Parallel Machine Scheduling Through Column Generation: Minimax Objective Functions
(Extended Abstract)

J.M. van den Akker, J.A. Hoogeveen, and 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

Abstract. In this paper we consider one of the basic problems in scheduling and project management: scheduling on parallel identical machines. We present 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. Our framework is based on column generation. Although column generation has been successfully applied to many parallel machine scheduling problems with objective functions of the minsum type, the number of applications for minimax objective functions is still small.

We determine a lower bound on the objective function in the following way. We first turn the minimization problem into a decision problem by bounding the outcome value. We then ask ourselves 'Are m machines enough to feasibly accommodate all jobs?'. We formulate this as an integer linear programming problem and 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 or when local search fails, we use a time-indexed integer linear programming formulation to solve the pricing problem.

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 almost always succeeds. 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 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 [2] and [7]), 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 [3] for an overview. One notable exception is due to Brucker and Knust ([4], [5], [6]), who apply column generation to a number of resource constrained 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 et al. [9] to denote scheduling problems. The remainder of this paper is organized as follows. In Sect. 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 Sect. 3 and 4. In Sect. 5, we add release dates and generalized precedence constraints. In Sect. 6, we describe our local search algorithm to solve the pricing problem approximately, and in Sect. 7 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 Sect. 8 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, which are not related to the due dates.

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 [10]; 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?

We 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

$$(\mathbf{P_1})\min \quad \sum_{s \in S} x_s$$

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 (2) with $x_s \geq 0$ for all $s \in S$; we do not need to enforce the upper bound of 1 for x_s, since this follows immediately from (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 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 [11]. 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 et al. [2] 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* on basis of the reduced cost.

Theorem 1. *An intermediate lower bound for the optimal value of LP-relaxation of P_1 is given by $\sum_{j=1}^{n} \lambda_j / \hat{c}$, where $\hat{c}$ is the solution value of the maximization problem (3).*

We can use this lower bound to decide whether m machines are sufficient. If it 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 pricing 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.

5 Release Dates and Precedence Constraints

In this section, we assume that next to the deadlines there are release dates 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. To be mathematically correct, we should replace q_{ij} by q^1_{ij} and q^2_{ij}, if the arc (i, j) occurs in both A^1 and A^2, but to ease notation, we do not make this distinction. We denote the union of A^1, A^2, and A^3 by the multi set A. This leads to the following integer linear programming formulation

$$(\mathbf{P_2}) \min \quad \sum_{s \in S} x_s$$

subject to

$$\begin{aligned} \sum_{s \in S} a_{js} x_s &= 1, \text{ for each } j = 1, \ldots, n, \\ \sum_{s \in S} C_{js} x_s - \sum_{s \in S} C_{is} x_s &\geq q_{ij} \text{ for each } (i, j) \in A^1; \\ \sum_{s \in S} C_{js} x_s - \sum_{s \in S} C_{is} x_s &\leq q_{ij} \text{ for each } (i, j) \in A^2; \\ \sum_{s \in S} C_{js} x_s - \sum_{s \in S} C_{is} x_s &= q_{ij} \text{ for each } (i, j) \in A^3; \\ x_s &\in \{0, 1\}, \text{ for each } s \in S. \end{aligned}$$

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

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

over all machine schedules $s \in S$. We solve this problem approximately using local search (see Sect. 6). Again, we can compute an intermediate lower bound.

Theorem 2. *An intermediate lower bound for the optimal value of LP-relaxation of P_2 is given by $\left[\sum_{j=1}^{n} \lambda_j + \sum_{(j,k) \in A} \delta_{jk} q_{jk}\right] / \hat{c}$, where $\hat{c}$ is the outcome value of the maximization problem (4).*

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 with a branching strategy based on splitting the execution intervals. 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 Sect. 8).

6 Generating New Columns by Local Search

In this section, we describe the local search algorithm that we have implemented to solve the pricing problem.
Recall from (4) that solving the original pricing problem is equivalent to finding the single-machine schedule that obeys the release dates and deadlines and maximizes

$$\sum_{j=1}^{n} \lambda_j a_{js} + \sum_{j=1}^{n} Q_j C_{js}, \tag{5}$$

where $Q_j = \sum_{h \in P_j} \delta_{hj} - \sum_{k \in S_j} \delta_{jk}$. 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, which can be done in linear time using a shifting procedure, which resembles the procedure for a similar problem given by Garey et al. [8].

We now describe the first step in 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 algorithm 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.

We choose to apply simulated annealing. In our computational experiments, we added up to 50 columns with negative reduced cost per iteration.

7 Time Indexed Formulation

Finally we describe how 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 (see for instance [12] and [1]) and solve the LP-relaxation, which gives an upper bound. We checked the intermediate lower bound by computing this upper bound on $\hat{c}$ every 50 iterations, or when our local search algorithm could not find any column with negative reduced cost. If infeasibility could not be decided and if we could not find a column with negative reduced cost, we turn to the time-indexed ILP formulation of the pricing problem. We compute for which value of the objective function of the pricing problem 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.

8 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 7.

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 solving this time-indexed ILP formulation through CPLEX, 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 when solving the time-indexed formulation; it 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). Hence, the hybrid method first spends some time to find the lower bound and passes this information to the time-indexed formulation.

Results

In our experiments we compare our hybrid algorithm to the direct ILP solved by CPLEX (that is, without knowing the value of the lower bound). We have applied both algorithms on 13 scenarios; for each scenario we ran five test instances. The scenarios 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 scenarios are used to compare our hybrid algorithm to solving the time-indexed formulation directly, without knowing the lower bound. Hence, we are testing whether spending time on determining the lower bound is worthwhile. The scenarios 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 applying CPLEX (Version 9.0) to the ILP formulation appear in the same row between brackets, 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'). 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

Table 1. Test scenarios

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 2. Results of comparing the direct time-indexed approach and the hybrid algorithm

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

find out whether the intermediate lower bound could decide the problem already, and whenever the local search algorithm could not find an improving column.

We also tested the performance of our local search algorithm on the pricing problem by comparing it to the method of only generating columns found by the ILP formulation of the pricing problem. For the scenarios 0, 5, 11 and 12 we ran another set of 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, and Ii denotes the results obtained on scenario i by the algorithm in which the pricing algorithm is solved by the ILP. Scenario 0 is used to show the difference for *easy* instances, where scenario 5 is used to investigate *difficult* instances. Finally scenarios 11 and 12 are used to investigate the influence of the doubling of time values on the results.

Table 3. Results of comparing LS to only ILP solving

	# 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

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 ignorant time-indexed 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 algorithm of solving the time-indexed formulation without specifying the lower bound 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 ignorant time-indexed 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 to solve the pricing problem quadruples the average time for 4 instances and is not able to compute a lower bound for one instance within 30 minutes.

9 Conclusion and Future Research

We have described a column generation approach for the parallel machine scheduling problem with minimax objective subject to release dates and precedence constraints. We have tested the algorithm for maximum lateness subject to release dates, but with greater than or equal precedence constraints only. We suspect that problems with maximum cost are harder, 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. 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. 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.
5. 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.
6. P. Brucker and S. Knust (2003). Lower bounds for resource-constrained project scheduling problems. *European Journal of Operational Research 149*, 302–313.
7. Z.L. Chen and W.B. Powell (1999). Solving parallel machine scheduling problems by column generation. *INFORMS Journal on Computing 11*, 78–94.
8. 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.
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. J.R. Jackson (1955). *Scheduling a production line to minimize maximum tardiness*, Research Report 43, Management Sciences Research Project, UCLA.
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. 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.