A Cutting Plane Approach for the Multi-Machine Precedence-Constrained Scheduling Problem

Prahalad Venkateshan* Joseph Szmerekovsky† George Vairaktarakis‡

Abstract

A cutting-plane approach is developed for the problem of optimally scheduling jobs with arbitrary precedence constraints on unrelated parallel machines to minimize weighted completion time. While the single machine version of this problem has attracted much research efforts, enabling solving problems with up to 100 jobs, not much has been done on the multiple machines case. A novel mixed-integer programming model is presented for the problem with multiple machines. For this model, many classes of valid inequalities that cut off fractional linear programming solutions are developed. This leads to an increase of the linear programming lower bound from 89.3% to 94.6% of the corresponding optimal solution, and a substantial reduction in the computational time of an optimal branch-and-bound algorithm for this problem. This enables us to report optimal solutions for problem instances with up to 25 jobs and 5 machines, which is more than twice the size of problems for which optimal solutions have been reported in the literature thus far. For a special case of the problem – that of minimizing makespan – application of our model helps solve 18 of 27 previously unsolved problem instances to optimality.

Keywords/Subject Classification: Unrelated Machine Scheduling; Precedence-Constrained Scheduling; Optimization; Integer Programming; Valid Inequalities

1 Introduction

We consider the problem of scheduling jobs on unrelated parallel machines to minimize weighted completion time when the jobs are subject to precedence constraints. This problem is well known in the literature and is denoted as $R\|prec\|\sum_j W_j C_j$ using the classic three field notation of Graham et al. (1979). Since the problem was first introduced, much research has been done on unrelated machine scheduling and it remains an active area of research to this day. For example, Arroyo and Leung (2017a), Arroyo and Leung (2017b), Che at al. (2017), Cheng and Huang (2017), Fanjul-Peyro et al. (2017), Joo and Kim (2017), Shahvari and Logendran (2017), Sitters (2017),

*Indian Institute of Management, Vastrapur, Ahmedabad - 380015, India.prahalad@iima.ac.in

†College of Business, North Dakota State University, Fargo, ND - 58102, USA. joseph.szmerekovsky@ndsu.edu

‡Corresponding author, Weatherhead School of Management, Case Western Reserve University, Cleveland, OH - 44106, USA. gxv5@case.edu. Ph:216-368-5215

and Woo et al. (2017) all represent very recent advances in scheduling unrelated machines. A survey is provided in Mokotoff (2001). Despite the vast amount of literature on unrelated machine scheduling, few papers have addressed environments with precedence constraints. Herrmann et al. (1997) provide heuristics and lower bounds for minimizing the makespan under the special case where precedence constraints form chains of tasks. Numerical experiments are used to show that the heuristics are fast and can provide good, sometimes optimal, solutions. Kumar et al. (2009) also consider a special structure for the precedence constraints; that of treelike precedences. They develop polylogarithmic approximation algorithms for the makespan and weighted completion time objectives. Tavakkoli-Moghaddam et al. (2009) appear to be the first to treat the problem with arbitrary precedence constraints. They consider sequence dependent setup times, release dates, and due-dates while minimizing the bi-objective for number of tardy jobs and total completion time. For this complex problem they provide a genetic algorithm and show that it performs well as compared to solutions obtained by a branch and bound procedure. Liu and Yang (2011) also consider precedence constraints, but consider only the makespan objective. Their approach is to use a heuristic – in this case a priority rule based algorithm – which is shown to improve on the results obtained by Herrmann et al. (1997). Liu (2013) also treats the problem with arbitrary precedence constraints and the objective of minimizing total tardiness. A hybrid genetic algorithm is developed which is shown to outperform a conventional genetic algorithm. Afzalirad and Rezaeian (2016) make use of metaheuristics to minimize the makespan with arbitrary precedence constraints while considering additional resources, sequence dependent setup times, release dates, and machine eligibility. They develop both a genetic algorithm and an artificial immune system algorithm for the problem, showing that the artificial immune system algorithm performs best for large problems.

To the best of our knowledge, Szmerekovsky (2003) is the only other research to consider the problem of scheduling unrelated machines with arbitrary precedence constraints and the objective of minimizing weighted completion times. Szmerekovsky (2003) indicates the many special cases of $R\|prec\|\sum_j W_jC_j$ that are NP-hard. He also presents heuristics and lower bounds that are shown to perform reasonably well for small problems. Given that the combination of precedence constraints and the weighted completion time objective allows for a variety of scheduling environments and objectives to be modeled (i.e. flowshops, jobshops, assembly lines, project scheduling, makespan, tardiness, etc.), the problem under consideration is of vast practical interest. Hence, in this work, we provide an optimal procedure for solving $R\|prec\|\sum_j W_jC_j$ based on the concept of cutting planes.

The first use of cutting planes to solve a scheduling problem appears to be Balas (1985) who used disjunctive programming to explore the facial structure of the scheduling polyhedra for minimizing the makespan in a job shop environment. Cutting planes have also been used to solve single machine scheduling problems. Early efforts include Dyer and Wolsey (1990) who developed a procedure for generating valid inequalities for minimizing the weighted sum of completion times on a single ma-

chine with release dates, Queyranne and Wang (1991) who show how to generate valid inequalities for scheduling on a single machine with series-parallel-type precedence constraints, and Queyranne (1993) who analyzes the polyhedron defined by vectors of completion times for jobs sequenced on a single machine with no precedence constraints. The inequalities developed in Queyranne and Wang (1991) and Queyranne (1993) are not directly useful to problem $R\|prec\|\sum_j W_jC_j$ since there is always the possibility of idle time between jobs scheduled on the same machine due to precedence constraints. Building on these works, Schulz (1995) developed an approximation algorithm for scheduling a single machine with precedence constraints. More recent work includes Šorić (2000) who uses cutting planes for single machine scheduling with job arrivals and setup times to minimize average backlog, and de Farias et al. (2010) who provide valid inequalities for a robust single machine scheduling problem. Mokotoff (2004) develop valid inequalities for parallel machine scheduling with no precedence constraints and makespan minimization objective. These inequalities are of type $\sum_i s_i x_{ij} + k \leq y$, where y is the makespan and the x_{ij} variables indicate assignment of the ith job to the jth machine. Since y is also minimized in the objective, inequalities of such type are quite useful. However, such inequalities do not easily carry over to $R\|prec\|\sum_j W_jC_j$, where even on considering the completion time of a final dummy job as the makespan, its weight in the objective function is 0 since it is a dummy job. As a result, the makespan can be made arbitrarily large without violating such constraints at no penalty to the objective function. Cutting plane approaches have also been used for project scheduling (Olaguíel and Goerlich 1993) and flowshop scheduling (Nishi et al. 2010; Nishi and Hiranaka 2013). Mokotoff and Chrétienne (2002) apply cutting plane methods to solve problem $R\|\|C_{max}$, which is a special case of the problem considered in this work. Chen and Powell (1999) develop a column generation-based approach for solving the parallel machine scheduling problem with independent jobs for various objectives. For the weighted completion time objective, they exploit the fact that in an optimal solution the jobs on every machine follow the shortest weighted processing time (SWPT) order (Smith 1956). Unfortunately, this property does not hold for optimal solutions in the presence of the arbitrary precedence constraints considered in our work. It remains an avenue for further research as to how our model can be specialized to be sufficiently competitive to address problems without precedence constraints in which the jobs scheduled on a machine will satisfy the SWPT order. While computational success has been attained in solving problem instances with up to 100 jobs for the single machine version of the problem (see Potts 1985), similar success has been hard to attain for the multiple machine version. This could be due to some inherent difficulty with such problems. It is pertinent to note that the model provided in Potts (1985), for instance, cannot be generalized easily and immediately to allow for multiple machine scheduling. In particular, equation (3) in Potts (1985) ensures that two jobs i and j are scheduled on the single machine with either i starting before j or vice-versa. This simple property does not hold for the case of multiple machines since both i and j can start simultaneously on different machines.

There have been few compact mixed-integer programming (MIP) models attempting to solve problems with unrelated machines and arbitrary precedence constraints. However, none of these models address the most general weighted completion time objective. Coll et al. (2006) apply cutting plane methods to solve problem $R\|prec\|C_{max}$, which is also a special case of our problem. The authors report lower bounds on a large number of unsolved problem instances in the literature. Hassan et al. (2016) also consider the problem $R\|prec\|C_{max}$. The latter two references make use of variable definitions that are novel and it remains an avenue of further investigation if and how the inequalities derived by those authors are extendable to the model developed in our work. Nonetheless, using our model, in Section 5, we are able to report, to the best of our knowledge, the first optimal results for many of the unsolved instances in Coll et al. (2006). Tavakkoli-Moghaddam et al. (2009) provide a compact MIP formulation for the bi-objective optimization problem of minimizing the number of tardy jobs and the total completion time of jobs. They report solving problems with up to 10 jobs and 2 machines optimally within two hours of CPU time using an off-the-shelf MIP solver LINGO. Liu (2013) provides a MIP formulation for minimizing total tardiness. The problem is solved optimally by the CPLEX MIP solver for instances up to 11 jobs and 2 machines. Furthermore, the MIP models in Tavakkoli-Moghaddam et al. (2009) and Liu (2013) are constrained, requiring that each machine be assigned at least one job and that the number of machines be no greater than the number of jobs. Afzalirad and Rezaeian (2016) provide a MIP model that is not so constrained. However, their model is integer time-indexed making the model unusable for non-integral processing times and in their model they minimize the makespan. Using the LINGO MIP solver, the authors report optimal solutions on problems with up to 6 jobs and 3 machines. The MIP formulation developed in this article is applicable to arbitrary non-integer processing times. We develop valid inequalities and compare the quality of the solutions and the corresponding CPU requirements by directly solving the problem using a state-of-the-art off-the-shelf mixed-integer programming solver (CPLEX). Our results show that our procedure performs significantly better than CPLEX and is capable of solving medium sized problems (up to 25 jobs and 3 machines) to optimality within two hours of CPU time. The remainder of the paper is organized as follows. In Section 2 we formally define the problem and provide our model. In Section 3 we identify new valid inequalities for the model. In Section 4 we detail our computational solution approach. In Section 5, we report our computational experience on using the model. In Section 6 we summarize our conclusions and provide directions for future research.

2 Model

The following notation is used throughout the paper:

Parameters

$\mathcal{N}$: set of N jobs, 1 through N

h, i, j, i', j': indices for jobs

$G(V, E)$: directed acyclic precedence graph (DAG)

$\mathcal{K}$: set of K machines

k, k_1, k_2: indices for machines

B_h^k: 1 if h can be processed on k; 0 otherwise

p_h^k: processing time of h on k, assumed positive and integer (the model developed in this work is applicable even if the processing times are non-integral)

$\overline{\mathcal{N}}$: $\mathcal{N} \cup \{0, N+1\}$, where 0 $(N+1)$ is a dummy job that immediately precedes (succeeds) all jobs in $\mathcal{N}$ with 0 processing time on all machines

w_i: weight of i, assumed non-negative. Also, $w_0 = w_{N+1} := 0$.

If $(i, j) \in E$ then i immediately precedes j, and j immediately succeeds i. Job j can start only after the completion of i. If there is a directed path from h to i, then h precedes i, and i succeeds h. If neither i precedes j, nor j precedes i, then i and j are said to be independent.

Decision Variables

x_{ij}^k: 1 if j is scheduled immediately after i and both are jobs scheduled on k; 0 otherwise

C_i: completion time of i

V_{ij}^k: this equals $C_i x_{ij}^k$. Two inequalities, (5) and (6), in model [MIP] linearize this nonlinear relationship.

Using this notation, the model [MIP] is now presented.

$$\text{[MIP]} \quad \text{Min} \sum_{i=1}^{N} w_i C_i \tag{1}$$

$$\text{s.t.} \sum_{k=1}^{K} \sum_{\substack{i=0 \\ i \neq h}}^{N} B_h^k x_{ih}^k = 1 \qquad \forall h \in \mathcal{N} \tag{2}$$

$$\sum_{j=1}^{N} x_{0j}^k \leq 1 \qquad \forall k \in \mathcal{K} \tag{3}$$

$$\sum_{\substack{i=0 \\ i \neq h}}^{N} x_{ih}^k = \sum_{\substack{i=1 \\ i \neq h}}^{N+1} x_{hi}^k \qquad \forall k \in \mathcal{K}, \forall h \in \mathcal{N} \tag{4}$$

$$V_{ih}^k \leq M x_{ih}^k \qquad \forall k \in \mathcal{K}, \forall i \in \mathcal{N}, \forall h \in \mathcal{N}, i \neq h \tag{5}$$

$$V_{ih}^k \geq C_i - M[1 - x_{ih}^k] \qquad \forall k \in \mathcal{K}, \forall i \in \mathcal{N}, \forall h \in \mathcal{N}, i \neq h \tag{6}$$

$$C_h - \sum_{k=1}^{K} \left\{ p_h^k x_{0h}^k + \sum_{\substack{i=1 \\ i \neq h}}^{N} V_{ih}^k + \sum_{\substack{i=1 \\ i \neq h}}^{N} p_h^k x_{ih}^k \right\} \geq 0 \qquad \forall h \in \mathcal{N} \tag{7}$$

$$C_h - C_i - \sum_{\substack{i'=0 \\ i' \neq h}}^{N} \sum_{k=1}^{K} B_h^k p_h^k x_{i'h}^k \geq 0 \qquad \forall\ (i,h) \in E \tag{8}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall k \in \mathcal{K}, \forall i \in \overline{\mathcal{N}}, \forall j \in \overline{\mathcal{N}}$$

$$V_{ij}^k \geq 0 \qquad \forall k \in \mathcal{K}, \forall i \in \overline{\mathcal{N}}, \forall j \in \overline{\mathcal{N}}$$

$$C_i \geq 0 \qquad \forall i \in \mathcal{N}$$

Objective function (1) minimizes the total weighted completion time. Equality (2) ensures that each job is assigned on exactly one machine. Inequality (3) ensures that at most one job is scheduled to start on a machine. Equation (4) can be thought of as a flow balance constraint which ensures for every machine-job pair that the flow into the job is equal to the flow from the job. Inequalities (5), (6) and the fact that all weights $w_i, \forall i \in \mathcal{N}$ are non-negative, ensure that there exists an optimal solution in which $V_{ij}^k = C_i x_{ij}^k$. Note that this equality is nonlinear while inequalities (5) and (6) help linearize it. The value of M used in these inequalities is derived in Proposition 1. Using V_{ih}^k, inequality (7) ensures that C_h is no less than p_h^k if it starts first on machine k, or no less than $p_h^k + C_i$ if it is scheduled to follow job i on machine k.

Values that are derived based on the parameters and the values of decision variables are now listed. Some of these are constant for a given problem instance while others change due to the decision variables they depend on.

Derived Values

M: an upper bound on the completion time of any job

P_i^{max}: maximum processing time of i amongst all machines it can be processed on; i.e., $P_i^{max} = \max_{k \in \mathcal{K}, B_i^k = 1} p_i^k$

P_i^{min}: minimum processing time of i amongst all machines it can be processed on; i.e., $P_i^{min} = \min_{k \in \mathcal{K}, B_i^k = 1} p_i^k$

$MinCT^k(i)$: a lower bound on the completion time of i if it is processed on k

$MinCT(i)$: a lower bound on the completion time of i

$MinST(i)$: a lower bound on the starting time of i

$MinCT^k(i,j)$: a lower bound on the completion time of j if it is processed on k immediately after k completes processing i. One possible way to compute this is $MinCT^k(i,j) = Max\{MinCT^k(j), MinCT^k(i) + p_j^k\}$.

S_i: starting time of i. It is equal to $C_i - \sum_{k \in \mathcal{K}} \sum_{\substack{h=0 \\ h \neq i}}^{N} p_i^k x_{hi}^k$. Note that this reflects non-preemptive job processing.

$Pred(i)$: set of jobs in G that precede i

$Level(i)$: level of $i \in \mathcal{N}$ in G. Computed as $\max_{h \in Pred(i)} Level(h) + 1$. $Level(0) = 0$.

Proposition 1 helps derive one possible value for M. Note that since G is a DAG, a topological ordering can be obtained. As a result, processing the nodes in ascending order implies that we always process a predecessor before a successor. Using this fact, the procedure to compute the other derived values is now outlined.

---

procedure COMPUTING $MinCT()$

Step 0: Set $MinCT(h) = 0, \forall h \in \overline{\mathcal{N}}$.

Step 1: For each i chosen from $\overline{\mathcal{N}}$ in ascending order, let $predctime = MinCT(i)$.

Step 1.1: For each $j \in \mathcal{N}$ such that $(i,j) \in E$, set $MinCT(j) = max\{MinCT(j), predctime + P_j^{min}\}$.

---

procedure COMPUTING $MinCT^k()$ AND $MinST()$

Step 1: For each i chosen from $\mathcal{N}$ in ascending order, let $maxpredctime = \max_{h \in Pred(i)} MinCT(h)$. This is the maximum amongst all of i's predecessors' $MinCT()$.

Step 1.1: For each machine, k, set $MinCT^k(i) = maxpredctime + p_i^k$.

Step 1.2: After Step 1.1 is completed for all machines k, set $MinST(i) = \min_{k \in \mathcal{K}}[MinCT^k(i) - p_i^k]$

Observe that in Step 1.2 of the procedure to compute $MinCT^k()$ and $MinST()$, model [MIP] can itself be used to compute $MinST(i)$ exactly. Disregarding jobs that are independent of i and jobs that succeed i, the objective can be changed to minimization of S_i. Doing so may increase the value such that it is greater than *maxpredctime*. In our computational tests, we have calculated exactly $MinST(i)$ values for jobs up to level 6 in the precedence diagram. Doing so was not computationally expensive and the total reported times include this computation. The following iterative mechanism is then used to propagate the improved bound in either of $MinCT()$, $MinCT^k()$ or $MinST()$ amongst all three. Note that $MinCT^k(i) \geq MinST(i) + p_i^k$, $MinCT(i) = \min_{k \in \mathcal{K}} MinCT^k(i)$ and $MinCT^k(i) = \max_{h \in Pred(i)} MinCT(h) + p_i^k$. These relationships are iteratively made to hold. This iterative procedure is convergent since $\overline{\mathcal{N}}$ is a finite set and p_i^k's are assumed positive.

3 New Valid Inequalities for $R\|prec\| \sum_j W_j C_j$

In this section, we discuss four classes of inequalities which proved most effective in our computational experiments. Nine other additional classes of inequalities that play an indirect role in improving convergence of our optimization procedure are specified in Appendix A. First, we derive a valid upper bound, M, for the completion time of any job.

Proposition 1. *There exists an optimal solution in which the maximum completion time of any job is* $M = \sum_{i=1}^{N} P_i^{max}$.

Proof. Since $w_i \geq 0$ for all $i \in \mathcal{N}$ there exists an optimal schedule where each job starts at the earliest possible start time, i.e., a non-delay schedule, say S. Consider an arbitrary topological order of activities in $G(V, E)$, say $T = \{i_1, i_2, \ldots, i_N\}$, and suppose that jobs are processed in the this order, one after the other, on a processor k such that $B_{i_r}^k = 1$ for $r = 1, 2, \ldots, N$. Clearly, the makespan of the resulting schedule, say S_T, is $M = \sum_{i=1}^{N} P_i^{max}$ which does not depend on the topological order T. Hence, M provides an upper bound for every job, and any given topological order. Moreover, observe that S_T is not necessarily a non-delay schedule and hence M provides an upper bound for the completion time of any job in S. This completes the proof of the proposition. □

The aforementioned inequalities are presented next.

Inequality class I

Given j, let set $R = \{(k, i) : i$ does not succeed j, and $\delta_{kij} = MinCT^k(i) + p_j^k - MinCT(j) > 0)\}$. Then, the following inequality is valid.

$$C_j \geq MinCT(j) + \sum_{(k,i)\in R} \delta_{kij} x_{ij}^k$$

The validity of the inequality follows from the fact that if j is scheduled to start immediately after i on k, its completion time cannot be less than $MinCT^k(i) + p_j^k$.

Inequality class II

Given two independent jobs, i and j, in $\mathcal{N}$, such that they immediately succeed h. Let sum denote a lower bound on $(C_i - C_h) + (C_j - C_h)$. To compute sum, first, let $sum = \infty$. Choose ordered machine pair, (k_1, k_2), $k_1 \in \mathcal{K}, k_2 \in \mathcal{K}$, not necessarily distinct. If $k_1 = k_2 = k(say)$, perform update $sum = Min\{sum, 2 \times Min\{p_i^k, p_j^k\} + Max\{p_i^k, p_j^k\}\}$. This corresponds to the case where i and j are scheduled on machine k. If $k_1 \neq k_2$, perform update $sum = Min\{sum, p_i^{k_1} + p_j^{k_2}\}$. This corresponds to the case where i is scheduled on k_1 and j on k_2. Iterate over all such pairs (k_1, k_2). Then, the validity of the following inequality follows from the definition of sum:

$$C_i + C_j - 2C_h \geq sum$$

Inequality class III

Consider $i^{'} \in \mathcal{N}$ and let an immediate predecessor in G be $j \in \mathcal{N}$. Then, when $MinCT^k(i,j) > MinST(i^{'})$ we have

$$x_{ij}^k \leq \frac{S_{i'} - MinST(i^{'})}{MinCT^k(i,j) - MinST(i^{'})}$$

Note that $S_{i'} \geq MinST(i^{'})$ and hence the right-hand side can never be negative. If $x_{ij}^k = 1$, then $C_j \geq MinCT^k(i,j)$ and since $S_{i'} \geq C_j$ for $(j, i^{'}) \in E$, we have $S_{i'} - MinST(i^{'}) \geq MinCT^k(i,j) - MinST(i^{'})$. Therefore, the right-hand side is greater than or equal to 1. This proves the validity of the inequality.

Inequality class IV

Given jobs i and j, in $\mathcal{N}$, and machine k, the following inequality holds:

$$MinCT^k(i) x_{ij}^k + x_{ji}^k - \sum_{i'=1, i'\neq j}^{N+1} x_{ii'}^k \leq V_{ij}^k$$

This is because $V_{ij}^k = C_i x_{ij}^k$, and by definition, $MinCT^k(i) x_{ij}^k \leq C_i x_{ij}^k$. If $x_{ij}^k = 1$, then $x_{ji}^k = \sum_{i'=1, i'\neq j}^{N+1} x_{ii'}^k = 0$. If $x_{ji}^k = 1$ instead, then $x_{ji}^k - \sum_{i'=1, i'\neq j}^{N+1} x_{ii'}^k = x_{ij}^k = 0$. In all cases, the inequality holds.

4 Solution Approach

The number of inequalities described in Section 3 and in Appendix A is prohibitively large for apriori inclusion into formulation [MIP]. Instead, we attempt to solve problem [MIP] using the following method:

First, we begin by solving its linear programming relaxation, say LP. Let us define

LB_0: the objective function value of this LP solution.

If the solution obtained has integral values for $\{x_{ij}^k\}$, we are done having found the optimal solution to problem [MIP]. Otherwise, the values obtained by the incumbent LP solution may violate one or more of the 13 inequality classes (four classes discussed earlier, plus nine additional classes discussed in Appendix A). Such violated inequalities are identified efficiently and stored in a list for easy access. This is accomplished by solving a *separation* subproblem for each inequality class. In principle, the idea behind the separation subproblem is to evaluate the left-hand side (lhs) of inequalities and compare it to the right-hand side (rhs). If the inequality class can be represented as $lhs \leq rhs$ (e.g., class III and class IV) a violation by a fractional solution occurs if $lhs > rhs$, with violation amount $lhs - rhs$. If the inequality class can be represented as $lhs \geq rhs$ (e.g., class I and class II) a violation by a fractional solution occurs if $lhs < rhs$, with amount of violation equal to $rhs - lhs$. To make this identification process more efficient, as a preprocessing step, all the constant derived values are stored in appropriate data structures; multidimensional arrays proved sufficient for this purpose. The separation routine then loops through all jobs and machines calculating lhs and rhs based on the incumbent LP solution $\{x_{ij}^k\}$. In all of our computational tests, the preprocessing time spent on separation routines and the actual separation of inequalities were completed within 5 seconds on every problem instance and for this reason we do not report separate CPU times for these routines. These times are included in the total reported computational times.

The process described above iterates between LP re-optimization and finding new violations by solving the separation subproblem until all four classes of inequalities presented above as well as those in Appendix A are satisfied. Let us define

LB_1: the objective function value of the LP relaxation for which all valid inequalities presented are satisfied.

When all valid inequalities are satisfied for the incumbent solution $\{x_{ij}^k\}$, the incumbent LP includes all the inequalities that were found to be violated in previous iterations. This expanded LP formulation is submitted to an off-the-shelf MIP solver (CPLEX, in our case) for solving after replacing the constraints $x_{ij}^k \geq 0$ by $x_{ij}^k \in \{0, 1\}$ for all i, j, k.

The basic framework presented thus far was enhanced in several ways for greater computational efficiency. Whenever a violated inequality with violation of 0.5 or greater is found, the separation

procedure for that inequality class is terminated. Until this threshold is met, violated inequalities with increasing amounts of violation are stored. That is, if the first stored inequality has a violation of 0.2, the next violated inequality is stored only if its violation exceeds 0.2. The most violated inequality so found is added to the LP and the LP is re-optimized. Then, the stored inequalities from the previous iteration are checked so as to identify a violation using the new incumbent LP solution. If one exists, the most violated inequality is added to LP which is then re-optimized. Thus, the separation subproblem need not be explicitly solved as long as previously identified inequalities continue to be violated by the incumbent LP solution. If none of the previously identified inequalities is violated, the separation subproblem is re-solved. Finally, we should mention that the threshold value of 0.5 was selected by trial and error; it seems to be a good balance between the size of the list of violated inequalities, and the number of times this list is refilled with newly violated inequalities.

To evaluate the relative contribution of the various valid inequalities presented in this article we define the following:

Z^*: the optimal value for problem [MIP],

$lb_0 := \frac{LB_0}{Z^*}$,

$lb_1 := \frac{LB_1}{Z^*}$.

Evidently, $lb_1 - lb_0$ reflects the effectiveness of our valid inequalities as a mechanism for improving the lower bound for Z^*, above and beyond the value produced by the straightforward LP relaxation of [MIP]. In particular, the computational experiments discussed in detail in the following section, demonstrate that for 25-job, 5-machine problem instances

- Class I of inequalities by themselves improve $lb_0 = 92.8\%$ to $lb_1 = 97\%$,
- Class II of inequalities by themselves improve $lb_0 = 92.8\%$ to $lb_1 = 94\%$,
- Class III of inequalities by themselves improve $lb_0 = 92.8\%$ to $lb_1 = 93.1\%$,
- Class IV of inequalities by themselves improve $lb_0 = 92.8\%$ to $lb_1 = 93\%$.

We proceed with detailed computational experiments in the next section.

5 Computational Tests

In this section, results from computational tests are reported. These were carried out on an Intel i5 processor with clockspeed 3.2 GHz and 8 GB RAM. CPLEX 12.7 was used as the LP/MIP solver. The methodology was tested on different sets of problem classes with varying characteristics – a 25-job, 5-machine set, and a 20-job, 3-machine set. The project scheduling problem generation method

PSPLIB (Kolisch and Sprecher 1997) was used to generate the precedence graph with a project complexity factor of 1.25 and 2.5. Project complexity is a measure of the average number of arcs leaving a job in the precedence graph. Thus, the methodology was tested on four different sets of problem classes. Each set comprised of 10 different problem instances. Evidently, this construction method used allows for redundant edges. For instance, if $(h,i),\ (i,j) \in E$, (h,j) is redundant. As a result, a better measure of project complexity is the *graph density* which is calculated as the ratio of the number of non-redundant arcs in G to the total number of arcs possible in G; i.e. $\binom{|V|}{2}$. The average graph density was 9.5% for the 25-job, 5-machine set with complexity factor 1.25, 11.6% for the 20-job, 3-machine set with complexity factor of 1.25, 17.5% for the 25-job 5-machine set with complexity factor of 2.5, and 18.7% for the 20-job, 3-machine set with complexity factor of 2.5. The weights and processing time of a job on a machine are randomly generated integers in the range [1,10]. CPLEX-based implementations of branch-and-cut allow the addition of violated cuts at nodes other than the root node. However, application of branch-and-cut for our problem did not lead to improvement in the computational time. The tradeoff is between the size of the branch-and-bound tree and the LP-reoptimization time within each node. As more cuts are added within the tree, the problem size increases dynamically leading to greater LP-reoptimization time. However, this has the potential of reducing the total tree size. Three different implementation versions were compared: (1)adding violated inequalities only at the root node, (2)adding violated inequalities at all nodes at a depth of 10 or less in the tree, and (3)adding violated inequalities at all nodes in the tree. On one set of 10 problem instances (see Table 3), version (1) required an average total computational time of 37 seconds per problem and an average tree size of 6924 nodes per problem. Version (2) required time of 101 seconds and 1864 nodes. Version (3) required 170 seconds and 1152 nodes. As a result, Version (1), in which violated inequalities are only added at the root node, was chosen for further computational tests. For each problem instance reported in this section, a two-hour time limit was provided. All problem instances on which results are reported in this work are available for download from the online supplement.

5.1 Value of Additional Inequalities

A commonly encountered phenomenon in cutting plane approaches to different mixed-integer programming problems is the "snowballing effect" (e.g., see Agarwal 2018). For a given fractional solution $\{x_{ij}^k\}$, no violation may occur of a particular inequality class. However, when a new inequality class is added, the new fractional solution may violate a large number of inequalities of classes previously considered. This is experienced in our cutting plane approach as well. It is also observed that, the greater the number of inequalities added to [MIP], the less the resulting computational time to solve the problem; similar experience has been reported in Correia et al. (2012), among others. This is demonstrated on all problem classes. Initially, each of the different

inequality classes (the four described in Section 3 and the nine in Appendix A) was individually and separately added to LP. As indicated in Section 3, only the four inequalities described in that section led to an improvement in lb_1 over lb_0. Amongst these inequalities, class I led to the greatest increase in lb_1. Table 1 compares the impact of adding all violated inequalities as opposed to just class I of inequalities. This is done separately for each of the 10 problem instances and the average values across these 10 instances are reported in Table 1.

Table 1: Effect of additional inequalities on different problem classes

Problem Class	**Method Used**	**Avg.** $lb_1(\%)$	**Avg. No. Nodes**	**Avg. Time (in secs.)**	**Avg. No. Inequalities**
25 Jobs, 5 Machines	Only Class I	97	24,210	132	20
1.25 Complexity	All Inequalities	97.1	17,905	88	216
25 Jobs, 5 Machines	Only Class I	96.5	10,671	74	35
2.5 Complexity	All Inequalities	97.2	6924	37	208
20 Jobs, 3 Machines	Only Class I	91.5	589,839	2271	32
1.25 Complexity	All Inequalities	92.3	469,575	1609	195
20 Jobs, 3 Machines	Only Class I	90.9	655,616	2070	32
2.5 Complexity	All Inequalities	91.8	226,525	352	204

Evidently, a significant number of violated inequalities come from the 12 classes other than class I of inequalities; on average 216-20=196 versus 20, respectively, for the 25-jobs, 5-machines, 1.25-complexity case, for instance. It was also observed that the number of violated inequalities in the above table corresponding to the method used of adding all inequalities classes is larger compared to only considering 2 classes of inequalities simultaneously; class I and any one of the remaining 12 classes. As we saw in the previous section, the separation of the total of 13 classes of valid inequalities is not computationally expensive while considering multiple classes helps us identify many more violations. For these reasons, in all subsequent computational tests we add violated inequalities from all 13 classes.

It is now time to discuss the merits of the valid inequalities included in Appendix A. While these nine inequality classes by themselves do not lead to a significant improvement in lb_1, their inclusion to the LP nonetheless perturbs the LP solution $\{x_{ij}^k\}$ in a material way so as to significantly affect the CPU time expended by CPLEX. Given a fractional solution $\{x_{ij}^k\}$, the CPLEX MIP solver employs a branch-and-bound algorithm in its search for an optimal integral solution $\{x_{ij}^k\}$. The number of nodes in this branch-and-bound tree as well as the resulting CPU times are reported in Table 1. Evidently, the addition of the inequalities in Appendix A reduces the average size of the branch-and-bound tree from 24,210 nodes to 17,905, and the corresponding CPU time from 132 seconds to 88 seconds on average; a 33% improvement for the 25-jobs, 5-machines, 1.25-complexity case, for instance. (Such improvements can be seen for other problem classes as well.) This is despite the fact that the associated lb_1 values are quite similar (97% vs. 97.1%) using four versus all 13 classes of inequalities. This indicates that it may be possible to obtain further computational

efficiencies by discovering new classes of valid inequalities.

5.2 Comparison with CPLEX

In this section, the cutting plane approach is compared against the only other exact alternative approach available; namely, simply submitting formulation [MIP] to an off-the-shelf MIP solver. Tables 2, 3, 4 and 5 report on the computational experience. Different CPLEX settings were tried in an attempt to improve computing times. This included an emphasis within branch and bound on lower bound improvement (MIPEMPHASIS BESTBOUND) since as will be seen shortly, improving the lower bound within the tree is the primary challenge in solving larger problem instances. Another attempt was made to generate cuts more aggressively than the default setting. Yet, the overall performance was worse off in all cases. As a result, default CPLEX settings were used.

Table 2: Comparison with CPLEX Solver for 25-job, 5-machine, 1.25 complexity instances

Instance	lb_0	**CPLEX Solver**		**Cutting Plane Approach**			
	(in %)	**Nodes**	**Time (in secs)**	lb_1 **(in %)**	**Nodes**	**Time (in secs)**	**Violated Inequalities**
1	92.1	1,459,998	7200[1]	97.5	2604	20	240
2	97	62,867	134	99.3	1405	11	160
3	95.4	332,833	950	98.9	301	6	170
4	94	545,685	2333	97.3	2049	14	134
5	96.7	2,015,597	3644	99.7	44	6	230
6	94.2	2,252,997	7200[2]	96.7	12,219	61	326
7	94.6	1,307,049	3515	97.8	4238	20	194
8	93.2	520,021	1150	98.1	825	10	197
9	83.4	1,833,002	7200[3]	91	62,898	278	269
10	88.1	837,898	7200[4]	95	92,467	457	242
Average	**92.8**	**1,116,795**	**4053**	**97.1**	**17,905**	**88**	**216**

[1]Gap at termination: 3.86%, Unexplored nodes: 1,120,165

[2]Gap at termination: 0.4%, Unexplored nodes: 191,401

[3]Gap at termination: 7.1%, Unexplored nodes: 1,470,683

[4]Gap at termination: 8.2%, Unexplored nodes: 711,409

Table 3: Comparison with CPLEX Solver for 25-job, 5-machine, 2.5 complexity instances

Instance	lb_0	**CPLEX Solver**		**Cutting Plane Approach**			
	(in %)	**Nodes**	**Time (in secs)**	lb_1 **(in %)**	**Nodes**	**Time (in secs)**	**Violated Inequalities**
1	87.7	1,844,400	7200[1]	94.1	54,760	237	221
2	94	17,819	50	99.4	9	5	215
3	94.1	7487	46	98.1	14	4	213
4	96.9	58,654	120	98	224	6	153
5	93	526,751	1891	95.7	4392	33	308
6	98.9	302,857	690	99.7	229	9	186
7	90.1	313,842	835	95.2	1748	26	225
8	97.9	24,966	60	99.8	87	6	192
9	91.2	834,595	1503	95.3	5598	41	236
10	93.5	84,075	226	97	2177	10	133
Average	**93.7**	**401,545**	**1262**	**97.2**	**6924**	**37**	**208**

[1]Gap at termination: 6.86%, Unexplored nodes: 1,505,112

Table 4: Comparison with CPLEX Solver for 20-job, 3-machine, 1.25 complexity instances

Instance	lb_0	**CPLEX Solver**		**Cutting Plane Approach**			
	(in %)	**Nodes**	**Time (in secs)**	lb_1 **(in %)**	**Nodes**	**Time (in secs)**	**Violated Inequalities**
1	83	749,400	7200[1]	91.4	526,417	739	172
2	78.9*	2,294,676	7200[2]	87*	1,031,913	7200[6]	235
3	84.8	4,118,297	7200[3]	91.1	531,855	1572	185
4	84.2	3,387,973	3021	92.6	121,403	137	193
5	89.9	758,370	7200[4]	94.1	16,376	25	222
6	86.2	3,699,756	3042	94.3	17,715	35	221
7	93.5	587,841	376	97.4	11,916	16	159
8	80.1	6,170,200	7200[5]	89	613,135	1520	213
9	81.8	5,304,857	4187	90	1,751,512	4753	169
10	91.4	577,261	331	96.9	76,544	96	178
Average	**85.3**	**2,764,863**	**4696**	**92.3**	**469,575**	**1609**	**195**

[1]Gap at termination: 7.8%, Unexplored nodes: 640,580

[2]Gap at termination: 10.1%, Unexplored nodes: 1,847,686

[3]Gap at termination: 1.6%, Unexplored nodes: 852,439

[4]Gap at termination: 6.2%, Unexplored nodes: 591,337

[5]Gap at termination: 7.9%, Unexplored nodes: 4,044,403

[6]Gap at termination: 3.5%, Unexplored nodes: 547,075

*Best estimate after 7200 seconds since this problem was not solved to optimality

Table 5: Comparison with CPLEX Solver for 20-job, 3-machine, 2.5 complexity instances

Instance	lb_0 (in %)	CPLEX Solver		Cutting Plane Approach			
		Nodes	Time (in secs)	lb_1 (in %)	Nodes	Time (in secs)	Violated Inequalities
1	81.5	636,799	7200[1]	89	614,848	1245	194
2	88.6	136,525	76	93.5	38,739	40	216
3	86	440,168	1596	92.7	27,072	45	195
4	85.9	964,593	4346	91.5	14,370	26	189
5	90.4	903,439	7200[2]	93	34,849	62	283
6	85	5,998,429	7200[3]	93	133,212	186	188
7	77.1	3,094,670	3467	86.8	1,252,610	1694	177
8	90	716,628	7200[4]	95.6	50,388	60	159
9	85.6	512,804	7200[5]	91.1	56,086	87	192
10	87.7	516,792	386	92.5	43,081	79	249
Average	**85.7**	**1,392,084**	**4587**	**91.8**	**226,525**	**352**	**204**

[1]Gap at termination: 6.9%, Unexplored nodes: 491,364

[2]Gap at termination: 2.5%, Unexplored nodes: 404,896

[3]Gap at termination: 8.1%, Unexplored nodes: 4,405,867

[4]Gap at termination: 1.6%, Unexplored nodes: 295,951

[5]Gap at termination: 6.2%, Unexplored nodes: 383,864

From these tables we observe that the 2-hour time limit is more than sufficient for our cutting plane approach in 39 of the 40 problem instances generated. In contrast, without the benefit of our procedures the CPLEX solver encounters significant difficulty in solving the same problems. Using the CPLEX solver, 15 out of the 40 problems tested do not generate an optimal solution within 2 hours. The average computing times are smaller for 5-machine instances than for 3-machine instances. A partial explanation for this behavior is that the gaps $100\% - lb_0$ and $100\% - lb_1$ are much greater for 3-machine problems than for 5-machine instances indicating higher integrality gap that has to be closed by the MIP solver. For 20-job, 3-machine instances the inequalities help improve the lower bound from 85.3% to 92.3% of Z^* when project complexity is 1.25, and from 85.7% to 91.8% of Z^* when project complexity is 2.5. In nearly all cases, and with both the CPLEX and to a lesser extent our cutting plane approach, the optimal solution is discovered very early in the branching process. The majority of the time is spent in validating the optimality of the incumbent solution. Hence, the improvement in the lower bound from lb_0 to lb_1 is highly beneficial to the solution process. Also, this suggests that our cutting plane approach with a tight CPU time limit offers a good heuristic alternative for solving [MIP].

Having established the superiority of the cutting plane method for the four problem sizes considered until now, in what follows we explore the boundary of problem sizes that can be solved in reasonable time (2 hours in our experiments) by CPLEX and our cutting plane method. In particular, we test the two approaches on 25-job, 3-machine instances with project complexity 1.25, and on 30-job, 5-machine instances with complexity 1.25.

Table 6: Comparison with CPLEX Solver for 25-job, 3-machine, 1.25 complexity instances

Instance	CPLEX Solver		Cutting Plane Approach		
	Gap at termination (in %)	Unexplored Nodes	Gap at termination (in %)	Unexplored Nodes	Violated Inequalities
1	18.6	2,917,743	8.4	476,106	321
2	12.3	3,567,369	5.8	558,080	225
3	17.1	2,479,502	9.5	796,365	291
4	18.7	1,085,191	11.9	921,964	248
5	7.7	3,223,477	4.5	524,606	342
6	19.4	562,668	14.3	509,451	309
7	7.6	2,418,236	3.0	432,269	299
8	17.3	1,631,132	9.4	955,860	269
9	23.4	1,557,876	14.6	408,777	278
10	5.3	2,131,160	0.0	0	436
Average	**14.7**	**2,157,435**	**8.1**	**558,348**	**302**

Table 7: Comparison with CPLEX Solver for 30-job, 5-machine, 1.25 complexity instances

Instance	CPLEX Solver		Cutting Plane Approach		
	Gap at termination (in %)	Unexplored Nodes	Gap at termination (in %)	Unexplored Nodes	Violated Inequalities
1	9.3	841,735	5.5	128,900	305
2	3.5	351,366	0.0	0	308
3	4.3	329,940	0.0	0	247
4	8.3	670,921	2.0	363,560	274
5	6.9	318,924	0.0	0	293
6	1.2	411,705	0.0	0	278
7	4.8	457,872	0.0	0	363
8	8.5	358,962	0.0	0	337
9	19.9	240,692	6.6	201,161	301
10	10.7	261,801	0.6	114,354	281
Average	**7.8**	**424,392**	**1.5**	**80,798**	**299**

Tables 6 and 7 show the gap at termination and the number of unexplored nodes at the end of the two-hour time limit. For problem instances with 25 jobs and 3 machines just one instance (instance 10) was optimally solved by the cutting plane approach (in 3986 seconds) within the 2-hour limit, indicating limited effectiveness. For problem instances with 30 jobs and 5 machines, six of the ten problem instances were solved by the cutting plane approach (in 1260 seconds on average). One may say that the corresponding performance of the direct CPLEX approach is abysmal: none of these 20 instances are solved to optimality within two hours. Even in problems where both methods fail to produce an optimal solution within two hours of CPU time, it can be seen that the cutting plane approach is able to close the optimality gap to a much greater extent than direct CPLEX, uniformly across all 20 instances.

5.3 Comparison on pre-existing problem instances in the literature

In this section, we apply the method developed in this paper on problem $R||prec||C_{max}$. Problem instances were obtained from Coll et al. (2006). That work reports the lower bound obtained using their method. However, it does not report the computational time to obtain the lower bound. The authors report a large number of problem instances that are as yet unsolved where there is a non-zero gap between their lower bound and the best known heuristic solution. The authors classify problems into small instances and large instances. The small instances contain up to 25 jobs. There were 27 unsolved such instances. To the best of our knowledge, we are the first to report optimal solutions on 18 of these 27 problems (See Table 8.) For many of these problem instances, in addition to proving optimality, our optimal solution was also better than the best known heuristic solutions. Two of the problems reported remain unsolved even after 2 hours of computational time, yet the final heuristic solution is better than that known in the literature. For the 7 unreported problems, the final heuristic solution (the solution is heuristic since computation did not complete even after 2 hours) did not improve on the best reported solution in Coll et al. (2006).

Table 8: Comparison with benchmark instances of $R||prec||C_{max}$

Instance	**Optimal Solution**[1]	**Time (in secs.)**	**Nodes**
v.22.8.5	15*	1434	141,918
f.14.8.5	13	25	0
f.14.4.5	13	101	0
d.25.8.5	20*	7200[2]	319,833
d.25.4.5	20*	7200[2]	293,120
d.25.8.2	14*	202	0
d.25.4.2	14*	493	23,791
f.14.8.2	8	29	0
v.22.8.10	18*	2186	133,643
v.22.8.2	10*	81	0
d.25.8.10	23*	1884	102,023
d.25.4.10	23*	1976	178,563
g.23.4.2	13	3855	102,141
f.14.2.5	13	7	0
r.24.8.5	18	1429	85,502
r.24.4.5	18	1124	135,171
i.22.4.10	19*	1589	145,858
r.24.8.10	20*	204	18,097
g.23.8.10	21	3492	641,356
d.25.2.10	24	5031	1,710,581

* - New solution

[1] - Solution is optimal only if Time (in secs.) is less than 7200, else it is heuristic

[2] - Problem unsolved even after 2 hours of computational time

6 Conclusion

The contribution of this paper is two-fold. First, a novel compact MIP formulation was provided for problem $R||prec|| \sum_j W_j C_j$ in scheduling literature. This problem of minimizing weighted completion time on unrelated machines and arbitrary job precedence constraints is a very general scheduling problem that captures a variety of other scheduling problems as a special case. Secondly, for this formulation, a large number of classes of valid inequalities were reported. These inequalities play a significant role in helping to solve problem instances with up to 25 jobs and 5 machines. In comparison, solving these problems optimally using CPLEX directly on [MIP] was not possible within a two-hour time limit. Quick improvement of the lower bound in the branch-and-bound tree is a major challenge in solving this class of problems optimally. The inequalities presented in this article lead to an improvement in the lower bound from 85.3% to 92.3% of the optimal solution Z^* on the most difficult problem instances, leading to a decrease in the number of branch-and-bound nodes explored within CPLEX, and the resulting computational time. The average improvement in the lower bound across all 40 problem instances in Tables 2 through 5 is from 89.3% to 94.6% of Z^*.

The problem sizes on which optimal solutions have been reported in our work (i.e., up to 25 jobs) are more than twice the size of problems for which optimal solutions have been reported in related literature. For even larger problem sizes, discovery of newer classes of inequalities and possibly facet-defining inequalities are imperative. Since the problem considered is quite general, it will be worthwhile to see if any of the inequalities reported in our work can be tightened for specialized problems.

Acknowledgements

The authors would like to thank the authors of Coll et al. (2006) for sharing problem instances reported in their work. All of the problem instances on which solutions have been reported in our paper have been made available as an online supplement. We also thank the editor and the two anonymous referees whose comments helped improve the contribution of the paper.

References

Agarwal, Y.K. 2018. Network Loading Problem: Valid Inequalities from 5-and Higher Partitions. *Computers & Operations Research*. 99. 123–134.

Afzairad, M., J. Rezaeian. 2016. Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. *Computers and Industrial Engineering*. 98. 40-52.

Arroyo, J.E.C., J.Y.-T. Leung. 2017a. An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non-identical capacities and unequal ready times. *Computers and Industrial Engineering*. 105. 84-100.

Arroyo, J.E.C., J.Y.-T. Leung. 2017b. Scheduling unrelated parallel batch processing machines with non-identical job sizes and unequal ready times. *Computers & Operations Research*. 78. 117-128.

Balas, E. 1985. On the facial structure of scheduling polyhedra. *Mathematical Programming Study*. 24. 179–218.

Che, A., S. Zhang, X. Wu. 2017. Energy-conscious unrelated parallel machine scheduling under time-of-use electricity tariffs. *Journal of Cleaner Production*. 156. 688–697.

Chen, Z-L., W.B. Powell. 1999. Solving parallel machine scheduling problems by column generation. *INFORMS Journal on Computing* 11.1. 78–94.

Cheng, C.-Y., L.W. Huang. 2017. Minimizing total earliness and tardiness through unrelated parallel machine scheduling using distributed release time control. *Journal of Manufacturing Systems*. 42. 1-10.

Coll, P.E., C.C. Ribeiro, C.C. de Souza. 2006. Multiprocessor Scheduling Under Precedence Constraints: Polyhedral Results. *Discrete Applied Mathematics*. 154. 770–801.

Correia, I., L.L. Lourenço, F. Saldanha-da-Gama. 2012. Project Scheduling with Flexible Resources: Formulation and Inequalities. *OR Spectrum*. 34. 635–663.

de Farias Jr, I.R., H. Zhoa, M. Zhao. 2010. A family of inequalities valid for the robust single machine scheduling polyhedron. *Computers & Operations Research*. 37. 1610–1614.

Dyer, M.E., L.A. Wolsey. 1990. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*. 26. 255-270.

Fanjul-Peyro, L., F. Perea, R. Ruiz. 2017. Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. *European Journal of Operatonal Research*. 260. 482-493.

Graham, R.L., E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan. 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*. 5. 287-326.

Hassan, M.A., I. Kacem, S. Martin, I.M. Osman. 2016. Unrelated Parallel Machine Scheduling Problem with Precedence Constraints: Polyhedral Analysis and Branch-and-Cut. Combinatorial Optimization: 4th International Symposium, ISCO 2016. 308–319.

Herrmann, J., J.-M. Proth, N. Sauer. 1997. Heuristics for unrelated machine scheduling with precedence constraints. *European Journal of Operational Research*. 102. 528-537.

Joo, C.M., B.S. Kim 2017. Rule-based meta-heuristics for integrated scheduling of unrelated parallel machines, batches, and heterogeneous delivery trucks. *Applied Soft Computing*. 53. 457-476.

Kolisch, R., A. Sprecher. 1997. PSPLIB-a project scheduling problem library: OR software-ORSEP operations research software exchange program. *European Journal of Operational Research*. 96.1. 205–216.

Kumar, V.S.A., M.V. Marathe, S. Parthasarathy, A. Srinivasan. 2009. Scheduling on Unrelated Machines under Tree-Like Precedence Constraints. *Algorithmica*. 55. 205-226.

Liu, C. 2013. A Hybrid Genetic Algorithm to Minimize Total Tardiness for Unrelated Parallel Machine Scheduling with Precedence Constraints. *Mathematical Problems in Engineering*. Article ID 537127.

Liu, C., S. Yang. 2011. A heuristic serial schedule algorithm for unrelated parallel machine scheduling with precedence constraints. *Journal of Software*. 6(6). 1146-1153.

Mokotoff, E. 2001. Parallel machine scheduling problems: A survey. *Asia-Pacific Journal of Operational Research*. 18.2. 193–242.

Mokotoff, E. 2004. An exact algorithm for the identical parallel machine scheduling problem. *European Journal of Operational Research* 152.3 (2004): 758–769.

Mokotoff, E., P. Chrétienne. 2002. A Cutting Plane Algorithm for the Unrelated Parallel Machine Scheduling Problem. *European Journal of Operational Research*. 141. 515–525.

Nishi, T., Y. Hiranaka. 2013. Lagrangian relaxation and cut generation for sequence-dependent setup time flowshop scheduling problems to minimise the total weighted tardiness. *International Journal of Production Research*. 51(16). 4778-4796.

Nishi, T., Y. Hiranaka, M. Inuiguchi. 2010. Lagrangian relaxation with cut generation for hybrid flowshop scheduling problems to minimize the total weighted tardiness. *Computers & Operations Research*. 37. 189-198.

Olaguíel,R.S.-V., J.M.T. Goerlich 1993. The project scheduling polyhedron: Dimension, facets, and lifting theorems. *European Journal of Operational Research*. 67. 204-220.

Potts, C.N. 1985. A Lagrangean based branch and bound algorithm for single machine sequencing with precedence constraints to minimize total weighted completion time. *Management Science*. 31.10: 1300–1311.

Queyranne, M. 1993. Structure of a simple scheduling polyhedron. *Mathematical Programming*. 58. 263–285.

Queyranne, M., Y. Wang. 1991. Single-machine scheduling polyhedra with precedence constraints. *Mathematics of Operations Research*. 16(1). 1-20.

Schulz, A.S. 1995. Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds. International Conference on Integer Programming and Combinatorial Optimization. Springer, Berlin, Heidelberg.

Shahvari, O., R. Logendran. 2017. An Enhanced tabu search algorithm to minimize a bi-criteria objective in batching and scheduling problems on unrelated-parallel machines with desired lower bounds on batch sizes. *Computers & Operations Research*. 77. 154-176.

Sitters, R. 2017. Approximability of average completion time scheduling on unrelated machines. *Mathematical Programming Series A*. 161. 135-158.

Smith, W.E. 1956. Various optimizer for single-stage production. *Naval Research Logistics Quarterly* 3:59–66.

Šorić, K. 2000. A cutting plane algorithm for a single machine scheduling problem. *European Journal of Operational Research*. 127. 383–393.

Szmerekovsky, J.G. 2003. Maximizing Project Net-Present Value and Minimizing Work-in-Progress Costs in Projects. Case Western Reserve University, Cleveland, Ohio.

Tavakkoli-Moghaddan, R., F. Taheri, M. Bazzazi, M. Izadi, F. Sassani 2009. Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints. *Computers & Operations Research*. 36. 3224-3230

Woo, Y.-B., S. Jung, B.S. Kim. 2017. A rule-based genetic algorithm with an improvement heuristic for unrelated parallel machine scheduling problem with time-dependent deterioration and multiple rate-modifying activities. *Computers and Industrial Engineering*. 109. 179-190.

7 Appendix A: More Valid Inequalities

In this section, nine different classes of valid inequalities are presented. While inclusion of these inequalities did not help improve lb_1, these inequalities play a significant role in fathoming the branch-and-bound tree quicker; refer to Section 5.1 for empirical evidence. Additional notation is now introduced that is needed to describe these inequalities.

Derived Values

$MinC^k(S)$: a lower bound on the sum of completion times of jobs in set S, all scheduled to be processed on k

$fthr^k(i,j,i^{'})$: directed flowthrough (defined below) on k along $i \rightarrow j \rightarrow i^{'}$

$InActivity_j^k$: $\sum_{i=0,i\neq j}^{N} x_{ij}^k$, denoting the total flow into job j on k

Flowthrough, $fthr^k(i,j,i^{'})$, denotes a lower bound on the "flow" (as defined by the x variables) originating at i and reaching $i^{'}$ via j. This is calculated as $fthr^k(i,j,i^{'}) = x_{ij}^k - \sum_{h=1,h\neq i,h\neq i^{'}}^{N+1} x_{jh}^k$. Intuitively, this captures the "incoming" flow at j from i, reduced by the amount of flow that leaves j but does not go to $i^{'}$.

Inequality class 1

Consider machine k and a subset $S \subseteq \mathcal{N}$ of s jobs that complete on k. Then, the following inequality holds:

$$\sum_{i\in S} C_i \geq MinC^k(S)\Big[\sum_{i\in S}\sum_{h=0,h\neq i}^{N} x_{hi}^k - s + 1\Big]$$

The validity of the inequality stems from the fact that in an integral solution, the right-hand side of the inequality equals $MinC^k(S)$ if the jobs in S are scheduled on machine k, and negative otherwise.

Computing $MinC^k(S)$ is based on the following observation. Given a set of jobs, $i_1, \ldots, i_s$, re-indexed if necessary such that $p_{i_1}^k \leq \ldots \leq p_{i_s}^k$, a lower bound on the sum of their completion times on machine k is $p_{i_1}^k + (p_{i_1}^k + p_{i_2}^k) + \ldots + (p_{i_1}^k + p_{i_2}^k + \ldots + p_{i_s-1}^k + p_{i_s}^k)$.

Inequality class 2

If h succeeds j in G, then, for a machine k, the following inequality holds:

$$\sum_{i=0,i\neq j}^{N} x_{ij}^k \leq 1 - x_{0h}^k$$

The validity of the inequality stems from the fact that if h is scheduled to be processed first on k, the right-hand side is 0. The left-hand side of the inequality, $InActivity_j^k$, cannot be positive as j cannot be scheduled on k as j should complete before h's start.

Inequality class 3

Consider triplet $h, i, j \in \mathcal{N}$, each job distinct, and two distinct machines k_1 and k_2. Then, the following inequality holds:

$$x_{hi}^{k_1} + x_{ih}^{k_1} + x_{hj}^{k_2} + x_{jh}^{k_2} \leq 1$$

The validity of the inequality follows from the fact that if $x_{hi}^{k_1} = 1$, i is scheduled on k_1 immediately after h. This precludes the possibility of h being scheduled after i, or of j being scheduled after h on k_2 or of h being scheduled after j on k_2. Other cases are similarly proven.

Inequality class 4

Consider pair $i, j \in \mathcal{N}$, each job distinct, and two distinct machines k_1 and k_2. The following inequality holds:

$$x_{ij}^{k_1} + x_{ji}^{k_1} + \sum_{h=0, h\neq j}^{N} x_{hj}^{k_2} \leq 1$$

The validity of this inequality is proven along similar lines to that of Inequality 3.

Inequality class 5

Consider distinct jobs $i, j \in \mathcal{N}$. Then, the following inequality holds:

$$\sum_{k\in\mathcal{K}} x_{ij}^{k} + \sum_{k\in\mathcal{K}} x_{ji}^{k} \leq 1$$

The validity of this inequality follows since if i immediately precedes j on a machine k, then neither can j immediately precede i on any machine, nor can i immediately precede j on machine other than k.

Inequality class 6

Consider $j \in \mathcal{N}$ and k. The following inequality holds:

$$\sum_{i=0, i\neq j}^{N} V_{ij}^{k} + p_j^k \sum_{i=0, i\neq j}^{N} x_{ij}^{k} \leq \sum_{i=1, i\neq j}^{N+1} V_{ji}^{k}$$

In a feasible integer solution, the left-hand side is the sum of the completion time of job i that immediately precedes j in a schedule, both on k and the processing time of j. The right-hand side denotes the actual completion time of j on machine k. This proves the validity of the inequality.

Inequality class 7

Consider the case where $fthr^k(i, j, i^{'}) = 1$. In this case, the lower bound on the completion time of $i^{'}$ can be updated to $C = max(MinCT^k(i^{'}), p_i^k + p_j^k + p_{i^{'}}^k)$. Then, for $h \notin \{i, j, i^{'}\}$, the following inequality holds:

$$V_{i^{'}h}^{k} \geq (C - MinCT^k(i^{'}))(fthr^k(i, j, i^{'}) - 1) + Cx_{i^{'}h}^{k}$$

To prove the validity of this inequality, first note that in a feasible solution, the possible values of $fthr^k(i, j, i^{'})$ can be 1, 0 or -1. If $fthr^k(i, j, i^{'}) = 1$, then $V_{i^{'}h}^{k} = C_{i^{'}} x_{i^{'}h}^{k} \geq Cx_{i^{'}h}^{k}$. If $fthr^k(i, j, i^{'}) = 0$, the right-hand side of the inequality becomes $MinCT^k(i^{'}) - C(1 - x_{i^{'}h}^{k})$. If $x_{i^{'}h}^{k} = 1$, the inequality simplifies to $C_{i^{'}} \geq MinCT^k(i^{'})$. If $x_{i^{'}h}^{k} = 0$, the right-hand side is non-positive and the inequality is trivially true. The case where $fthr^k(i, j, i^{'}) = -1$ follows along similar lines.

Inequality class 8

Consider i, and j that is either independent of i or succeeds i in G. If $x_{ij}^k = 1$ for some k, then $V_{ij}^k \geq MinCT(i)$, by virtue of definition of $MinCT(i)$. If, however, j does not immediately succeed i on any machine, $\sum_{k\in\mathcal{K}} \sum_{i'=1,i'\neq j}^{N+1} x_{ii'}^k = 1$. This proves the validity of the following inequality.

$$\sum_{k\in\mathcal{K}} V_{ij}^k + MinCT(i)\Big(\sum_{k\in\mathcal{K}} \sum_{i'=1,i'\neq j}^{N+1} x_{ii'}^k\Big) \geq MinCT(i)$$

Inequality class 9

Consider distinct $i, j, i^{'} \in \mathcal{N}$ such that $(i, i^{'}) \in E$ and $(j, i^{'}) \in E$. For machine k, the following inequality holds:

$$C_{i'} - C_i \geq P_{i'}^{min} + p_j^k x_{ij}^k$$

Note that if $x_{ij}^k = 1$, earliest that $i^{'}$ can start is $C_i + p_j^k$, proving the validity of the inequality. In the computational tests, a generalized version of this inequality is used. The generalization is the case where $(i, h) \in E$, $(h, i^{'}) \in E$, $(j, j^{'}) \in E$ and $(j^{'}, i^{'}) \in E$. In this case, the inequality becomes $C_i^{'} - C_i \geq P^{min}(i^{'}) + Max\{p_j^k + P^{min}(j^{'}), P^{min}(h)\}x_{ij}^k$. Further generalization is possible by considering predecessors that are 3 or more levels above $i^{'}$, but computationally that did not provide significantly better improvement.