Utrecht University

Master Thesis

Robust parallel machine scheduling with relations between jobs

Author:
Dirk-Jan Hoppenbrouwer

Supervisor:
dr. ir. Marjan v.d. Akker
Utrecht University

October 31, 2011
ica-0301191

Abstract

In this work we look at the well known problem of parallel machine scheduling and insert a notion of robustness. Throughout the paper we will look at robustness as an a-priori property of the generated schedule with an objective of minimizing delay propagation between machines in the case of disruptions. This is achieved by introducing a measure for possible delay propagating relations. The minimization of this measure will be combined with known scheduling objectives such as $C_{\max}$ and $\sum w_j C_j$.

For the resulting problems models using Column Generation, Constraint Optimization and Machine- and Time-indexed ILP will be given and compared.

Contents

1 Introduction

A lot of scheduling problems ask to allocate resources to tasks in one form or another. One well known class of these scheduling problems is that of parallel machine scheduling, also denoted $P||f$ by Graham [6]. It asks to allocate a set of jobs to P identical machines such that each job is executed once and by exactly one machine. Furthermore a machine can only process on job at a time. The start time of jobs and the way they are assigned to machines is determined by some objective function f and possibly by additional constraints.

One common property of most types of parallel machine scheduling problems is that they are difficult. If we take the objective function $C_{\max}$ which asks to minimize the maximum completion time over all jobs, we get $2||C_{\max}$ that is already NP-complete in the ordinary sense [10]. The more general $P||C_{\max}$ is NP-complete in the strong sense [5]. Most exceptions of these hardness results are caused by the introduction of fixed processing times and precedence relations with a defined structure such as in- and out-trees or chains. For a large overview on known hardness results see the on-line overview of Brucker [2].

Related problems that will be treated in this report are parallel machine scheduling problems of the type $P|\text{prec}|f$ and variants of it. Now precedence relations can be given between two jobs, forcing one job to be finished before another may start, etc ... For the parallel machine scheduling problem without precedence constraints and the problem with these constraints the number of practical applications is large and hence finding good solutions is important. A good amount of work in this field has been done, see for example the review by Cheng and Sin [13].

Another interesting addition to the parallel machine scheduling problem is that of uncertainty. Scheduling jobs is all fine but can we be sure that the generated schedule be executed as planned? The answer is no, often in real world situations things can and will go wrong. Research in this direction has been done, mostly problem specific. See for example work of Herroelen and Leus [7] for a survey on robustness with project scheduling.

In this work we will define a notion of robustness with respect to delay propagation and treat it as an a-priori property of a schedule. In Section 2 we will define a more formal notion of delay propagation and state an objective function for this. Also some examples of combinations of classic scheduling objectives such as $C_{\max}$ with the delay propagation objective will be given. Two combinations will be used as running examples throughout the rest of the work. Section 3 describes several models for both examples and combinations of models leading to several algorithms. We extend the TIF model by v.d. Akker et.al. [14] to include robustness and compare this approach with a Column Generation approach where we combine generated columns into a feasible schedule. The setup and results of experiments will be given in Section 4, conclusions and recommendations for further work will be given in Section 5.

2 Robustness and Parallel Machine Scheduling

2.1 Parallel Machine Scheduling

For consistency and clarity some definitions are given here and used throughout the rest of the work. A set of jobs is denoted J and a specific job is one of $J_1, J_2 \dots J_n$. The number of machines is denoted m. Each job has several (possibly optional) properties. The mandatory property is processing time p_j, optional properties are a deadline $\bar{d}_j$, release date r_j and weight w_j. A schedule consists of a set of starting times $S_1 \dots S_n$. Furthermore the completion time of a job is then $C_j = S_j + p_j$ and each job has a machine assignment $m(J_i) \in 1 \dots m$

Two jobs can have a precedence relation, for this we use the definition given by van den Akker et.al. [14] with the difference that precedence relations will be defined by comparing start times. Let A^1 be the arc set containing all pairs (J_i, J_j) such that $S_j - S_i \geq q_{ij}$. Similarly let A^2 contain all pairs for which $S_j - S_i \leq q_{ij}$ must hold and A^3 for all pairs with the constraint $S_j - S_i = q_{ij}$. The union of A^1, A^2 and A^3 form the multi-set A and the pair (J, A) form a directed acyclic graph G.

2.2 Delay propagation

The goal of this work is to device a measure on robustness as a property of a parallel machine schedule. We look more specifically into minimization of the number of possible delay propagation points in a schedule.

When a schedule is executed and a delay occurs at a certain machine the job currently executed on that machine, or next in line to be executed, will be delayed. This is cannot be prevented. Also the jobs that are scheduled for a later time on that machine will delay unless there is enough idle time. However most scheduling objective do not allow for much idle time. What is preventable is delay propagation through precedence constraints.

For example consider jobs J_i and J_j, assigned to different machines and the constraint that J_j must start exactly p_i time after J_i starts. Whenever J_i delays so will J_j if we want to keep the precedence constraint intact and this will cause a disruption on the second machine. The more precedence constraints we have spanning multiple machines the more possibilities for delay propagation there are. The first idea would be to minimize the number those machine-spanning precedence constraints leading to a graph partitioning type of objective. The problem is more subtle however.

The drawback of a graph partitioning objective is that it counts all precedence relations spanning two machines. But if we look at delay propagation this does not always make sense. For example consider three jobs J_1, J_2, J_3 and precedence constraints $(J_1, J_2), (J_1, J_3)$ both stating the second job must start at least p_1 time after the start of J_1. Two possible machine and time assignments are present in Figure 1. If we count the number of cuts according to graph partitioning both have 2. But the assignment in the first case implies a new precedence relation (J_2, J_3). Precedence relations are transitive relations, that is if aRb and bRc then aRc with relation R and elements a, b, c. Because of this transitivity the relation (J_1, J_2) becomes redundant and can be forgotten without losing any information. Doing this does lead to only 1 cut. If we consider delay propagation from machine a to machine b as an event that either happens or does not happen, given a disruption somewhere, this makes sense too.

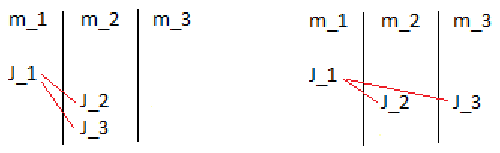

Figure 1: Different number of delay propagation option with equal number of graph partitioning cuts.

Therefore we introduce another measure to count the number of propagation opportunities. Given a job that is a predecessor in one or more precedence relations, we count the number of machines containing a successor job of that job. For any given machine $k : 1 \ldots m$ and job $i : 1 \ldots n$ we define γ_k^i as the possibility of a delay propagation towards machine k when job i delays. Hence we want γ_k^i to be 1 if and only if job i is assigned to a machine other than k and it has at least one successor assigned to k. The constraint expressing this for a given k is as follows, where a_{ik} is 1 if job i is present on machine k and 0 otherwise:

$$\gamma_k^i + a_{ik} - a_{jk} \geq 0 \text{ for each } j \in J, \text{ for each } i \in \text{Pred}(j)$$

Since the objective is to minimize this γ we can setup a truth table:

a_{ik}	a_{jk}	γ_k^i
0	0	0
0	1	1
1	0	0
1	1	0

Hence if we have a relation (J_i, J_j) and J_j is assigned to machine k while J_i is not, then γ_k^i is 1. Thus job J_i has a successor on machine k and thus a delay of J_i may propagate to machine.

Given this measure on delay propagating precedence relations one can ask if it applies to all types of precedence relations. For sure it holds for $J_j - J_i \geq q_{ij}$, even though $q_{ij} < p_i$ ensures that the jobs will be on different machines. The same holds for $J_i - J_i = q_{ij}$. For the last precedence relation, $J_i - J_j \leq q_{ij}$, things change a bit. Whenever J_i delays the constraint is always met and no delay propagation occurs. Things are different if J_j delays. Then J_i can be forced to a later start in order to obey the constraint. Rewriting to $J_i - J_j \leq q_{ij} \equiv J_i - J_j \geq -q_{ij}$ lets this precedence relation work with the above described measure.

The objective of minimizing the number of delay propagating precedence relations is then

$$\min \sum_{i \in J} \sum_{k=1}^{m} \gamma_k^i$$

As a shorthand $\sum \gamma$ is also used throughout this work.

2.3 Combined Objectives

With this measure on possible delay propagating relations we can define combinations of this measure with another scheduling objective. The main two options to combine two objective functions f_1, f_2 is by hierarchical ordering and in a composite function, see for example Hoogeveen [8]. In the case of hierarchical ordering the first objective is solved to optimality. Then for all solutions with this objective value the best one is selected according to the second objective. This is denoted as $Lex(f_1, \sum \gamma)$, where objective function f_1 takes priority over the minimization of delay propagation.

In the composite function case both f_1 and f_2 are combined into one new objective functions. This can be an arbitrary function but in we model it with a linear combination: $\alpha f_1 + \beta \sum \gamma$.

The main two options are combination with hierarchical ordering and in a composite function. With the first option we solve the problem for the first objective and use the solution value as a hard constraint for solving the second objective. An example used in the experiments is $P|\text{prec}|\text{Lex}(C_{\max}, \sum \gamma)$. For this problem we find the best value for $C_{\max}$ and having substituted the deadlines with this value we solve the minimization of $\sum \gamma$.

The second combination option is to couple both objectives into one, possibly giving weights to each. For example $P|\text{prec}|\alpha \sum w_j C_j + \beta \sum \gamma$.

3 Models

This section describes the models used for solving the Lex($C_{\max}, \gamma$) problem. Pointers will be given for an adaptation towards $\alpha \sum_{j \in J} \sum_{k:1\ldots m} \gamma_k^j + \beta \sum_{j \in J} w_j C_j$.

3.1 Machine and Time Indexed ILP

The first model explored is a machine- and time-indexed ILP (MTIF). This is an adaptation of the time indexed ILP given by v.d. Akker [14]. It will be given for $P|\text{prec}, C^*_{\max}|\gamma$, the problem that arises when the higher priority problem $C_{\max}$ has a known result. This $C_{\max}$ can be found for example with Column Generation and ILP [14, 3], in our case we use a Constraint Optimization formulation.

$$\min \sum_{i=1}^{n} \sum_{k=1}^{n} \gamma_k^i \tag{1}$$

s.t.

$$\sum_{k=1}^{m} \sum_{t=r_j}^{\bar{d}_j - p_j} x_{jtk} = 1 \qquad \text{for each } j : 1 \ldots n \tag{2}$$

$$\sum_{k=1}^{m} \sum_{j=1}^{n} \sum_{t=\max 0, \tau - p_j + 1}^{\tau} x_{jtk} \leq m \qquad \text{for each } \tau : 0 \ldots T-1 \tag{3}$$

$$\sum_{k=1}^{m} \sum_{t=r_j}^{\bar{d}_j - p_j} t * x_{jtk} + p_j \leq C_{\max} \qquad \text{for each } j : 1 \ldots n \tag{4}$$

$$\sum_{k=1}^{m} \Big(\sum_{t=r_j}^{\bar{d}_j - p_j} t * x_{jtk} - \sum_{t=r_i}^{\bar{d}_i - p_i} t * x_{itk} \Big) \geq q_{ij} \qquad \text{for each } (i,j) \in A^1 \tag{5}$$

$$\gamma_k^i + \sum_{t=r_i}^{\bar{d}_i - p_i} x_{itk} - \sum_{t=r_j}^{\bar{d}_j - p_j} x_{jtk} \geq 0 \qquad \text{for each } i \in J, j \in Succ(i), \text{for each } k : 1 \ldots m \tag{6}$$

$$x_{jtk} \in \{0,1\}, \gamma_k^i \in \{0,1\} \tag{7}$$

Whereas the time-indexed ILP is sufficient for $C_{\max}$ and $L_{\max}$ problems it will not work for our secondary objective because we need a machine assignment as well. Hence a machine index is added to the decision variable giving x_{jtk}, which is 1 if job j starts at time t on machine k. Constraint 4 is a hard constraint on the maximum allowable $C_{\max}$. Constraints $2, 3, 5$ demand that each job is executed once, no more than m jobs are processed at any given time and the precedence relations are met respectively (only A^1 is given). Constraint 6 keeps track of the number of possible delay propagations. Changing this model to accommodate linear combinations of objectives is straightforward. In constraint 4 the bound $C_{\max}$ is changed into a variable (e.g. $C_{\max}$ itself or C_j) and added to the objective function.

A major drawback of this machine- and time-indexed ILP is the symmetry introduced by the machine index. This leads to an ILP where one solution can be changed by switching the machines of sets of jobs leading to new solutions while they are actually the same. This is detrimental to the performance of an ILP. However this MTIF model will be used later in combination with other models to decrease the influence of this phenomenon.

3.2 Constraint Optimization

Another simple model is created by formulating the problem as a Constraint Optimization Problem (COP). The objectives and constraints are in principle the same as in the MTIF but since it is implemented in the IBM/ILOG CP solver we can simplify the mathematical formulations a bit by using the build-in composite variables and constraints. The used model is derived from an example model given by IBM/ILOG in the documentation of Cplex Studio 12.2.

In the CP model we have three types of variables, the *interval variable*, *integer variable* and *interval sequence variable* The interval variable is a composition of several variables that form an interval. An interval variable v_i can be optional and thus contains the boolean property *isPresent*. Furthermore the interval variable contains three more variables for start, end and duration. All three have a bounded domain of integers. The integer variable is equal to that in (I)LP models, a variable over a (possibly bounded) domain of integers. The interval sequence variable represents a total ordering on all present interval variables in a set.

We use two sets of interval variables: $T \leftarrow \{v_1 \dots v_n\}$ and $D \leftarrow \{u_{n\times m}\}$. For each machine $k : 1 \dots m$ we have an interval sequence variable $S \leftarrow \{s_1 \dots s_m\}$. Each interval sequence variable is defined over the set of interval variables $\{u_{n\times m}\}$. In order to count the number of delay propagating precedence relations we use the same construct as in the ILP model, the integer variables γ_k^i.

Set T is used to determine the start of each job and thus each interval variable in T is mandatory. The set D determines the machine assignment of each job and therefore all variables in D are optional variables.

To make everything work several constraints are added to the problem. First variables from sets T and D need to be linked together. This is done by the *alternative* constraint. This constraint states that v_i is present if and only if exactly one of $u_{i\times m}$ is present with the same start and duration.

$$\textit{alternative}(v_i, u_{i\times m}), \forall i : 1 \dots n$$

To ensure each machine processes only one job at a time we use the constraint *nonOverlap* over the interval sequence variable sets in S. The constraints ensure that no pair of interval variables in the sequence overlap.

$$\mathit{nonOverlap}(s_k), k : 1 \ldots m$$

The precedence relations are easily modeled by the specialized precedence constraints.

$$\mathit{startBeforeStart}(v_i, v_j, q_{ij}), \forall (i, j) \in A^1$$

$$\mathit{startAtStart}(v_i, v_j, q_{ij}), \forall (i, j) \in A^3$$

The γ variables are constrained in a way equal to the ILP model, replacing the ILP decision variable x with the *isPresent* property of the interval variables in D.

This model can be used to determine a result with a minimum number of possible delay propagating relations or to (quickly) provide a non-optimal solution which can be used as a start solution for other models. Without the count on possible delay propagations this model is also used to determine an optimal $C_{\max}$ directly.

3.3 Column Generation

In order to cope with the symmetry introduced by the machine-index in the ILP formulation we tried a Column Generation approach with the intention to combine generated columns into a feasible (and hopefully good) schedule.

For Column Generation we formulate the problem as a selection problem over all feasible single machine schedules S. We then have to select m schedules in S such that we get the best, and feasible, solution to the problem. This is formulated in the Restricted Master Problem (RMP). Because the set S is very large for all practical problems we start with a small and feasible initial set. Solving an LP-relaxation of the RMP gives a set of dual variables, one for each constraint. With these duals the pricing problem is solved, a function that returns a column with lowest possible reduced cost. If this reduced cost is negative the column will improve the RMP when added. After addition the RMP is solved again,... Until no improving columns can be found. Then the solution to the LP-relaxation of the RMP is optimal and is a lower bound for the integral variant of the selection problem.

3.3.1 Restricted Master and Pricing Problems

The Master Problem (MP) is rather straightforward and given below.

$$\min \sum_{s \in S} c_s x_s \tag{8}$$

s.t.

$$\sum_{s \in S} a_{js} x_s = 1 \qquad \text{for each } j \in J \tag{9}$$

$$\sum_{s \in S} x_s \leq m \tag{10}$$

$$\sum_{s \in S} S_{js} x_s - \sum_{s \in S} S_{is} x_s \geq q_{ij} \qquad \text{for each } (i,j) \in A^1 \tag{11}$$

$$x_s \in \{0,1\} \tag{12}$$

Constraint (9) states that each job must be executed exactly once. Another constraint that is often seen in these types of problems is the cover constraint. Then the demand is that each job is executed at least once. There are benefits to using covering constraints, see for example Barnhart [1]. For parallel machine scheduling and VRP type of problems removing one of any double assigned jobs or locations results in a feasible new schedule that is at least as good. Hence an optimal solution for the covering formulation is as good as an optimal solution for the partitioning formulation. In our case the measure of possible delay propagating relations does not lead to sub-columns with cost at most that of the original column. See the example with schedules s and $s' \subset s$ below, with precedence relation (i,j). Define $\gamma^i_{s'}$ as the column generation counterpart of γ^i_k. Because we do not have jobs explicitly assigned to machines now we cannot count possible delay propagations from job i to machine k. But we can count possible delay propagations towards single machine schedule s'. Replacing a_{ik} with $a_{is'}$ and γ^i_k with $\gamma^i_{s'}$ we have the same mechanic with the same measure. For s' we have $\gamma^i_{s'} = 1$ which is not the case with s.

$$s = \begin{pmatrix} i \\ \vdots \\ j \end{pmatrix} s' = \begin{pmatrix} \vdots \\ j \end{pmatrix}$$

However the cover constraint is only a good replacement when we can remove jobs that are executed more than once from a column without changing the objective value. In our problem this is not the case since removal of a job from a column can increase the objective value if it had a successor in that column.

Constraint (10) limits the number of single machines schedule that are combined to the number of machines available. The 'greater-equal' precedence relation is given by constraint (11), the other types follow in the same way.

We chose to not define the cost globally in the Restricted Master Problem since the previously described measure on propagation delay works fine on a per column basis. Let the constraints of the RMP have dual variables $\lambda_j, \pi, \delta_{ij}$. As is given by v.d. Akker [14] we can aggregate the δ_{ij} duals into values for specific jobs in the following way. Let Q_j be

$$\sum_{h \in \text{Prec}(j)} \delta_{hj} - \sum_{k \in \text{Succ}(j)} \delta_{jk}$$

.

With these duals a pricing problem can be formulated.

$$\min \sum_{i \in J} \gamma^i_{s'} - \sum_j a_{js'} \lambda_j - \sum_j Q_j S_{js'} - \pi \tag{13}$$

s.t.

$$\gamma^i_{s'} + a_{is'} - a_{js'} \geq 0 \qquad \text{for each } j \in J, \text{for each } i \in \text{Pred}(j) \tag{14}$$

$$S_{js'} a_{js'} \leq C^*_{\max} \qquad \text{for each} j \in J \tag{15}$$

$$\sum_{j \in J : S_{js'} \in \{t - p_j + 1 \ldots t\}} a_{js'} \leq 1 \qquad \text{for each } t \in \{0, \ldots C^*_{\max}\} \tag{16}$$

$$a_{js'}, \gamma^i_{s'} \in \{0, 1\}$$
$$S_{js'} \in \{0, \ldots C^*_{\max}\}$$

The first constraint of the pricing problem defines the cost structure of the problem. It forces $\gamma^i_{s'}$ to be 1 whenever there is a precedence relation (i, j) with $a_{js'} = 1$ and $a_{is'} = 0$. Hence every job that is a predecessor in a relation, not present in s' and has as at least one successor in s' counts as 1 possibility for delay propagation. The second and third constraint enforce that the given $C_{\max}$ is obeyed and that at any given time slice t there is at most one job executed.

These RMP and pricing problems can again be easily adapted for other objectives. In the case of $\alpha\gamma + \beta \sum w_j C_j$ the RMP remains unchanged. The pricing problem is extended to include the weighted sum cost by adding the variables C_j for each job $j \in J$. In order to have a meaningful value for C_j we replace constraint (15) with:

$$a_{js'} S_{js'} + p_j \leq C_j, \forall j \in J$$

This guarantees that C_j is at least the completion time of job j or the processing time p_j if the job is not present. Since C_j will be minimized it will also be at most the completion time or the processing time if $a_{js'} = 0$. The new objective function of the pricing problem becomes:

$$\sum_{i \in J} \gamma^i_{s'} + \sum_{j \in J} a_{js'} w_j C_j - \sum_j a_{js'} \lambda_j - \sum_j Q_j S_{js'} - \pi$$

3.3.2 Simulated Annealing Pricing Problem

The pricing problem itself is already a difficult problem. Hence we implemented it in three different ways to differentiate between exact and heuristic, and slow and fast.

The first pricing implementation is a two phase Simulated Annealing procedure as described by v.d. Akker et.al. [14], see also Kirkpatrick [9] for an introduction to SA in general. The procedure starts out with an initial empty column and employs two phases in each iteration. The first phase is a standard SA step where a change of the current solution is chosen from a set of possible changes. The first phase selects a subset of jobs and puts them in an ordering. This set of selected jobs is changed iteratively by one of the following moves:

- Add a job at a random position in the column
- Remove a job from the column
- Swap the position of two jobs in the column
- Swap a job in the column with a unselected job

After this step the second phase is executed. In this phase the exact start times are determined. The best start time of a job is influenced by the term $Q_j S_{js'}$, if $Q_j > 0$ it is beneficial to start it as late as possible and the reverse holds if $Q_j < 0$.

1. Set the start time of selected jobs as early as possible.
2. Starting from the last job to the first, if $Q_j > 0$ the start time $S_{js'}$ is increased until either
    - $C_j = \bar{d}_j$
    - $C_j = S_{ks'}$ where job k is the next selected job in the ordering. If this happens jobs j and k are joined together to form a new job J_{new} with $Q_{\text{new}} = Q_j + Q_k$ and deadline $\bar{d_{\text{new}}} = \min(\bar{d}_j, \bar{d}_k)$. If $Q_{\text{new}} > 0$ we repeat the process on this new job, otherwise we look at the previous job in the ordering.

Also during this phase the reduced costs are calculated. As usual with SA changes are always accepted if they improve the solution and accepted with a probability determined by the change in the objective value and a cooling schedule if non-improving. The chance of acceptance is $e^{(f(x)-f(y))/T}$ where $f(x)$ is the objective value before the change and $f(y)$ that after the change. Variable T is the temperature of the cooling schedule. Initially this T has a value high enough to frequently accept changes that increase the objective value and is lowered during the process. This temperature allows the SA procedure to climb out of local optima during the search but decreases the chance of worsening moves in order to converge to a solution. The decrease of T is accomplished by $T \leftarrow T * \alpha$ after every Q accepted changes. The parameters α, T, Q are chosen in such a way that the runtime of the procedure is limited but the results are acceptable. In the Experiments Section we will look into these values.

During the search a lot of solutions are encountered since each move results in a new, although not necessarily unique, new solution. Instead of only returning the best solution we keep a set of solutions. In order to do this we maintain a ordered tree data structure with the ordering determined by the reduced cost of a solution. Each time a new solution is accepted by the SA procedure we try to add the solution to the tree set. This way we can guarantee uniqueness of the solutions, although it is possible that a new unique solution is discarded because it has a reduced cost equal to that of a previous solution. The big upside is that checking whether a new solution is unique only costs $O(\log n)$ time in the number of solutions and the solutions are already sorted by reduced cost. The number of unique solutions found can be very large thus at the end of the procedure a subset of all columns is returned containing the x best columns. The size of x is dependent on whether the Column Generation is used for a lower bound or to combine columns in the IP-CG.

3.3.3 Cplex and CP pricing

Because the Simulated Annealing procedure is a heuristic there is no guarantee that the solution returned is optimal. And therefore we cannot conclude that the current set of columns in the RMP is optimal by only working with the SA pricing. To counter this the pricing problem is also implemented as an ILP problem. This is slower than Simulated Annealing but will give guarantees about optimality. Like the Simulated Annealing the ILP will find intermediate solutions most of the time and these are always returned so that they can be added to the RMP. We do not limit the number of solutions returned because in general the number is rather low.

The third implementation of the pricing problem is a CP formulation. Just as with the complete problem the ILP and CP variants have distinct properties. The CP finds reasonable good solutions fast, the ILP is better in proving optimality and will find better columns if the calculation time is longer.

3.3.4 Column Deletion and Stability

Two measures implemented to try to speed up the Column Generation procedure are column deletion and stability.

With column deletion we try to keep the number of columns in the RMP low. To do this the outcome of the RMP is checked every 20 iterations. For all non-basic column the reduced cost are calculated and when those reduced cost are higher than a certain threshold, say 0.1, the column is deleted from the RMP. Preliminary testing shows that while this strategy decreases the number of columns in the RMP it also causes some side effects. The first is that the procedure itself takes time: for all columns we must check if it is basic or not. And if not new reduced costs must be calculated based on the dual variable values of that iteration. Furthermore the deletion of columns seems to increase the tailing-off effect of the Column Generation. That is with column deletion the Column Generation requires more iterations than without it. Since the pricing problem is difficult most of the time is spent solving pricing problems. Hence more iterations is likely to be worse than a larger RMP in terms of efficiency. In the end there is no positive effect (or even a negative effect) from the deletion of non-basic columns.

Another explored addition to the Column Generation procedure is that of stabilization. In general several issues arise with Column Generation. One of them is instability of the dual variables while they convergence to a set of optimal values. Each iteration when columns are added (and thus rows in the dual problem) the estimate of a set of optimal dual values gets better. However there is little guarantee on the way this convergence happens. Even more so early on in the Column Generation the estimate of dual variables can be way off, leading to columns with positive reduced cost for an iteration, but no meaning in the final set of columns. These problems and possible solutions have been observed several times, see Lübbecke et.al. [11] for an overview. We chose to implement the box penalty method by du Merle et.al. [4]. The idea is to define a box around an estimate of the optimal dual variables values. Inside this box dual variables can take any value. If the dual variable gets a value outside of this box a penalty must be paid. With this strategy a more stable convergence of the dual variables is ensured. This box is easily defined by extending the Primal problem (P) to a new problem (P').

$$\begin{aligned} (P)\min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

$$\begin{aligned} (P')\min \quad & c^T x - \delta_- y_- + \delta_+ y_+ \\ \text{s.t.} \quad & Ax - y_- + y_+ = b \\ & y_- \leq \epsilon_- \\ & y_+ \leq \epsilon_+ \\ & x, y_-, y_+ \geq 0 \end{aligned}$$

In the new Primal problem slack and surplus variables are added to each constraint. Their values are bounded by some ϵ and their use has a cost δ. In the Dual problem we already had a dual variable for each constraint in the Primal, but now they are bounded by δ_-, δ_+. Furthermore there are additional dual variables w_-, w_+ for the slack and surplus binding constraints of the Primal problem. These w variables are slack and surplus variables in the Dual problem for the original dual variables and their use is penalized by ϵ_-, ϵ_+.

Thus in the dual problem the variables are boxed in by $[\delta_-, \delta+]$. If the variable takes a value outside this box either ϵ_- or ϵ_+ must be paid for each unit outside the box. This method does have some difficulties. At each iteration suitable values for δ_-, δ_+ must be determined. The same holds for ϵ_-, ϵ_+. A suggested approach by du Merle [4] is to set the boundaries of the box equal to the dual estimate of the previous iteration, possibly increasing the boundaries with some ξ to both sides. The values of ϵ can be decreased every few iterations.

These update rules are implemented. Every iterations we defined δ_-, δ_+ to be the dual estimate of the previous iteration. Furthermore every 20 iterations ϵ_-, ϵ_+ was decreased by multiplying it with some $0 \leq \alpha_-, \alpha_+ \leq 1$. In experimenting to find good values for the initial ϵ and α we did not find any suitable values that resulted in better run times. The question why remains. It is possible that the update rules used are not very suitable for this type of problem. There are several other options to stabilize the dual variables, see for example Rousseau et.al. [12] for another stabilization method. An advantage of the interior point stabilization by Rousseau is that it is not dependent on any update rules. Another possibility is that we just failed to find good values for the initial ϵ and α. At any rate the stabilization was discarded from further investigation.

3.3.5 IPCG and Diversity

When the Column Generation procedure is finished we have a lower bound on the number of possible delay propagating relations and a set of single machine schedules. The set of columns can be used by making an integer problem of the RMP again in order to select m columns forming a feasible and good solution. This was our first idea when designing the Column Generation procedure.

However there are some issues with this idea. It appears to be fairly difficult to end the Column Generation with a set of columns that contains a feasible subset (other than the initial solution). Some small testing on generated columns show that often times columns share at least one job. A probable reason for this is that certain jobs are very beneficial to take in a column. This may be caused by the structure of the precedence relations. For example a small strongly connected component in the precedence graph may be included in a column entirely or not at all because cutting it up leads to increased reduced cost based on the cut measure. To improve chances of starting the IP step with such a set we introduced several methods to increase diversity among the columns. For this end we use two types of column: RMP columns and diversity columns. The RMP columns are columns that are added to the master problem during the column generation procedure. Diversity columns are generated only to aid the IP step and are stored during the column generation in order to decrease the time it takes to solve the LP-relaxation of the RMP.

The first step towards more diversity is to ramp up the number of solutions returned by the Simulated Annealing pricing. During the search the SA may encounter a lot of unique solutions so we raise the number of columns returned by the procedure to be the minimum of the number of encountered solutions or 100. Adding all these columns to the RMP is infeasible because we need to solve the RMP each iteration. Therefore we only add a few of the best columns to the RMP and keep the rest in the pool with diversity columns. Because we only need unique columns both column collections are implemented as ordered trees based on a lexicographic ordering of the column. This allows for cheap insertion of new columns while maintaining uniqueness.

The second method is to generate additional columns for the sake of diversity. This option is explored by Diepen [3] for the gate assignment problem. Each column added to the Restricted Master Problem is also used as a mask to generate more columns. All jobs included in the added column are forbidden and we price again. For this pricing we do not require the reduced cost to be negative, only as low as possible. Also this step will be used very frequently. Because of these two properties the CP implementation of the pricing problems looks very suitable. It generates reasonable solutions very fast without guarantee. It will however find only a small number of solutions. Hece we use a combination of the SA and CP pricing. For a small number of job masks SA pricings is done, usually 5, and the rest of is done by CP. This way we do have a large number of columns due to the SA pricing and considered all masks due to the CP pricing. All the columns generated in this step are not added to the RMP but stored to be added later. For one they do not need to be improving columns and also the large number of columns would severely slow down solving the RMP.

The last procedure for additional diversity is a sort of 1-depth branching when the column generation is completed. All columns that have a variable value of at least 0.05 are used for this step. One by one we force each of these variables to have a value of 1, solve the LP-relaxation of the RMP for fresh dual variables and we price with the Simulated Annealing. Also the jobs in the forced column are forbidden for the pricing to use. All the resulting columns are added to the RMP and the variable is freed again. This procedure adds a large set of new column to the RMP and each of these is compatible with at least one other column that is good in the LP-relaxation.

After the Column Generation and 1-depth branching are completed all the stored columns are added to the RMP and the problem is made integral again. As can be seen in the experimental results the total number of columns generated can be large, in the order of tens of thousands.

3.3.6 Using the Column Generation with machine- and time-indexed formulation

Another use of the Column Generation procedure is to have it as a preprocessing step for the MTIF. This is the option used by v.d. Akker et.al. [14], although their formulations are a bit different.

The Column Generation procedure is followed as described above without the creation of additional columns for diversity. When the lower bound is found we pass it on to the MTIF formulation where it is added as a constraint on the objective. This lower bound then greatly improves the ability of the MTIF formulation to prove optimality of a solution. In addition the problem instance itself is tightened. Given the basic columns of the RMP for each job we set the release date to the earliest start time in the set of columns. Also the deadline is set to the latest completion time in the columns. This reduces the number of variables in the MTIF formulation which allows the solver to find solutions more easily.

Type	N	M	$\|A\|$
01	40	4	20
02	70	7	35
03	120	12	60
04	150	15	75
05	80	8	30
06	80	8	40
07	80	8	50
08	90	6	45
09	90	9	45
10	90	12	45

Table 1: Problem types with $p_j = 1$ and $r_j = 0$.

4 Experiments

4.1 Hardware and software

All the experiments are conducted on a AMD Athlon X2 2.6 GHz with 4GB of memory and 64 bit Windows. The Java SE version used is 1.6 in 64-bit mode. For the CP and (I)LP formulations the IBM/ILOG CPLEX Studio 12.2 is used. All code is single-threaded and the IBM/ILOG solvers are given 2 threads to work with.

4.2 Problem types and Solution Strategies

For testing and comparing the different solving strategies we use a set of problem types listed in Table 1. For each type a number of problem instances is generated with randomly determined precedence relations. The first four problem types are chosen to check scalability in terms of the number of jobs and machines. The next two sets of three types are designed to determine scalability in terms of precedence relations and number of machines respectively.

All problem types have jobs with constant processing time. Whether this means $p_j = p$ or $p_j = 1$ is irrelevant since we only use $q_{ij} = p_i$ and do not include release dates and deadlines. Also we limited the problems to only have precedence relations of the type $S_j - S_i \geq q_{ij}$.

4.2.1 Plain strategies

The ILP and CP models are easily implemented and tested. However there are several drawbacks in using both models as is shown in a small test. For the first four problem types 5 instances are generated and solved by the machine- and time-indexed formulation (MTIF), the CP formulation and again by the MTIF with the addition of a start solution provided by CP with limited running time (arbitrarily chosen to be 15 seconds). The maximum computation time is 300 seconds for all three strategies (thus the MTIF may spend $300 - 15$ seconds if it is given an initial solution by CP). The reason that we only look at the first four problem types is because of the results they provide. All three strategies are fail to find optimal values for all but the easiest problems. The results can be found in Table 2. The column *opt* denotes the number of solutions proven optimal divided by the total number of instances.

What we can see is that the CP has the most difficulty of the three with finding the best solutions and proving optimality. When looking into the output logs however there is an interesting distinction between the CP and the MTIF in the sense that the CP does find an initial solution rather quickly where the MTIF may spend a sizable portion of its time in the root node of the branch tree without finding integral solutions. This lead to the idea of the third strategy that combines these two properties. Although the number of experiments is to low for any definitive conclusions the idea to use CP as a generator of initial solutions is used in some of other experiments.

type	MTIF			CP			MTIF+start		
	$\sum\gamma$	t(s)	opt	$\sum\gamma$	t(s)	opt	$\sum\gamma$	t	opt
1	1	98.41	0.8	1	240.367	0.25	1	22.88	1
2	1	244.31	0.2	3	300.19	0	1.4	247.025	0.2
3	1.4	241.86	0.2	23.4	300.25	0	1.6	232.46	0.2
4	9	300	0	35	300	0	7.6	300	0

Table 2: Timings for MTIF, CP and MTIF with initial solution

4.2.2 Simulated Annealing

In order to determine the parameters T, Q, α a small test is conducted. Since Q and α both determine the speed in which T is decreased during the procedure we have chosen to set α to 0.95, T to 8 and variate in Q. The reason to set T to 8 is because the value of T does not have a great impact on the effectiveness of the Simulated Annealing procedure for this problem. This may be caused by the solution space not containing many local optima which are hard to get out of. A higher value for T does mean a longer running time since the stopping criterion of the Simulated Annealing is a value of T which does not allow deteriorating moves. Tables 3 and 4 give results for the Simulated Annealing pricing with different combinations of values for T and Q. For this test an instance is solved with Column Generation. Besides the normal CG procedure additional pricing problems are run on the same set of duals as the pricing in the normal procedure in order to have somewhat realistic pricing problems. Each additional pricing problem is executed with a different set of parameters. The performance of these pricings is measured and their results discarded. In Table 3 we find the number of iterations in which a certain parameter configuration was able to find an improving column. As expected a higher value of Q leads to more pricings being successful.

In order to investigate the influence of Q somewhat more in-depth tests are conducted on instances of problem type 02. For each generated instance

T	Q					
	4	8	12	16	24	32
4	254	288	308	321	349	361
8	250	295	313	320	348	357
12	266	289	311	332	346	366
16	256	297	327	332	347	369
20	251	285	308	325	350	367

Table 3: Total number of SA pricings able to find an improving column in a $N = 100$, $m = 10$, $|A| = 50$ instance.

T	Q					
	4	8	12	16	24	32
4	2.2	3.3	4.9	5.4	7.6	10.8
8	2.7	4.0	5.2	6.2	9.7	11.8
12	3.1	4.4	5.7	6.9	9.4	12.2
16	3.7	4.8	6.2	7.8	10.5	14.2
20	4.0	5.1	7.0	7.8	10.6	13.9

Table 4: Sum of computation time in seconds for SA pricings in a $N = 100$, $m = 10$, $|A| = 50$ instance.

the lower bound column generation is run several times, each with different values for Q. The total number of generated instances is 200. The results are recorded in Table 5. Looking at the results we see that the performance of the lower bound Column Generation is not that much different for the different values of Q. However larger values of Q do result in less deviation in the running times. This is caused by the larger number of successful Simulated Annealing pricing problems and thus the smaller number of ILP pricings. And the deviation in run time for the ILP pricing can be rather large. Hence we will use $T \leftarrow 8, Q \leftarrow 32, \alpha \leftarrow 0.95$ for the Simulated Annealing procedure.

Q	avg CG time(s)	stdev	avg time(s) ILP pricing	stdev
24	9.65	6.51	5.55	6.57
32	9.64	5.55	5.02	5.5
40	10.14	5.55	4.94	5.5

Table 5: Timings for the lower bound Column Generation procedure with different values of Q in the Simulated Annealing pricing.

4.2.3 MTIF with information

Designated as MTIF+ we use the same aforementioned strategies with additional information from the Column Generation. The information we propagate is the lower bound found by the Column Generation and set this as a hard constraint in the two formulations. This does not necessarily help the formulations to find better values faster but it does help to determine that an optimal solution is actually optimal.

Furthermore the release dates and deadlines are updated with use of the results from the Column Generation. We look at the columns with non-zero value in the final iteration. From these columns we determine for each job the earliest start and latest completion. These timings are then set as release dates and deadlines for the two formulations.

The Column Generation procedure is initialized with a solution generated by the CP procedure. The CP is given an arbitrary 5 seconds to find a reasonable initial solution. Furthermore the precedence relations are limited to the type $S_j - S_i \geq q_{ij}$. We recorded the result of the initial CP as an upper bound on $\sum \gamma$, the lower bound is determined the Column Generation. Non-opt denotes the fraction of problems for which no optimal solution was found. In the case of Column Generation this means whether or not the process terminated because no improving column was found by the ILP pricing. In the case of MTIF we require an optimality proof from the solver. The results are listed in Table 6. The lower bound CG is given a maximum of 600 seconds and the MTIF one of 300 seconds.

type	CP	LBCG					MTIF+		
	ub($\sum \gamma$)	lb($\sum \gamma$)	t(s)	non-opt	# ilp	# iter	$\sum \gamma$	t(s)	non-opt
1	0.95	0.4	7.09	0	11.75	105.2	0.4	0.12	0
2	16.5	1.6	44.63	0	54.5	524.25	1.6	0.483	0
3	47.7	2.3	285.83	0.05	143.35	1837.7	2.3	7.15	0
4	63.05	3.2	552.35	0.7	238.7	3723.4	3.3	45.79	0.1
5	15.3	0.5	14.4	0	24.65	272	0.55	15.4	0.05
6	23.95	1.35	53.6	0	56.65	603.4	1.35	0.65	0
7	32.95	4.4	428.56	0.3	286.75	2743.15	4.45	20.68	0.05
8	23.15	1.75	542.53	0.75	566.05	5740.1	1.75	0.94	0
9	30.3	1.6	103.88	0	78.3	941.55	1.6	2.45	0
10	32.6	3.2	25.8	0	38.55	384.05	3.25	17.16	0.05

Table 6: Experimental results for MTIF with information from the column generation.

The first four problem types give a good indication of the scalability in terms of the number of jobs, machines and relations. Type 1 is very easy and even the initial solution generated by the CP formulation is often already optimal. If this is not the case the column generation and MTIF do not need much time to find the best solution. Up to problem type 3 the combination of an initial solution, column generation and the MTIF will usually lead to an optimal result. However the running times increase considerably. Problem type 4 is in general to large to find optimal results. The MTIF formulation can cope with the size but the column generation cannot find the lower bound in 10 minutes in 70% of the instances.

Type	N	M	$\|A\|$
01	10	2	5
02	20	2	10
03	40	4	20
04	60	6	30

Table 7: Problem types with $p_j = 1$ and $r_j = 0$ for IP-CG.

Problem types 5-7 illustrate the influence of the number of precedence relations. The largest influence can be found in the column generation. It greatly benefits from a relative low number of relations. The MTIF formulation is also sensitive to the number of relations. If we leave out the instances that the MTIF cannot prove optimal we end up with average running times of $0.42s$ for type 5 and $5.98s$ for type 7. However this is still fast enough for practical purposes.

The scalability check in terms of the number of machines shows a difference between the column generation and MTIF formulation. When the number of machines is lowered the column generation struggles greatly to find a lower bound. This is as expected since a lower number of machines means more jobs on a machine and this increases the difficulty of the pricing problem. A higher number of machines makes the pricing, and thus the column generation, easier. For the MTIF formulation it works the other way around. More machines lead to more symmetry in the model. Taken both observations into account we see that overall more machines lead to better performance for the tested number of machines.

On all problem types we see a relative large number of ILP pricings. This could mean that a more efficient Simulated Annealing implementation or other heuristic may increase the overall performance of the Column Generation procedure.

4.2.4 IP-CG

As described in the previous section we also implemented a IP Column Generation Scheme. Because of the difficulty in combining columns the problem types used are scaled down.

The experiment is to let the Column Generation run on the instances without an initial solution by the COP formulation. The reason to do this is because the CP may find optimal solutions for these smaller instances and thus obscuring the performance of the (IP-)CG. For problem type 4 only 6 instances are tested, the reason for this being that the IP step almost always times out at 5 minutes. See table 8 for a listing of the results. We see that the results are not good for IP-CG in the sense that the quality of the ILP solutions is low. For problem types 01 and 02 it works reasonable, always finding the optimum for type 01 and in most cases for type 02. However for the larger problems the gap between the optimum and result found with IP-CG quickly increases. For type 04 the IP step usually times out on the large number of columns obscuring the average

Type	CG t(s)	1-depth t(s)	ILP t(s)	$\sum \gamma$	lb($\sum \gamma$)	cols
01	1.97	0.01	0.19	0.25	0.25	878.45
02	23.16	0.06	3.54	0.15	0	13291.1
03	85.6	0.6	165.16	3.4	0.55	40044.1
04(*)	147.87	3.64	272.47	19.83	0.83	65371.33

Table 8: Results for the IP-CG, timings are given for the total time CG takes, the 1-depth branch step that creates additional diversity and the ILP step.

Type	CG t(s)	1-depth t(s)	ILP t(s)	$\sum \gamma$	lb($\sum \gamma$)	cols
01	1.91	0.32	0.27	0	0	852.2
02	18.53	0.1	7.78	0.05	0	11132.15
03	66.54	0.6	258.34	10.7	0	30982.85
04	105.54	3.24	301.13	23.75	0	48789.45

Table 9: Results for the IP-CG with $C_{\max}$ relaxed by 20%

objective value found.

We also tested the IP-CG setup with a relaxed $C_{\max}$ constraint. The idea is that the $C_{\max}$ limit is very tight, making it difficult to have unfavorable jobs in single machine schedules. Hence we relaxed the optimal $C_{\max}$ by 20%. The results can be found in table 9.

Interestingly enough relaxing $C_{\max}$ does not have a positive effect on the IP-CG as a whole. It does make the Column Generation part easier because the lower bound is always 0, however the IP step has a much harder time. This is likely due to the relaxed $C_{\max}$ that expands the space of possible columns while reducing the number of columns generated.

5 Conclusion

We described a way to measure delay propagating precedence relations in a parallel machine scheduling problem. In order to tackle this problem ILP, CP and Column Generation models are presented and several combinations are compared.

Extending the work of v.d.Akker et.al. [14] with the robustness measure has shown some merit. We added an additional index to the time-indexed ILP formulation introducing symmetry in the model. However using information from a Column Generation procedure several problem instances proved solvable. Experimenting with column deletion and stabilization did not provide additional benefits.

The strategy of Column Generation and the combination of generated columns into a feasible (and good) solution does not work well in the presented way. In the case where we used the normal Column Generation no combinations can be found. If the Column Generation is extended to produce more columns we see that combining those columns becomes a bit easier. However the run time also greatly increases and combining columns remains a difficult problem.

The creation of CP implementations of models proved to be useful. While they are not very suitable to find guaranteed optimal results by themselves they can be deployed during many stages of other solution strategies. Providing Column Generation with a good initial schedule can decrease the time it takes to find the lower bound significantly. Furthermore the CP pricing turned out to be useful for creating columns for the sake of diversity.

6 Further work

The difficulty of combining columns can be countered in several ways. The first option is to explore a Branch-and-Price approach. In this approach no diversity columns would be needed. Whenever the procedure finds a non-integral lower bound a branch occurs, for example for a pair of jobs i, j we either forbid and enforce their execution on the same machine. This gives two new problems which can be solved by Column Generation, ... until a integral solution is found. The IP-CG and Branch-and-Price can also be combined by checking whether or not a feasible and good combination can be found after the CG but before branching. Whether or not this will result in a better algorithm is unknown.

Another combination can be created by letting the Column Generation produce partial schedules. When a lower bound is found one can find a partial schedule filling only half of the machines with jobs. This is definitely easier than finding a full schedule and likely requires less diversity steps in the Column Generation procedure. The jobs that are in this partial schedule can then be enforced in a MTIF or CP model asking to complete the schedule. Doing this greatly reduces the number of variables and symmetry in the completion step. However finding a feasible full schedule is no longer guaranteed in the case of Lex($C_{\max}, \sum \gamma$) so the $C_{\max}$ constraint should be made a soft constraint with a penalty. For problems where multiple objectives are combined in a composite function this is not an issue.

For more difficult problems with for example release dates and large processing times it is questionable whether the Column Generation and MTIF combination will still work since the number of variables will grow fast in the MTIF. For these types of problems local search may be an interesting alternative.

References

[1] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H.Vance. Branch-and-price: Column generation for solving huge integer problems. *Operations Research*, 46(3):316–329, 1998.

[2] P. Brucker and S. Knust. Complexity results for scheduling problems, www.informatik.uni-osnabrueck.de/knust/class/, 2011.

[3] G. Diepen. *Column Generation Algorithms for Machine Scheduling and Integrated Airport Planning*. PhD thesis, Utrecht University, 2008.

[4] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1-3):229–237, 1999.

[5] M.R. Garey and D.S. Johnson. "strong" np-completeness results: motivation, examples, and implications. *Journal of the Association for Computing Machinery*, 2011.

[6] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics 5*, pages 287–326, 1979.

[7] W. Herroelen and R. Leus. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research 165*, pages 289–306, 2005.

[8] H. Hoogeveen. Multicriteria scheduling. *European Journal of Operations Research 167*, pages 592–623, 2005.

[9] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 2011.

[10] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1977.

[11] M.E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[12] L. Rousseau, M. Gendreau, and D. Feillet. Interior point stabilization for column generation. *Operations Research Letters*, 35(5):660–668, 2007.

[13] C.C.S. Sin T.C.E. Cheng. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47(3):271–292, 1990.

[14] J.M. van den Akker, J.A. Hoogeveen, and J.W. van Kempen. *Parallel Machine Scheduling Through Column Generation: Minimax Objective Functions*, volume 4168 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006.