how to tackle the problem in which there are generalized precedence constraints as well.

6 The full problem

In this section, we assume that there are release dates, deadlines, and generalized precedence constraints. We again translate the problem into one of minimizing the number of machine schedules that are needed. Since two jobs that are connected through a precedence constraint do not have to be executed by the same machine, we assume that the machine schedules obey the release dates and deadlines, and we include a constraint in the integer linear programming formulation for each of the generalized precedence constraints. We define A^1 as the arc set containing all pairs (i, j) such there exists a precedence constraint of the form $C_j - C_i \geq q_{ij}$; similarly, we define A^2 and A^3 as the arc sets that contain an arc for each pair (i, j), for which $C_j - C_i \leq q_{ij}$ and $C_j - C_i = q_{ij}$, respectively. Note that the intersection of A^1 and A^2 does not have to be empty. We denote the union of A^1, A^2, and A^3 by the multiset A. This leads to the following integer linear programming formulation

$$\min \quad \sum_{s \in S} x_s$$

subject to

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

$$\sum_{s \in S} C_{js} x_s - \sum_{s \in S} C_{is} x_s \geq q_{ij} \text{ for each } (i, j) \in A^1;$$

$$\sum_{s \in S} C_{js} x_s - \sum_{s \in S} C_{is} x_s \leq q_{ij} \text{ for each } (i, j) \in A^2;$$

$$\sum_{s \in S} C_{js} x_s - \sum_{s \in S} C_{is} x_s = q_{ij} \text{ for each } (i, j) \in A^3;$$

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

Here C_{js} denotes the completion time of job J_j in column s, which we define to be equal to 0 if J_j is not contained in s. If we want to solve the LP-relaxation by applying column generation, then we find that the reduced cost of a machine schedule s is equal to

$$c'_s = c_s - \sum_{j=1}^{n} a_{js} \lambda_j - \sum_{j=1}^{n} \left[\sum_{h \in P_j} \delta_{hj} C_{js} - \sum_{k \in S_j} \delta_{jk} C_{js} \right].$$

Here the sets P_j and S_j are defined as the sets containing all predecessors and successors of job J_j in A, respectively. Hence, we must solve the maximization problem

$$\sum_{j=1}^{n} a_{js}\lambda_j + \sum_{j=1}^{n}\left[\sum_{h\in P_j}\delta_{hj}C_{js} - \sum_{k\in S_j}\delta_{jk}C_{js}\right].$$

over all machine schedules $s \in S$. We solve this problem approximately using local search (see Section 7). Again, we can compute an intermediate lower bound. Let $\hat{c}$ denote the outcome value of the maximization problem, and let c^* denote the minimum reduced cost. Hence, we find that

$$c_s \geq c^* + \sum_{j=1}^{n} a_{js}\lambda_j + \sum_{j=1}^{n}\left[\sum_{h\in P_j}\delta_{hj}C_{js} - \sum_{k\in S_j}\delta_{jk}C_{js}\right].$$

We plug this in, and use that

$$\sum_{j=1}^{n}\left[\sum_{h\in P_j}\delta_{hj}C_{js}x_s - \sum_{k\in S_j}\delta_{jk}C_{js}x_s\right] = \sum_{(j,k)\in A}\delta_{jk}\left[C_{ks}x_s - C_{js}x_s\right].$$

We then obtain

$$\sum_{s\in S} c_s x_s \geq c^*\sum_{s\in S} x_s + \sum_{j=1}^{n}\lambda_j\left[\sum_{s\in S} a_{js}x_s\right] + \sum_{(j,k)\in A}\delta_{jk}\left[\sum_{s\in S} C_{ks}x_s - C_{js}x_s\right] \geq$$

$$c^*\sum_{s\in S} x_s + \sum_{j=1}^{n}\lambda_j + \sum_{(j,k)\in A}\delta_{jk}q_{jk},$$

where we use that in case of a constraint with $\geq$ sign the dual multiplier has value ≥ 0, whereas in case of a constraint with $\leq$ sign the dual multiplier is ≤ 0. Hence, we may conclude that

$$\sum_{s\in S} x_s \geq \left[\sum_{j=1}^{n}\lambda_j + \sum_{(j,k)\in A}\delta_{jk}q_{jk}\right]/\hat{c}$$

is an intermediate lower bound on the number of machines that we need.

If we get stuck, that is, the outcome of the LP-relaxation does not lead to 'no' on the decision problem, then we assume for the time-being that the decision problem is feasible, and we decrease the upper bound L on $L_{\max}$ that we want to test. If we end up with a value L for which we know that $L-1$ yields an infeasible decision problem and for which the LP-relaxation cannot decide whether the decision problem obtained by putting the upper bound on $L_{\max}$ equal to L is feasible, then we can apply branch-and-bound. Here we use the branching strategy developed by Carlier (1987) of splitting the execution interval $[r_j, \bar{d}_j]$ into two disjunct intervals $[r_j, Q]$ and $[Q - p_j + 1, \bar{d}_j]$, where Q is some time point in the middle of the interval $[r_j, \bar{d}_j]$. It turned out in our experiments, however, that it is better to solve an integer linear programming formulation in which we request that $L_{\max} = L$ by using CPLEX (see Section 9).

7 Generating new columns by local search

In this section, we describe the local search algorithm that we have implemented to solve the pricing problem, which is the problem of finding the feasible single-machine schedule s that minimizes

$$1 - \sum_{j=1}^{n} a_{js}\lambda_j - \sum_{j=1}^{n} \left[\sum_{h \in P_j} \delta_{hj} C_{js} - \sum_{k \in S_j} \delta_{jk} C_{js} \right]. \tag{4}$$

Solving the original pricing problem is equivalent to finding the single-machine schedule that obeys the release dates and deadlines and maximizes

$$\sum_{j=1}^{n} \lambda_j a_{js} + \sum_{j=1}^{n} C_{js} \left[\sum_{h \in P_j} \delta_{hj} - \sum_{k \in S_j} \delta_{jk} \right]. \tag{5}$$

If we define $Q_j = \sum_{h \in P_j} \delta_{hj} - \sum_{k \in S_j} \delta_{jk}$, then we have to maximize

$$\sum_{j=1}^{n} \lambda_j a_{js} + \sum_{j=1}^{n} Q_j C_{js} \tag{6}$$

Looking at this formula we see that, if job J_j gets selected, then this Q_j value determines whether it is more profitable to execute the job as late as possible ($Q_j > 0$) or as early as possible ($Q_j < 0$). In a preprocessing step, we can even determine the time interval during which we must complete job J_j, if selected, since its total contribution to the objective function would be negative otherwise, in which case it would have been better not to select J_j.

In our local search we use a two-phase procedure. In the first phase, we determine the jobs that are selected and the order in which they are executed. In the second phase, we then determine the optimal set of completion times. We first discuss the procedure used in the second phase. Since the values Q_j are fixed, we can solve this subproblem in linear time using the following shifting procedure, which resembles the procedure for a similar problem given by Garey, Tarjan, and Wilfong (1988).

Procedure **Optimize completion times**

1. Consider the selected jobs in the given order and place all jobs as early as possible, that is, each job is started at the maximum of its release date and the completion time of its predecessor.

2. Look at the jobs in reversed order. Suppose that we are currently working on job J_j. If $Q_j > 0$, we will delay the job until either

   (a) Job J_j reaches its deadline: we put $C_j = \bar{d}_j$.

(b) Job J_j *hits* its successor: J_j is *glued* to its successor J_k (which may already have been glued to some of its successors) to form a new job J_{new} with $Q_{new} = Q_j + Q_k$ and the new deadline is the deadline that is reached first when this job J_{new} is shifted towards the end of the schedule.

i. If $Q_{new} > 0$, we try to shift J_{new} further to the right as described before.

ii. If $Q_{new} \leq 0$, J_{new} stays at this position.

Note that if and only if there is no placement possible for the given selection and order, step (1) will be forced to end a job after its deadline. It is straightforward to show that the above procedure finds the optimum set of completion times for the given selection of jobs and the given order in which these are scheduled. Since in each step we either take a job on an earlier position into consideration, or we glue together some jobs, we are done after $O(n)$ steps, where n is the number of jobs that have been selected. Since each step can be implemented to run in constant time, the algorithm runs in linear time.

We now describe the first step of the local search procedure. We define a solution in our local search as a selection of the jobs and the order in which they should be processed, after which we find the value of this solution by solving the second step. Our local search uses the following methods to exploit the solution space:

(i) Remove a random job from the set of selected jobs;

(ii) Add a random, yet unselected job at a random place in the order of selected jobs;

(iii) Replace a random job from the current selection by a random, yet unselected job;

(iv) Swap the positions of two random jobs in the set of selected jobs.

In our computational experiments, we added up to 50 columns with negative reduced cost per iteration. We computed the intermediate lower bound this was conducted after each

8 Time indexed formulation

The last item on the list is to find an upper bound for $\hat{c}$, which is defined as the outcome of the maximization problem (5), such that we can compute the intermediate lower bound. To this end, we formulate the problem as an integer linear programming problem using a time-indexed formulation. Here we have variables x_{jt} ($j = 1, \ldots, n; t = r_j, \ldots, \bar{d}_j - p_j$), which get value 1 if job J_j is started at time t, and value 0 otherwise. The corresponding cost coefficient c_{jt} is easily determined. We need constraints to enforce that each job is started at most once and that the machine executes at most one job at a time. The corresponding ILP-formulation is

$$\max \sum_{j=1}^{n} \sum_{t=r_j}^{\bar{d}_j - p_j} c_{jt} x_{jt}$$

subject to

$$\sum_{t=r_j}^{\bar{d}_j - p_j} x_{jt} \leq 1 \quad \forall j = 1, \ldots, n$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1 \quad \forall t = 1, \ldots, T$$

$$x_{jt} \in \{0, 1\} \quad \forall j = 1, \ldots, n; \forall t = r_j, \ldots, \bar{d}_j - p_j$$

Here T denotes the largest time at which at least two jobs can be executed. In our experiments, we computed this upper bound every 50 iterations, or when our local search algorithm could not find any column with negative reduced cost. It is well known (see for instance Sousa and Wolsey (1992) and Van den Akker (1994)) that solving the LP-relaxation gives a very strong upper bound if *each job must be executed*. In our case, however, we can (partly) select jobs. In some cases, the possibility of executing a job partly decreases the quality of the upper bound, which disables the intermediate lower bound to decide the feasibility problem. If in this situation our local search algorithm cannot find a column with negative reduced cost, then we get stuck. To avoid this, we then turn to the original ILP formulation of the pricing problem. We compute for which value of the objective function of the pricing we find an intermediate lower bound equal to m. We then ask our ILP solver CPLEX whether there exists a solution to the pricing problem with this value or larger. If the answer to this decision problem is 'no', then we can conclude that m machines are not enough; if the answer is 'yes', then we add the corresponding column and continue with solving the LP-relaxation by column generation. In general, we do not solve the ILP formulation of the pricing problem to optimality. It turns out that the increased running time of solving the ILP instead of the LP does not add significantly to the total running time, for the number of ILP solves needed is limited.

9 Computational results

Compared methods

Since we could not find other results of reports trying to solve the problem $P|r_j, prec|L_{max}$, we have compared our method to the rather straightforward and direct approach using a time-indexed ILP formulation of this problem like the one stated in Section 8.

$$\min \quad L$$

subject to

$$\sum_{t=r_j}^{\bar{d}_j - p_j} x_{jt} = 1, \text{ for each } j = 1, \ldots, n;$$

$$\sum_{j=1}^{n} \sum_{t=t'-p_j+1}^{t'} x_{jt'} \leq m, \text{ for each } t' = 0, \ldots, T;$$

$$\sum_{t=r_j}^{\bar{d}_j - p_j} x_{jt} t + p_j - d_j \leq L, \text{ for each } j = 1, \ldots, n;$$

$$\sum_{t=r_j}^{\bar{d}_j - p_j} x_{jt} t - \sum_{t=r_i}^{\bar{d}_i - p_i} x_{it} t \geq q_{ij}, \text{ for each } (i,j) \in A^1;$$

$$\sum_{t=r_j}^{\bar{d}_j - p_j} x_{jt} t - \sum_{t=r_i}^{\bar{d}_i - p_i} x_{it} t \leq q_{ij}, \text{ for each } (i,j) \in A^2;$$

$$\sum_{t=r_j}^{\bar{d}_j - p_j} x_{jt} t - \sum_{t=r_i}^{\bar{d}_i - p_i} x_{it} t = q_{ij}, \text{ for each } (i,j) \in A^3;$$

$$x_{jt} \in \{0,1\}, \text{ for each } j = 1, \ldots, n \text{ and each } t = 0, \ldots, T.$$

To reduce the number of variables in the above formulation, we only allow L to be smaller than or equal to a some value of L' for which a feasible solution has been found through some heuristic; this is achieved by using deadlines $\bar{d}_j$ that are equal to $d_j + L'$. Moreover, we check the number of jobs that are being executed at time t' only until we reach the time point at which we are sure that no more than m jobs can be executed simultaneously.

When we compared the column generation approach that we had originally in mind, that is, with the branch-and-bound based on splitting the execution intervals, to the method of using CPLEX to solve this direct ILP formulation, we noticed that these methods have difficulty with exactly opposite problems. We noticed that for all our testing instances the lower bound on L_{max} found by the LP-relaxation coincided with the optimal value of L_{max}. The problem with our method is that it is often not able to generate a set of columns in the LP-relaxation that form a solution to the original ILP formulation. CPLEX has exactly the opposite problem and mostly has a very hard time to conclude that a solution is optimal. Therefore, we tried to exploit the best of both worlds in defining a *hybrid method*, using the very strong lower bound LB on L_{max} found by our column generation method and then let CPLEX find a solution with this value (thus adding the constraint $L = LB$ and ignoring a big part of the variables).

Number	p_j	r_j	d_j	n	m	# prec
0	U[1,20]	U[0,60]	U[50,80]	40	4	20
1	U[1,20]	U[0,40]	U[30,60]	70	5	35
2	U[1,20]	U[0,80]	U[80,150]	80	7	30
3	U[1,20]	U[0,40]	U[60,80]	100	9	40
4	U[1,20]	U[0,60]	U[80,110]	120	9	50
5	U[1,20]	U[0,60]	U[80,110]	140	10	50
6	U[1,20]	U[0,60]	U[80,110]	160	10	50
7	U[1,20]	U[0,60]	U[80,110]	180	10	60
8	U[1,20]	U[0,60]	U[40,80]	60	3	30
9	U[1,20]	U[0,60]	U[40,80]	60	5	30
10	U[1,20]	U[0,60]	U[40,80]	60	7	30
11	U[1,20]	U[0,60]	U[50,80]	30	3	15
12	U[1,40]	U[0,120]	U[100,160]	30	3	15

Table 1: Test scenario's

Results

In our experiments we compare our hybrid algorithm to the direct ILP solved by CPLEX. We have applied both algorithms on 13 scenarios; for each scenario we ran five test instances. The scenario's are described in Table 1; n denotes the number of jobs, m the number of machines, and # prec denotes the number of precedence constraints. The first 8 scenario's are used to compare our hybrid algorithm to CPLEX. The scenario's 8-10 are used to find out the influence of the number of machines, whereas in the last two the influence of a doubling of the times value is measured. The results of the experiments are summarized in Table 2. The results of the hybrid algorithm are denoted in the row starting with Hi, where i denotes the number of the scenario; the results of CPLEX (Version 9.0) for the ILP formulation appear in the row starting with Ci. The algorithms were encoded in Java (Version 1.4.2_05) and the experiments were run on a Dell Optiplex GX270 P4 2,8 Ghz computer. For each instance we let each algorithm run for at most 30 minutes. We keep track of the number of times out of 5 that an optimum was found ('# success') and also the average and maximum amount of time in seconds needed for the successful runs ('Avg t' and 'Max t'). For the hybrid algorithm, we have gathered some more information, by ('#LB=OPT') we denote the number of times that we could prove that the lower bound equalled the optimum; also we denoted the proven difference between the optimum and the lower bound ('Max dif'). Next, we measured the average and maximum time needed to find the lower bound for the successful runs ('Avg t LB' and 'Max t LB'). Finally, by ('Avg #ILP' and 'Max #ILP'), we denote the number of times that we solved the ILP formulation of the pricing problem; this was conducted after each series of 50 runs of the local search algorithm, since we wanted to find out whether the intermediate lower bound could decide the problem already, and whenever

the local search algorithm could not find an improving column.

	# success	Avg t	Max t	#LB=OPT	Max dif	Avg t LB	Max t LB	Avg #ILP	Max #ILP
H0	5	66	194	5	0	38	92	33	77
C0	2	27	53						
H1	4	53	170	4	0	14	26	4	16
C1	3	30	37						
H2	5	153	396	5	0	83	180	47	139
C2	3	231	645						
H3	5	342	1109	5	0	110	174	14	33
C3	0	-	-						
H4	5	342	393	5	0	183	302	38	57
C4	0	-	-						
H5	5	452	689	5	0	228	269	25	41
C5	0	-	-						
H6	5	636	1045	5	0	354	415	29	37
C6	0	-	-						
H7	1	553	553	1	0	470	470	27	27
C7	0	-	-						
H8	3	199	296	3	0	158	266	40	98
C8	0	-	-						
H9	5	88	197	5	0	53	85	17	42
C9	2	80	351						
H10	5	6	9	5	0	5	8	0	0
C10	5	2	3						
H11	5	31	55	5	0	24	35	15	50
C11	5	92	180						
H12	4	101	150	4	0	73	147	24	54
C12	0	-	-						

Table 2: Results of comparing CPLEX and the hybrid algorithm

We also tested the performance of our local search algorithm on the pricing problem by comparing it to the method of only generating columns by solving the ILP of the pricing problem. For the scenario's 0, 5, 11 and 12 we ran 5 instances each. We determined the lower bound on these instances by using our local search algorithm and by using the optimal solutions to the ILP only. The results are depicted in Table 3. Here Hi denotes the hybrid algorithm run on scenario i, where Ii denotes the results obtained on scenario i by the algorithm in which the ILP of the pricing algorithm. Scenario 0 is used to show the difference for *easy* instances, where scenario 5 is used to investigate *difficult* instances. Finally scenario's 11 and 12 are used to investigate the influence of

the doubling of time values on the results.

	# success	Average time	Maximum time
H0	5	36	89
I0	5	108	230
H5	5	190	298
I5	0	-	-
H11	5	28	48
I11	5	88	127
H12	5	51	91
I12	4	307	506

Table 3: Results of comparing LS to only ILP solving

Evaluation of our experiments

Our results clearly show our hybrid algorithm outperforms the method of letting CPLEX solve the full ILP by far. CPLEX is not able to solve the full ILP in less than 30 minutes for most of the tested instances, where our hybrid algorithm easily solves nearly all instances. Looking at scenario 3 we already see that CPLEX fails to solve any of the 5 instances with 100 jobs and 9 machines within 30 minutes, where our hybrid algorithm solves all instances we tested up to 160 jobs and 10 machines (scenarios 3-6).

Our results also show that for all instances we managed to solve, the derived lower bound was equal to the optimal value. There are some instances for which we could not check whether optimum and lower bound coincided, for we could not solve them within 30 minutes. It seems reasonable that in at least some of these cases this is due to the fact that the lower bound was not strict. However, we never were able to show that the lower bound differed from the optimum for any instance. Altogether we may draw the conclusion that our lower bound is extremely strong.

If we compare the CPLEX algorithm with the second part of the hybrid algorithm, then we see that specifying the optimum makes a lot of difference. If we for instance stopped an instance of C5, then the best found upper bound so far in general was way off the optimum. This may be explained partly by the reduction in size of the model, but it is most certainly also due to the preprocessing steps performed by CPLEX. Therefore, we may expect the technique of constraint satisfaction to work very well to find a solution of value L' if such solution exists.

The hardness of the problem seems to depend mostly on the number of jobs per machine: if we look at scenarios 8-10 we can see that 20 jobs per machine gets really difficult. However, doubling the time values (scenarios 11 and 12) adds a relatively little increase to the average time needed to solve an instance, but one problem becomes unsolvable for our hybrid algorithm. But also here our hybrid algorithm shows its merit in comparison to the CPLEX method, for doubling the times makes CPLEX incapable

to solve any of the instances: it does not even find any solution for these instances, which is of course due to the large increase in variables in the ILP model.

Table 2 already shows the quality of our local search algorithm since the number of (costly) ILP solves of the pricing problem is limited and does not seem to depend much on the size or difficulty of the problem. Table 3 further validates that our local search algorithm performs very well, for without the local search algorithm *easy* instances already take 3 times as much time to compute the lower bound. And *difficult* instances even become unsolvable within 30 minutes, while our local search algorithm only needs a little more than 3 minutes on average to compute the lower bound for these instances. Next, doubling the time values also doubles the time needed to compute the lower bound, where using only ILP solves quadruples the average time for 4 instances and is not able to compute a lower bound for one instance within 30 minutes.

10 Conclusion and future research

We have described how the parallel machine scheduling with identical machines and a minimax objective function can be solved by applying column generation, even in the presence of complications like release dates and precedence constraints. We have tested the algorithm on the objective of maximum lateness and with greater than or equal precedence constraints only. We expect (but this is only a guess) that problems with maximum cost are harder to solve, since changing the upper bound on the cost with a small amount does not necessarily change the deadlines of all jobs. We expect that the nature of the precedence constraints does not change the effectiveness of the algorithm. It is an interesting, nontrivial step to extend this algorithm to the case with *uniform*, or even *unrelated* machines.

The next step in the research will be to investigate the natural connection with constraint satisfaction, which for instance can be used to tighten the release dates and deadlines (see for instance the book by Baptiste, Le Pape, and Nuijten, 2001). This looks a very promising direction to improve the effectiveness of the algorithm, as already witnessed by the success of the preprocessing phase of the CPLEX algorithm.

References

[1] J.M. van den Akker (1994). *LP-based solution methods for single-machine scheduling problems*, PhD Thesis, Eindhoven University of Technology.

[2] J.M. van den Akker, J.A. Hoogeveen, and S.L. Van de Velde (1999). Parallel machine scheduling by column generation. *Operations Research 47*, 862–872.

[3] J.M. van den Akker, J.A. Hoogeveen, and S.L. Van de Velde (2005). Applying column generation to machine scheduling. G. Desaulniers, J. Desrosiers, and M.M. Solomon (eds.). *Column Generation*, Springer, 303–330.

[4] P. Baptiste, C. Le Pape, and W. Nuijten (2001). *Constraint-based scheduling*: Applying constraint programming to scheduling problems. Kluwer Academic Publishers, Dordrecht, The Netherlands.

[5] M.S. Bazaraa, J.J. Jarvis, and H.D. Sherali (1990). *Linear Programming and Network Flows*, Wiley, New York.

[6] P. Brucker and S. Knust (2000). A linear programming and constraint propagation-based lower bound for the RCPSP. *European Journal of Operational Research 127*, 355–362.

[7] P. Brucker and S. Knust (2002). Lower bounds for scheduling a single robot in a job-shop environment. *Annals of Operations Research 115*, 147–172.

[8] P. Brucker and S. Knust (2003). Lower bounds for resource-constrained project scheduling problems. *European Journal of Operational Research 149*, 302–313.

[9] 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.

[10] Z.L. Chen and W.B. Powell (1999). Solving parallel machine scheduling problems by column generation. *INFORMS Journal on Computing 11*, 78–94.

[11] M.R. Garey, R.E. Tarjan, G.T. Wilfong (1988). One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research 13*, 330–348.

[12] 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.

[13] J.R. Jackson (1955). *Scheduling a production line to minimize maximum tardiness*, Research Report 43, Management Sciences Research Project, UCLA.

[14] 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.

[15] J.P. de Sousa and L.A. Wolsey (1992). A time-indexed formulation of non-preemptive single-machine scheduling problems. *Mathematical Programming 54*, 353–367.