A novel integer programming formulation for scheduling with family setup times on a single machine to minimize maximum lateness

Oliver Hinder[a,b], Andrew Mason[b,*]

[a]*Management Science and Engineering, Stanford University*
[b]*Department of Engineering Science, University of Auckland, Private Bag 92019, Auckland, New Zealand*

Abstract

This paper focuses on the problem of scheduling n jobs with family setup times on a single machine with the objective of minimizing the maximum lateness. To solve this problem we develop a novel *ordered-batch* integer programming formulation. The formulation utilizes two properties of optimal solutions. Firstly, we observe that there is a restricted set of batches that we need to consider to find the optimal solution. Secondly, we know the order in which batches should be processed if they occur in an optimal solution.

Using branch and bound, our integer program finds optimal schedules for significantly larger problem instances than are reported in the literature. In contrast to existing algorithms, our formulation is strongest for problem instances with many families and large setup times. For example, we are able to find optimal solutions to problems with 1080 jobs and 270 families. We attribute this improvement to our linear programming relaxation being tighter than existing bounds for these cases.

To explain this performance, we analyze the theoretical tightness of this formulation. We show that if the number of jobs in each family is bounded then the gap between a heuristic rounding and the lower bound produced by the linear programming increases at most sub-linearly with the number of jobs. This improves on prior approximation algorithms that only guarantee optimality gaps that grow linearly with the number of jobs.

[*]Corresponding author

URL: `a.mason@auckland.ac.nz` (Andrew Mason), `http://des.auckland.ac.nz/mason` (Andrew Mason)

Keywords: Scheduling, family setup times, maximum lateness, single machine, integer programming

1. Introduction

It is well-known that in many factories it is quicker to process jobs in batches that share a setup time (Ahn and Hyun (1990), Jin et al. (2009), Potts and Wassenhove (1992)). Scheduling in these factories is complicated, because the goal is rarely to process orders as quickly as possible, but instead managers seek schedules that meet their order deadlines. In particular, making batches too large causes pressing orders to miss their due dates, while small batches result in many setups, causing delays in production. Good scheduling is therefore important to the efficient operation of a factory.

We consider the problem of scheduling jobs on a single machine to minimize maximum lateness. Each job belongs to a *family*. A *family setup* is required between any pair of consecutive jobs if they belong to different families. The time required for this setup depends on the family of the next job, and is assumed to be independent of the previous family. Any schedule can be decomposed into a sequence of *batches*, where a batch is a maximal set of jobs from one family that are processed consecutively following a family setup.

We are interested in solving this problem to optimality, and so this paper begins by reviewing previous exact algorithms used to tackle this NP-hard (Bruno and Downey, 1978) problem. One successful approach is based on a lower bound we term the *ordered-job* bound. Next, the paper discusses two key properties of optimal solutions known in the literature, and uses these to formulate a new 'ordered-batch' integer programming model. The integer requirements of this model are relaxed to form the *ordered-batch* linear program which can be solved to give our *ordered-batch* bound. To theoretically justify this relaxation, we show that the gap between the *ordered-batch* bound and the optimal schedule grows at most sub-linearly in the problem size. Following this, we empirically evaluate our integer program against other approaches. Our formulation outperforms a more traditional time-indexed integer program. Moreover, for problems with many families or large setup times our new *ordered-batch* bound is superior to the *ordered-job* bound, and our approach can solve huge instances of these problems.

2. Literature review

In general, machine scheduling problems with setups are difficult to solve; see Allahverdi et al. (1999), Allahverdi et al. (2008) and Allahverdi (2015) for a survey of such problems. The special class of family setups introduces additional 'batch' problem structure that can be exploited. (See Potts and Kovalyov (2000) for a review of batch scheduling.) However, such problems are challenging even in the simplest case of just a single machine. For example, Mason and Anderson (1991), Crauwels et al. (1998) and Dunstall et al. (2000) consider the family setup problem on a single machine with a weighted completion time objective, and develop enumerative approaches that can optimally solve problems with up to 70 jobs. Motivated by a problem arising in a Chinese factory, Jin et al. (2009) consider a single machine problem similar to ours but with sequence dependent family setups. Because their problems have several hundred jobs and many families, they are hard to solve optimally and so their work focuses on a Tabu search approach. Schutten et al. (1996) consider a similar single machine problem with release dates and due dates for which they develop a customised branch and bound algorithm; they solve problems with up to 50 jobs. In an example of more theoretical work, Grundel et al. (2013) also consider the single machine problem with family set-ups, but generalise the objective to one where jobs in a family share a common cost function that depends linearly on the job's completion time.

Our problem of minimizing maximum lateness with family setup times on a single-machine problem was shown to be NP-hard by Bruno and Downey (1978) and strongly NP-hard by Cheng et al. (2003). To solve problems where the number of families is small, Ghosh and Gupta (1997) wrote a dynamic programming algorithm which runs in $O(n^F)$ time, where n is the total number of jobs and F the total number of families. This allows the problem to be solved in polynomial time for a fixed number of families.

The problem becomes significantly harder as the number of families is increased, and so alternative approaches such as branch and bound have been proposed for this problem. Both Hariri and Potts (1997) and Baker and Magazine (2000) have presented branch and bound algorithms. The essential idea of Hariri and Potts (1997) is that a lower bound can be created by ignoring all the setups, except for those associated with the first job in each family; then the minimum maximum lateness schedule can be found by ordering the jobs by earliest due date. We will refer to this lower bound as the *ordered-job* bound and contrast it with our bound, which we will label the *ordered-batch* bound. The *ordered-job* bound can be computed quickly, but may not be tight, particularly if the problem consists of many families or setups are large. Baker and Magazine (2000) apply a similar approach but using partial schedules to create their lower bounds. While their lower bound may be slightly faster to compute, it is weaker than the Hariri and Potts *ordered-job* bound. These two papers used different computers and test examples, so it is difficult to make reliable comparisons of the solution times. Nonetheless, the results of Baker and Magazine (2000) do not give reason to believe that these two approaches have large differences in performance. Hariri and Potts create instances with up to 60 jobs, most of which they can solve within 100 seconds. These tests will form the basis of the evaluation of our own integer program in Section 9.

Much of the work in machine scheduling, such as that discussed above, focuses on the use of problem-specific lower bounds as part of customised branch and bound schemes. Our interest is in developing an effective integer programming approach. The use of mixed integer programming (MIP) for scheduling ('MIP-scheduling') has a number of advantages. Perhaps most importantly, the use of MIP models allows problems to be solved by commercial or open-source software packages, such as Gurobi (Gurobi, 2015), CPLEX (CPLEX, 2015), and CBC (COIN-OR CBC, 2015), without writing customised computer code. Furthermore, users of such models benefit from continual reductions in run times as a result of ongoing research and software improvements.

Since Wagner (1959) presented the first MIP-scheduling model in 1959, there have been an increasing number of MIP formulations for scheduling problems. Blazewicz et al. (1991) provide an early survey of MIP formulations for scheduling, with more recent surveys including Queyranne and Schulz (1994) and Keha et al. (2009). Li and Yang (2009) survey MIP models for parallel machines, while Pan (1997) present and compare five MIP models for job-shop and flow-shop problems. Crauwels et al. (2010) evaluate six MIP formulations for a single machine problem in which they seek to minimise the job tardiness sum. They find that problems with up to 40 or 50 jobs can be solved using the CPLEX solver (CPLEX, 2015), but larger problems require more specialised algorithms. They conclude that "systematic studies of [MIP model] effectiveness have been lacking" and "this subject warrants deeper investigation." Our work is a step in this direction.

If a problem becomes too large or complex to solve to optimality, then it is useful to have bounds on the quality of solutions found using heuristics.

The linear programming (LP) relaxations of MIP models can provide such bounds. However, it is well-known that the quality of the LP bound depends on the form and structure of the MIP formulation, with naive formulations often giving poor bounds. For example, Keha et al. (2009) present results for a single machine problem showing that problems that can be solved within 41 seconds using a 'time-indexed' formulation still have a bound gap of over 60% after one hour's computation using a 'completion time' formulation.

There is ongoing theoretical research into the properties of MIP-scheduling models and the development of stronger formulations through techniques such as cutting planes and valid inequalities. See, for example, the pioneering paper of Balas (1985) and the surveys by Queyranne (1993) and Queyranne and Schulz (1994). Some authors, such as Bigras et al. (2008) and van den Akker et al. (2000), have used Dantzig-Wolfe MIP reformulations to improve the strength of their models. MIP-models are also being used in the development of approximation algorithms; Savelsbergh et al. (2005), for example, evaluate different formulations of the single machine weighted-completion-time problem with release dates as part of LP-based heuristics and approximation algorithms, while Afrati et al. (1999) develop polynomial time approximation schemes for minimizing the average weighted completion time sum with release dates on multiple machines.

Our *ordered-batch* formulation differs from previous MIP scheduling work we have seen in that we first enumerate batches, and then use known optimality results to impose an ordering on these batches. This ordering is then exploited as a core component of the MIP model formulation. This enumeration of columns has similarities to column generation and Dantzig-Wolfe reformulation, and thus we can consider our approach to be part of this family of methods for developing stronger formulations. However, our formulation depends on the existence of a known optimal ordering for these batches, which is a feature not commonly found in column generation work.

3. Problem definition

We use the notation of Potts and Kovalyov (2000) and Webster and Baker (1995) to define a problem instance. Let there be F families, where each family $f \in \{1, ..., F\}$ has n_f jobs denoted $(1, f)$, $(2, f)$, ..., (n_f, f). Each job (k, f) has an associated due date $d_{(k,f)}$ and processing time $p_{(k,f)} > 0$. Each family f has an associated family setup of duration $s_f \geq 0$ which needs to be performed before processing some job (i, f) if the previous job is from another family (or job (i, f) is the very first job in the schedule). As we discuss shortly, jobs in the same family must be ordered by earliest due date (EDD), and all jobs in a family can be assumed to have distinct due dates. Therefore, we assume the indices of the jobs in each family are ordered by ascending due date, i.e. $d_{(k,f)} < d_{(k+1,f)} \ \forall \ f = 1, 2, ..., F, \ k = 1, 2, ..., n_f - 1$.

4. Preprocessing

Hariri and Potts (1997) show that jobs from the same family can often be merged to reduce the problem size. In particular, if $p_{(i+1,f)} \geq d_{(i+1,f)} - d_{(i,f)}$ then we can replace jobs (i, f) and $(i + 1, f)$ with a new single job with processing time $p_{(i,f)}+p_{(i+1,f)}$ and due date $\min \{d_{(i+1,f)}, p_{(i+1,f)} + d_{(i,f)}\}$. For the first job in each family it is only necessary that $p_{(2,f)} + s_f \geq d_{(2,f)} - d_{(1,f)}$. These two rules can be applied repeatedly until no further jobs can be merged. After this pre-processing step, all the jobs in a family have distinct due dates. We assume all our problem instances have been preprocessed in this way.

5. Optimal solution properties

There are several well known properties of optimal solutions that we use in our formulations.

Lemma 1. *(Monma and Potts, 1989) There exists some optimal solution in which the jobs from each family are processed in order of earliest due date (EDD).*

Thus, if we define a batch as a set of jobs from one family that are processed consecutively, then by Lemma 1 there exists an optimal solution that consists of a sequence of batches each of which contains jobs sequenced in EDD order. These EDD-batches will form the core of our integer programming formulations. Because our jobs are indexed in EDD order, we can use (i, j, f) to denote the batch in which jobs $(i, f), (i+1, f), ..., (j, f)$ from family f are processed consecutively following a setup of duration s_f. We enumerate all batches $(i, j, f), \ f = 1, 2, ..., F, \ i = 1, 2, ..., n_f, \ j = i, i + 1, ..., n_f$ to give a set of B batches $\{(i_b, j_b, f_b), b = 1, 2, ..., B\}$ where i_b, j_b and f_b define the first job index, last job index and family of batch b, respectively. Let $\mathcal{B}_f = \{b \in \{1, ..., B\} : f_b = f\}$ be the set of batches in family f.

Just as we ordered our jobs by earliest due date within each family, we assume the B batches are ordered by their 'effective due dates' in accordance with the following definition and lemma.

Definition 1. *The effective due date δ_b of a batch b defined by (i_b, j_b, f_b), is:*

$$\delta_b = \min_{i_b \leq k \leq j_b} \delta_{b,k}$$

where:

$$\delta_{b,k} = d_{(k,f_b)} + \sum_{q=k+1}^{j_b} p_{(q,f_b)}$$

Lemma 2. *(Webster and Baker, 1995) There exists an optimal schedule which consists of a sequence of batches ordered by effective due date.*

Thus, we assume our batches are numbered so that $\delta_1 \leq \delta_2 \leq ... \leq \delta_B$ (with ties broken arbitrarily). We use the following definition and corollary to define the lateness L_b of a batch as the maximum lateness of its jobs.

Definition 2. *The total processing time ρ_b of a batch b, including the setup time s_{f_b}, is $\rho_b = s_{f_b} + \sum_{k=i_b}^{j_b} p_{(k,f_b)}$.*

Corollary 1. *(Webster and Baker, 1995) If the processing of a batch b begins at time t_b, then the 'lateness' L_b of batch b, i.e. the maximum lateness over the jobs in batch b, is given by $L_b = t_b + \rho_b - \delta_b$*

6. Ordered Batch Integer Programming (OBIP) formulation

Assume we have enumerated all B batches and sorted them by their effective due dates. We can now form our 'ordered batch' integer program by introducing a binary variable x_b, $b = 1, 2, ..., B$ for each batch, where $x_b = 1$ if batch b is used in a solution and $x_b = 0$ otherwise. Because we know the order in which batches will be sequenced, the finish time of any batch b that is used in our solution can now be calculated as $\sum_{r=1}^{b} \rho_r x_r$. Our integer program uses this finish time and the effective due date δ_b of each batch b to impose a lower bound on another variable, $\hat{L}_{\max}$, which then gives the maximum lateness that we minimise. Our 'ordered batch IP' (OBIP) model comprises of the following expressions (1)-(6):

$$\text{OBIP: } \min \quad \hat{L}_{\max} \tag{1}$$

$$\text{s.t.} \quad \sum_{b \in \mathcal{B}_f : i_b \le k \le j_b} x_b = 1 \qquad \forall\ f \in \{1,2,...,F\}, k \in \{1,2,...,n_f\} \tag{2}$$

$$\sum_{r=1}^{b} \rho_r x_r - (\delta_{b-1}(1-x_b) + \delta_b x_b) \le \hat{L}_{\max} \qquad \forall\ b \in \{1,2,...,B\} \tag{3}$$

$$\hat{L}_{\max} \in \mathbb{R} \tag{4}$$

$$x_b \in \{0,1\} \qquad \forall\ b \in \{1,2,...,B\} \tag{5}$$

$$\sum_{r=1}^{b-1} \rho_r x_r + \rho_b - \delta_b \le \hat{L}_{\max} \qquad \forall\ b \in \{1,2,...,B\} : i_b = j_b = 1 \tag{6}$$

If we replace constraint (5) with $0 \le x_b \le 1$ then we call this the 'ordered batch LP' (OBLP).

Constraint (2) in our model ensures that each job occurs exactly once in the solution. As we will show shortly, constraints (3) guarantee that the maximum lateness, $\hat{L}_{\max}$, is no less than the latenesses of all batches $b \in \{1,2,...,B\} : x_b = 1$ used in the solution. Note that to define (3) for $b = 1$ we put $\delta_0 = \delta_1$. Constraint (5) ensures the solution corresponds to a selection of batches. Finally, as we will show, (6) is a valid inequality used to strengthen the formulation.

Next, we show that we have a correct formulation. Let $\mathbf{x} = (x_1, x_2, ..., x_B) \in G$ denote a set of x_b variables that satisfy constraints (2) and (5), where G is the set of all possible such solutions. Clearly, there is a one-to-one mapping between $\mathbf{x} \in G$ and a feasible schedule satisfying Lemmas 1 and 2, and thus our formulation allows an optimal schedule to be represented. We now show that the calculation of the objective $\hat{L}_{\max}$ in OBIP is correct for any $\mathbf{x} \in G$, and thus OBIP correctly solves our problem.

Definition 3. *Let $L_b(\mathbf{x})$ denote the lateness of batch b under some schedule defined by $\mathbf{x} \in G$, and let $L_b^{(3)}(\mathbf{x})$, $b = 1, 2, ..., B$ be the values of the left hand sides of constraint (3) evaluated at $\mathbf{x}$.*

Lemma 3. *Given some schedule defined by* $\mathbf{x} \in G$*, then for all* $b : x_b = 1$*,* $L_b^{(3)}(\mathbf{x}) = L_b(\mathbf{x})$*.*

Proof. This immediately follows from Corollary 1 by setting $t_b = \sum_{r=1}^{b-1} \rho_r x_r$. $\square$

We now prove the following lemma which is needed for the $x_1 = 0$ case we consider in Lemma 5.

Lemma 4. *Consider a batch* b *with* $i_b = j_b = 1$ *i.e. a batch consisting of the first job* $(1, f_b)$ *in family* f_b*. This batch* b *has the smallest index in family* f_b*; i.e.* $b = \min(\mathcal{B}_{f_b})$*.*

Proof. Consider any batch $a \in \mathcal{B}_{f_b} \setminus \{b\}$. Recall from Definition 1 that $\delta_b = d_{(1,f_b)}$ and $\delta_a = \min_{i_a \le k \le j_a} \delta_{a,k}$. If $k = 1$ then $j_a > 1$ and therefore $\delta_{a,1} \ge d_{(1,f_b)} + p_{(2,f_b)} > d_{(1,f_b)}$ because processing times are positive. If $k > 1$, $\delta_{a,k} \ge d_{(k,f_b)} > d_{(1,f_b)}$ because due dates are strictly increasing after preprocessing. Therefore $\delta_a > \delta_b$. $\square$

Lemma 5. *Given some schedule defined by* $\mathbf{x} \in G$*, then for all* $b : x_b = 0$*, then* $L_b^{(3)}(\mathbf{x}) \le L_{\max}(\mathbf{x})$ *holds.*

Proof. Let us assume that in solution $\mathbf{x}$, there is some batch b' that is the last batch to have been scheduled before batch b. That is we assume $\mathbf{x} = (x_1, x_2, ..., x_{b'}, 0, 0, ..., 0, x_b, x_{b+1}, ..., x_B)$ with $b' < b$, $x_{b'} = 1$, $x_b = 0$. (We will consider the case where no such b' exists shortly.) From Lemma 3, $L_{b'}(\mathbf{x}) = L_{b'}^{(3)}(\mathbf{x})$, hence:

$$L_{\max}(\mathbf{x}) \ge L_{b'}(\mathbf{x}) = L_{b'}^{(3)}(\mathbf{x}) = \sum_{r=1}^{b'} \rho_r x_r - \delta_{b'} \ge \sum_{r=1}^{b} \rho_r x_r - \delta_{b-1} = L_b^{(3)}(\mathbf{x})$$

where the inequality follows because $\delta_{b'} \le \delta_{b-1}$ and $\sum_{r=b'+1}^{b} \rho_r x_r = 0$.

Next consider the other case when there exists no such b'. In this case $L_b^{(3)}(\mathbf{x}) = -\delta_{b-1} \le -\delta_1$. Now by the previous lemma, batch $b = 1$ only contains one job so $-\delta_1 = -d_{(1,f_1)} \le L_{\max}(\mathbf{x})$. $\square$

Theorem 1. *For any fixed* $\mathbf{x} \in G$*, the minimal* $\hat{L}_{\max}$ *that satisfies constraints (3) correctly measures the maximum lateness of the associated schedule.*

Proof. Lemma 3 implies $\max_{b=1,\ldots,B:x_b=1} L_b^{(3)}(\mathbf{x}) = L_{\max}(\mathbf{x})$ and Lemma 5 implies $\max_{b=1,\ldots,B:x_b=0} L_b^{(3)}(\mathbf{x}) \leq L_{\max}(\mathbf{x})$. Hence $\max_{b=1,\ldots,B} L_b^{(3)}(\mathbf{x}) = L_{\max}(\mathbf{x})$. Since $\hat{L}_{\max}$ is minimal, $\hat{L}_{\max} = \max_{b=1,\ldots,B} L_b^{(3)}(\mathbf{x}) = L_{\max}(\mathbf{x})$. □

Lemma 6. *Equation* (6) *is a valid inequality, i.e.* $L_b^{(6)}(\mathbf{x}) \leq L_{\max}(\mathbf{x})$, *where* $L_b^{(6)}(\mathbf{x})$ *is the value of the left hand side of* (6) *evaluated at* $\mathbf{x}$.

Proof. A batch b with $i_b = j_b = 1$ consists of the single job $(1, f_b)$. Job $(1, f_b)$ is either completed in batch b at time $\sum_{r=1}^{b-1} \rho_b x_b + \rho_b$, or, by Lemma 4, in a later batch $b' > b$. Therefore, this job must have a lateness of at least $\sum_{r=1}^{b-1} \rho_b x_b + \rho_b - \delta_b = L_b^{(6)}(\mathbf{x})$. □

The next result justifies the OBIP formulation.

Corollary 2. *The optimal solution of OBIP finds a schedule that minimizes the maximum lateness.*

Proof. Observe that Theorem 1 and Lemma 6 imply that OBIP finds the optimal schedule $\mathbf{x}$ such that $\mathbf{x} \in G$. Furthermore, Lemma 1 and 2 imply that G contains an optimal schedule for the original problem. □

7. Worst case performance guarantees for the OBLP

In this section, we provide theoretical justification of our OBIP formulation, by arguing that the associated linear program relaxation, termed OBLP, is tight. We take an optimal fractional solution to the OBLP and randomly round it to produce a feasible schedule. The maximum lateness associated with this feasible schedule is, with high probability, not much worse than the linear program objective. More specifically, if the number of jobs in each family is bounded, the gap between the maximum lateness of the linear program solution and the rounded schedule is $O\left(\sqrt{n \log(n)}\right)$, where $n = \sum_{f=1}^{F} n_f$ is the total number of jobs. This is better than prior worst-case performance results that give optimality gaps of $O(n)$. We also show that if there is no bound on the number of jobs in each family, it is still possible to modify this technique to give LP rounding that is within $O\left(n^{2/3}\sqrt{\log(n)}\right)$ of the optimal maximum lateness.

7.1. Comparison with performance guarantees of previous algorithms

Prior work for worst-case performance guarantees makes the following statement:

$$L_{\max}^{H} \leq (1+\epsilon) L_{\max}^{*} \tag{7}$$

for some constant $\epsilon > 0$, where $L_{\max}^{H}$ is the maximum lateness of a heuristic schedule, and $L_{\max}^{*}$ is the maximum lateness of an optimal schedule. There have been several papers exploring approximation algorithms for this problem. For example, Zdrzałka (1995) gives an algorithm that runs in $O(n^2)$ with $\epsilon = 1/2$ and Hariri and Potts (1997) gives an algorithm that runs in $O(n \log(n))$ with $\epsilon = 2/3$. The result that gives the smallest ϵ is a polynomial time approximation scheme (PTAS) from Woeginger (1998). A PTAS is a family of algorithms that for a fixed ϵ gives a polynomial time algorithm. However, the required time grows exponentially in $1/\epsilon$. To acquire these results these papers assume that all due dates are non-positive. However, this assumption seems unnatural. Instead of examining this ratio, we will the consider the gap $L_{\max}^{H} - L_{\max}^{*}$. In Theorem 4 we show that if setup and processing times are bounded then the gap grows sub-linearly with the number of jobs:

$$L_{\max}^{H} \leq L_{\max}^{*} + O\left(n^{2/3}\sqrt{\log(n)}\right)$$

If we revert to the assumption that due dates are non-positive then $L_{\max}^{*} \geq np_{\text{av}}$, where p_{av} is the average job processing time. This implies that:

$$\frac{L_{\max}^{H}}{L_{\max}^{*}} \leq 1 + O\left(n^{-1/3}\sqrt{\log(n)}\right)$$

Hence

$$\frac{L_{\max}^{H}}{L_{\max}^{*}} \to 1 \text{ as } n \to \infty$$

which is equivalent to $\epsilon \to 0$ in Equation (7) as $n \to \infty$.

7.2. Approximation results with only two jobs in each family

We first analyse the case of two jobs in each family. To make the mathematics easier, this section uses a simplified version of the OBLP formulation. We remove constraints (3) and (6) and replace them with the following inequality:

$$\sum_{r=1}^{b} \rho_r x_r - \delta_b \leq \hat{L}_{\max} \qquad \forall\, b \in \{1, 2, ..., B\} \tag{8}$$

This gives a linear program that we term the Simplified OBLP formulation (SOBLP). It is trivial to see this makes the formulation weaker since $\delta_{b-1} \leq \delta_{b-1}(1-x_b)+\delta_b x_b \leq \delta_b$ for $0 \leq x_b \leq 1$. Therefore any tightness results that apply to SOBLP will also apply to OBLP.

Let z_f be one if the two jobs in family f are processed in the same batch or zero if they are processed in different batches. Formally, $z_f = x_{b_f^{\text{both}}}$, where in batch b_f^{both} both jobs in the family f are processed in the same batch; i.e. $b_f^{\text{both}} \in \mathcal{B}_f$, $i_{b_f^{\text{both}}} = 1$ and $j_{b_f^{\text{both}}} = 2$. We also let batches b_f^{first} and b_f^{last} be those batches in which the first job and last job are processed separately, i.e. $i_{b_f^{\text{first}}} = j_{b_f^{\text{first}}} = 1$ and $i_{b_f^{\text{last}}} = j_{b_f^{\text{last}}} = 2$. This allows us to re-write the SOBLP as:

$$\begin{aligned}
\min\ & \hat{L}_{\max} && && (9)\\
\text{s.t. } & \hat{L}_{\max} \geq L_b && \forall\, b \in \{1, 2, ..., B\} && (10)\\
& L_b = \sum_{f=1}^{F} \left(\gamma_{f,b} z_f + \eta_{f,b}(1 - z_f)\right) - \delta_b && \forall\, b \in \{1, 2, ..., B\} && (11)\\
& 0 \leq z_f \leq 1 && \forall\, f \in \{1, 2, ..., F\} && (12)
\end{aligned}$$

where we define $\gamma_{f,b}$ and $\eta_{f,b}$ as follows:

$$\begin{aligned}
\gamma_{f,b} &= I(b_f^{\text{both}} \leq b)\rho_{b^{\text{both}}}\\
\eta_{f,b} &= I(b_f^{\text{first}} \leq b)\rho_{b_f^{\text{first}}} + I(b_f^{\text{last}} \leq b)\rho_{b_f^{\text{last}}}
\end{aligned}$$

where $I(h) = 1$ if h is true, and $I(h) = 0$ otherwise.

Now consider some fractional solution to this new LP which we denote by z_f^{LP}, L_b^{LP}, $\hat{L}_{\max}^{\text{LP}}$. We apply the following rounding procedure to convert each fractional z_f^{LP} value into an integer equivalent denoted by Z_f as follows: with probability z_f^{LP} set $Z_f = 1$ otherwise set $Z_f = 0$. Observe that $\{Z_1, Z_2, ..., Z_F\}$ produces a feasible schedule since $Z_f \in \{0, 1\}$. Furthermore, this procedure creates random variables L_b^{R},

$$L_b^{\text{R}} = \sum_{f=1}^{F} \left(\left(\gamma_{f,b} Z_f + \eta_{f,b}(1 - Z_f)\right)\right) - \delta_b,$$

that describe the lateness of the batches, and a random variable $L^{\mathrm{R}}_{\max} = \max_{b\in\{1,\ldots,B\}} L^{\mathrm{R}}_b$ that is the maximum lateness of the rounded solution. We let $p_{\max}$ denote the maximum processing time of any job and $s_{\max}$ denote the maximum setup time of any family i.e. $p_{(i,f)} \leq p_{\max}$ and $s_f \leq s_{\max}$. Using this terminology the next lemma shows that the lateness of each batch in the rounded solution L^{R}_b is, with high probability, not much worse than L^{LP}_b:

Lemma 7. *If there are two jobs in each family, then with probability at most $\frac{1}{B^2}$:*

$$L^{\mathrm{R}}_b \geq L^{\mathrm{LP}}_b + 2(s_{\max} + p_{\max})\sqrt{F\log(B)}$$

Proof. Recall the concentration inequality of Hoeffding (1963) states that:

$$P\left(\sum_{t=1}^{T} W_t \geq E\left[\sum_{t=1}^{T} W_t\right] + \epsilon\right) \leq e^{-\frac{2\epsilon^2}{T(u-l)^2}}$$

where W_t are independent random variables with $l \leq W_t \leq u$.

Applying this to L^{R}_b by putting $L^{\mathrm{R}}_b = \sum_{f=1}^{F} W_{f,b} + \delta_b$ where $W_{f,b} = \gamma_{f,b} Z_f + \eta_{f,b}(1 - Z_f)$ gives:

$$P(L^{\mathrm{R}}_b \geq E[L^{\mathrm{R}}_b] + \epsilon) \leq e^{-\frac{2\epsilon^2}{F(2(s_{\max}+p_{\max}))^2}}$$

since $0 \leq W_{f,b} \leq 2(s_{\max} + p_{\max})$.

Setting $\epsilon = 2(s_{\max} + p_{\max})\sqrt{F\log(B)}$ gives

$$P(L^{\mathrm{R}}_b \geq E[L^{\mathrm{R}}_b] + 2(s_{\max} + p_{\max})\sqrt{F\log(B)}) \leq e^{-2\log(B)} = \frac{1}{B^2}$$

The result follows because $E[Z_f] = z^{\mathrm{LP}}_f$ and therefore $E[L^{\mathrm{R}}_b] = L^{\mathrm{LP}}_b$. □

Theorem 2. *If there are two jobs in each family, then with probability at least $1 - \frac{1}{3F}$:*

$$L^{\mathrm{R}}_{\max} \leq L^{\mathrm{LP}}_{\max} + 2(s_{\max} + p_{\max})\sqrt{F\log(3F)}$$

Proof. Because $L^{\mathrm{LP}}_b \leq L^{\mathrm{LP}}_{\max}\ \forall\, b = 1, 2, ..., B$, Lemma 7 gives $P(L^{\mathrm{R}}_b \geq \kappa) \leq \frac{1}{B^2}$ where we have $\kappa = L^{\mathrm{LP}}_{\max} + 2(s_{\max} + p_{\max})\sqrt{F\log(B)}$. Considering all B batches, and using the union equality, we have

$$P\left(L^{\mathrm{R}}_1 \geq \kappa \cup L^{\mathrm{R}}_2 \geq \kappa \cup ... \cup L^{\mathrm{R}}_B \geq \kappa\right) \leq B\frac{1}{B^2} = \frac{1}{B}$$

The result then follows by noting $B = 3F$. □

Since $L^{*}_{\max} \leq L^{\mathrm{R}}_{\max}$, Theorem 2 automatically makes a statement about the tightness of the OBLP.

7.3. Rounding fractional solutions with two or more jobs in each family

The rounding and analysis becomes more challenging when there are more than two jobs. As before, we want to round the fractional solution associated with each family independently so that we can once again apply the Hoeffding concentration inequality to the sum of random variables. To do this we introduce the concept of an f-schedule which is a feasible set of batches for a particular family.

Definition 4. *The set of batches $\pi \subseteq \mathcal{B}_f$ is an f-schedule if for each $k = 1, ..., n_f$ the job (k, f) is contained in exactly one batch $b \in \pi$.*

Consider the previous definition. If we choose one f-schedule π_f for each family f then the set of batches $\pi_1 \cup \pi_2 \cup \cdots \cup \pi_F$ (ordered by earliest effective due date) yields a feasible schedule. Next, consider the following probability distribution over f-schedules for each family f:

$$P_{(\Omega_f, \omega_f)}(\Pi_f = \pi) = \begin{cases} \omega_{f,\pi} & \text{if } \pi \in \Omega_f \\ 0 & \text{otherwise} \end{cases}$$

where Ω_f is a set of f-schedules for family f, Π_f is a random variables giving the f-schedule chosen for family f, and the vector ω_f has elements $\omega_{f,\pi} > 0$, $\pi \in \Omega_f$ each of which give the probability that f-schedule $\pi \in \Omega_f$ is chosen. This distribution allows us to define a random variable:

$$X_b^{\mathrm{R}} = I(b \in \Pi_f)$$

where $X_b^{\mathrm{R}} = 1$ if batch $b \in \mathcal{B}_f$ is in the randomly chosen schedule, and $X_b^{\mathrm{R}} = 0$ otherwise. We can then, based on the SOBLP, define a random variable for the maximum lateness of each batch in this random rounding:

$$L_b^{\mathrm{R}} = \sum_{r=1}^{b} \rho_r X_r^{\mathrm{R}} - \delta_b$$

Ideally, we would want the rounding to be similar, in expectation, to the original SOBLP solution i.e. $E[L_b^{\mathrm{R}}] = L_b^{\mathrm{LP}}$. Definition 5 gives a condition when this is true.

Definition 5. (Ω_f, ω_f) *covers* $(\mathbf{x}, f)$ *if:*

$$\sum_{\pi \in \Omega_f} I(b \in \pi)\omega_{f,\pi} = x_b \ \ \forall \ b \in \mathcal{B}_f$$

If (Ω_f, ω_f) covers $(\mathbf{x}, f)$ for all $f \in F$ then $E[X_b^{\text{R}}] = x_b$ hence $E[L_b^{\text{R}}] = L_b^{\text{LP}}$. This property is critical to the proof of Theorem 3.

Lemma 8. *For any family f and solution $\mathbf{x}$ to the SOBLP we can construct an (Ω_f, ω_f) that covers $(\mathbf{x}, f)$ in $O(|\mathcal{B}_f|^2 n_f)$ time.*

We leave the proof of this lemma to Section A.1. Here we sketch our algorithm (Algorithm A.1) for constructing an (Ω_f, ω_f) that covers $(\mathbf{x}, f)$.

Initially we start with $\Omega_f = \emptyset$ and $\hat{\mathbf{x}} = \mathbf{x}$.

At each iteration we find an f-schedule π such that $b \in \pi$ only if $\hat{x}_b > 0$. We add this f-schedule π to Ω_f with $\omega_{f,\pi} = \min_{b \in \pi} \hat{x}_b$. We then subtract this new f-schedule from the solution $\hat{\mathbf{x}}$ by $\hat{x}_b \leftarrow \hat{x}_b - I(b \in \pi)\omega_{f,\pi}$. We repeat this process until $\hat{\mathbf{x}} = 0$ at which point we have an (Ω_f, ω_f) that covers $(\mathbf{x}, f)$.

We next present several algorithms and associated results based on the (Ω_f, ω_f) we construct.

7.4. Approximation results when the length of each family is bounded

In Appendix A.2 we present a rounding algorithm, Algorithm A.3, which uses (Ω_f, ω_f) to create an integer solution. We now give approximation results for this algorithm in terms of the processing 'length' τ_f of a family, where τ_f is the maximum time a family could take to be processed in a schedule:

$$\tau_f = \sum_{k=1}^{n_f} \left(s_f + p_{(k,f)}\right)$$

Theorem 3. *With probability at least $1 - \frac{1}{B}$:*

$$L_{\max}^{\text{R}} \leq L_{\max}^{\text{LP}} + \sqrt{\log(B) \sum_{f=1}^{F} \tau_f^2}$$

where $L_{\max}^{\text{R}}$ is the maximum lateness of the schedule generated by Algorithm A.3.

We leave this proof to Appendix A.2, but note that it is similar to the proof of Lemma 7 and Theorem 2.

If we put $\tau_{\max} = \max_{f=1,\ldots,F} \tau_f$, then we have $\sum_{f=1}^{F} \tau_f^2 \leq \tau_{\max} \sum_{f=1}^{F} \tau_f \leq \tau_{\max}(p_{\max} + s_{\max})n$. Noting also that $B \leq n^2 \Rightarrow \log(B) \leq 2\log(n)$, we get

$$L_{\max}^{\mathrm{R}} \leq L_{\max}^{\mathrm{LP}} + \sqrt{2\log(n)\tau_{\max}(p_{\max} + s_{\max})n}$$

If there at most m jobs in each family then

$$L_{\max}^{\mathrm{R}} \leq L_{\max}^{\mathrm{LP}} + (p_{\max} + s_{\max})\sqrt{2\log(n)nm} \tag{13}$$

These results show that when there are many job families, each with a small number of jobs, the SOBLP relaxation will be relatively tight; i.e. the bound gap will grow with $O(\sqrt{n\log(n)})$ for constant m.

7.5. Approximation result with an arbitrary number of jobs in each family

We would like the bound gap to grow sub-linearly with n without requiring any assumptions on the size of the families. In Appendix 4 we present another rounding approach, Algorithm A.4, that generates solutions with this property. Theorem 4 shows that the difference between the optimal schedule to the original problem $L_{\max}^{*}$ and the heuristic maximum lateness $L'^{\mathrm{R}}_{\max}$ generated by Algorithm A.4 grows sub-linearly with the number of jobs, i.e. $O\left(n^{2/3}\sqrt{\log(n)}\right)$. Note that unlike our previous results, this result depends on the optimal integer objective value $L_{\max}^{*}$ and therefore does not make a statement about the strength of the lower bound generated by the OBLP.

Theorem 4. *With probability at least $1 - \frac{1}{B}$:*

$$L'^{\mathrm{R}}_{\max} \leq L_{\max}^{*} + n^{2/3}(1 + \sqrt{2\log(n)})(s_{\max} + p_{\max})$$

where $L'^{\mathrm{R}}_{\max}$ is the maximum lateness of the schedule generated by Algorithm A.4.

The proof of this result is given in Section A.3; here we sketch the general idea. The previous result, Theorem 3, requires the size of the families, i.e. m, to grow very slowly. To avoid the problem of large families we create a new artificial problem, where we split up families. In particular, we split the families evenly (maintaining the EDD order of jobs) until $n_f \leq n^{1/3}$. This process requires at most $n^{2/3}$ additional families. This makes the gap between the optimal schedule to the artificial problem, $L'^{*}_{\max}$, and original problem $L_{\max}^{*}$ at most $s_{max}n^{2/3}$. We can use Theorem 3 to bound the gap between the artificial optimum schedule $L'^{*}_{\max}$ and the random rounding of the artificial SOBLP optimal solution $L'^{\mathrm{R}}_{\max}$. Adding the two gaps together yields Theorem 4.

In the next sections, we focus on demonstrating the practical value of our new model in solving benchmark problems.

8. A Compact Ordered Batch Integer Programming (COBIP) formulation

Consider again our OBIP model. The number of constraints given by (3) increases linearly with B, and thus (3) typically creates many rows in OBIP which can make the problem difficult to solve. We show next that we can consider a 'compact ordered batch IP' (COBIP) that typically has fewer rows.

Suppose some consecutive batches $a, a+1, ..., c-1, c$ in our batch ordering belong to the same family, i.e. $f_a = f_{a+1} = ... = f_{c-1} = f_c$. Including more than one of these batches in the solution would result in an unnecessary setup, and so an optimal solution cannot contain more than one of these batches. This observation allows us to replace (3) by the following constraints.

$$\sum_{r=a}^{c} x_r \le 1 \qquad \forall\ a, c : 1 \le a < c \le B : f_a = f_{a+1} = ... = f_c \quad (14)$$

$$\sum_{r=1}^{c} \rho_r x_r + \sum_{r=a}^{c} (\delta_{a-1} - \delta_r) x_r - \delta_{a-1} \le \hat{L}_{\max}$$
$$\forall\ a, c : 1 \le a \le c \le B : f_a = f_{a+1} = ... = f_c \quad (15)$$

Constraint (14) enforces our observation above. Consider next constraint (15). To show this is valid in this new formulation, we first note that for any solution with $x_{a+1} = x_{a+2} = ... = x_c = 0$, constraint (15) is equivalent to (3) for $b = a$, and constraint (14) is redundant, giving us the original OBIP formulation. Assuming our solution satisfies (14), the only other possibility is one of x_{a+1}, x_{a+2}, ..., x_c has value 1. Letting $x_b = 1$ denote this non-zero variable, then in this case the left hand side of constraint (15) simplifies to $\sum_{r=1}^{b} \rho_r x_r - \delta_b$, which is the lateness $L_b(\mathbf{x})$ of batch b as required for a correct formulation.

We do not need all the constraints given by (15), but instead note that (15) simultaneously specifies a correct lower bound on $\hat{L}_{\max}$ for all batches a,

$a + 1$, ..., c. Thus, we can minimise the number of constraints in COBIP by always choosing a and c so that the number of consecutive batches $(c - a)$ is maximal. Constructing our new constraints in this way results in $c - a + 1$ rows from OBIP being replaced by 2 rows in COBIP for each $(a, c) : \ a > c$ pair. Note that we do not include constraint (14) for any (a, c) pair with $a = c$, because the resulting constraint is always satisfied. Furthermore, we observe that (2) dominates (14) if batches a, $a + 1$, ..., c share at least one job in common, and so (14) will often be removed by the solver during pre-processing.

If we included all these constraints given by (15) this formulation would be tighter because (3) is a special case of (15). However, by always choosing a and c so that the number of consecutive batches $(c - a)$ is maximal, the new formulation may be either weaker or stronger. However, as we see in Section 9.2 changing from OBIP to COBIP had a negligible impact on the root node linear programming objective value.

9. Computational results

In this section we evaluate the performance of our integer programming models. As we discuss shortly, our compact *ordered-batch* approach (COBIP) outperformed both our original *ordered-batch* approach (OBIP) and a time-indexed formulation (TIBIP) we tested (see Section 9.2), and so our focus is on a detailed evaluation of COBIP's performance. We do this primarily by comparing it against the results reported by Hariri and Potts for their branch and bound algorithm. (We would expect comparisons with Baker and Magazine's algorithm to yield similar conclusions because of the similarity of the two approaches.) We find that we are able to solve significantly larger problems, and that our integer program is particularly effective when there are many families or large setups. However, we recognise that hardware improvements make time-based comparisons difficult, and so we also compare the underlying *ordered-job* bounds produced using the Hariri and Potts approach with our *ordered-batch* bounds. We show that our bounds are usually tighter. We then briefly consider how our integer program reacts to variation in the due date spread. We find that our *ordered-batch* bound appears to be consistently tight, whereas the *ordered-job* bound does poorly for small due date spreads but is exceptionally tight when the due date spread becomes very large.

9.1. Methods

Our experimental testing follows the methodology used by Hariri and Potts. We implement the following approach, described in Hariri and Potts (1997), to generate 3375 random problem instances to solve. (Baker and Magazine (2000) also present test problems, but these are more restricted in that they assume the same setup time for all families. However, Baker and Magazine raise some issues with the Hariri and Potts data set which we address later in this section.) Each problem generated belongs to one of 45 different classes, where each class has a given number of families $F \in \{2, 4, 6, 8, 10\}$, a given total number of jobs $n \in \{60, 90, 120\}$, and either 'small', 'medium' or 'large' setup times. Although Hariri and Potts tested with problems with up to $n = 60$ jobs, we are able to easily solve larger problems, and so have doubled the maximum number of jobs we consider.

We generated 75 problem instances in each class as follows. The family setup times s_f were generated from one of three different uniform integer distributions: $[1, 20]$ (Small), $[1, 100]$ (Medium), and $[101, 200]$ (Large). Common random numbers were used to ensure problems differing only in their setup times were otherwise identical. For a problem with n jobs, the number of jobs in each family, n_f, was either $\lceil n/F \rceil$ or $\lfloor n/F \rfloor$ such that $\sum_{f=1}^{F} n_f = n$. Processing times of jobs were generated from a uniform distribution of random integers in the range $[1, 100]$. The 75 test instances in each class comprised 5 test problems for each of 15 different due date 'spreads', calculated as follows. The due dates for each job is were sampled from a uniform integer distribution in the range $[0, rP]$, where $P = \sum_{f=1}^{F} \sum_{k=1}^{n_f} p_{(k,f)}$ is the total processing time of the jobs, and r gives a specified due date spread. The 15 r values used were given by calculating $r = d - c$ for each $c \in \{0.0, 0.2, 0.4, 0.6, 0.8\}$, $d \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$, $c < d$. (Baker and Magazine (2000) note that this calculation is biased towards small r values in that a particular value r, $r \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$, is considered $6 - 5r$ times. This somewhat unusual choice allows comparison with Hariri and Potts's work where they use a range $[cP, dP]$ for the due dates, but incorporates the observation of Baker and Magazine (2000) that the absolute due dates do not matter, but only their range.)

The 3375 test problems (75 instances in each of 45 classes) were solved by Gurobi (Gurobi, 2015) to optimality (i.e. with 'MIPGap=0') on an Intel Xeon W3530 2.80GHz processor with 12.0 GB of RAM, using a time limit of 100 seconds for each instance. Solve times and the number of branch and bound nodes were recorded for each instance (with values of 100s being recorded if Gurobi reached this time limit), and averages computed using these values.

9.2. Comparing different integer programming formulations

We started our experiments by comparing the compact *ordered-batch* formulation (COBIP) with the original (OBIP) formulation. Tests using the 3375 test instances showed that the total number of constraints given by (14) and (15) in our compact formulation was 16.2% less, on average, than the number given by (3) in the original formulation. Changing from OBIP to COBIP had a negligible impact on the root node linear programming objective value (giving an increase of just 0.1% on average), with over 96% of instances having a COBIP root node objective value within $\pm1\%$ of the OBIP value, and the rest being within approximately $\pm25\%$. There was no obvious trend in this root node difference with problem difficulty. Despite the general similarity in root node values, the compact formulation reduced the average solution time by 14%, a result which we attribute primarily to there being fewer constraints in COBIP.

Having determined that COBIP outperformed OBIP, we next compared this compact *ordered-batch* formulation with a more traditional time-indexed (TIBIP) formulation. Such formulations can often outperform alternative approaches (e.g. see Keha et al. (2009)). For the sake of completeness, we develop such a model here so that we can compare its performance with our *ordered-batch* formulation. The full model is detailed in Appendix B.

The number of constraints in a time-indexed formulation can be very large, and so for our initial tests, we artificially reduced the number of time steps by a factor of 20 by scaling the job durations to be integer values between 1 and 5 inclusive, instead of between 1 and 100 as we used in our other tests. The set-up times and due dates were similarly scaled. We also considered just the smaller problem instances with $n = 60$. As before, we used a time limit of 100s.

Over the 1125 test problems, we found that the TIBIP run times for these simplified instances were on average almost 2000 times longer than those of COBIP, and never less than 27 times longer. (To avoid division by small values, this data was calculated using the 639 problem instances which had COBIP solve times greater than 0.01s.) This run time difference can perhaps be explained by TIBIP having on average more than 270 times as many columns and 8 times as many rows as COBIP. However, we also found that TIBIP had a root node linear programming relaxation value that, on average, was only 61% of the corresponding COBIP value, and thus the COBIP model is inherently stronger on average. (This is not true for all problem instances, with the TIBIP root node value varying from just 12% of the COBIP value to up to 20% larger than the COBIP value.) Given these poor initial results on these simplified problems, we chose to focus on the COBIP formulation for the rest of our analysis.

9.3. Comparing COBIP with prior work

Table 1 below shows the experimental results from our COBIP testing broken down by problem class. (This table contains similar information to that presented in Table 1 of Hariri and Potts (1997).) Each row is identified by a number of jobs n and number of families F, and gives average results for the 75 problems generated and solved for each of the small (S), medium (M) and large (L) setup sizes. The $\bar{n}'$ column shows the average number of jobs left in the problem after merging jobs during pre-processing (see Section 4), while the 'rows' and 'cols' columns show the average number of rows and columns in our COBIP formulation. (These values do not vary with the size of setups, and so these are averages over the 75 problems generated for each (n, F) pair.) The detailed results given for each setup size include the average solution time, the number of nodes required by Gurobi during the branch and bound process, and the number of problems that could not be solved to proven optimality within the 100s time limit.

Table 1: Computational results for Hariri and Potts test problems

Problem Size			COBIP Formulation Size		Average computation time (secs.)			Average number of branch-and-bound nodes			Unsolved problems (out of 75)		
n	F	$\bar{n}'$	rows	cols	Sm	Med	Lrg	Sm	Med	Lrg	Sm	Med	Lrg
60	2	9.5	41.6	51.9	0.11	0.09	0.04	1	26	5	0	0	0
	4	24.2	133.2	118.7	0.43	0.49	0.32	176	233	94	0	0	0
	6	32.4	155.8	125.8	0.39	0.37	0.19	118	221	70	0	0	0
	8	38.4	167.1	126.4	0.35	0.25	0.16	261	136	45	0	0	0
	10	42.2	169.7	121.6	0.18	0.19	0.11	52	99	35	0	0	0
90	2	12.8	76.9	100.8	0.60	1.05	0.34	170	110	19	0	0	0
	4	34.5	250.5	237.3	4.66	6.93	2.96	1251	2287	949	0	1	0
	6	48.7	308.8	269.7	3.25	4.90	1.52	1262	1744	570	0	0	0
	8	57.4	318.9	263.0	3.17	2.84	1.19	1681	2086	530	0	0	0
	10	63.6	320.5	253.8	1.51	1.85	0.82	554	1244	390	0	0	0
120	2	16.1	124.1	168.4	2.89	5.75	2.65	249	321	215	0	1	0
	4	46.2	428.0	420.3	17.22	19.55	14.64	1859	1374	1811	9	12	5
	6	64.0	500.9	454.3	22.11	20.43	12.78	3745	3014	3349	9	9	2
	8	76.7	522.4	452.1	18.87	17.98	9.30	4402	5291	4383	9	4	0
	10	84.4	512.8	429.0	10.76	10.49	4.33	3935	4162	2338	0	1	0

Note: Sm= small setup times; Med = medium setup times; Lrg = large setup times.

We note that, as expected, problems with more jobs (after pre-processing) result in larger COBIP formulations and, other things being equal, take longer to solve. The integer programs are not unduly large by modern standards, and most are solved to provable optimality within 100 seconds.

Comparing our results with those reported by Hariri and Potts, we note that we can solve all the problems with 60 jobs (the largest size they considered) within 100 seconds. In contrast, Hariri and Potts report that their algorithm failed to solve 114 instances of their 60-job problems. In particular, Hariri and Potts report that their hardest problem class – 60 jobs, 10 families and large setups – took an average of 34.38 seconds, with 20 problems being stopped after the 100 second time limit. We are able to solve all problems of this class within the time limit with an average solve time of 0.11 seconds. While much of this speedup can be attributed to improvements in processing power, we argue below that the stronger bounds we use are also a significant contributing factor.

We find that problems with a small or large number of families seem to be easiest to solve using our formulation. We note that preprocessing reduces the number of jobs in the problem to a greater extent when the number of families is small, and thus we are not surprised that problems with a small number of families are easier to solve. This helps explains why both Baker and Magazine (2000) and Hariri and Potts (1997) also report good performance on problems with few families. However, our solution times are also smaller for larger numbers of families. This is in contrast to the results given by Hariri and Potts and Baker and Magazine who reported that their runs times kept increasing as F increased.

To help explain why the performance of our new algorithm is less dependent on F, consider Table 2 which compares the performance of our bound with the Hariri and Potts bound, as follows. For each problem instance, we recorded the best integer solution objective value $L^*_{\max}$ found by Gurobi within 100 seconds, the *ordered-job* bound $\tilde{L}_{\max}$ produced using the procedure outlined by Hariri and Potts (1997), and the *ordered-batch* bound $\hat{L}_{\max}$ generated by solving the linear programming relaxation of COBIP. These were used to calculate the *ordered-job* bound gap $g_{\mathrm{OJ}} = L^*_{\max} - \tilde{L}_{\max}$ and the *ordered-batch* bound gap $g_{\mathrm{OB}} = L^*_{\max} - \hat{L}_{\max}$ for each problem instance. These values were then averaged over the 75 instances of each problem class to give average values $\bar{g}_{\mathrm{OJ}}$ and $\bar{g}_{\mathrm{OB}}$. (Note that we have reported $\bar{L}^*_{\max}$ for the sake of completeness, but we observe that $L^*_{\max}$ for any instance can be changed arbitrarily by simply adding a constant to all the due dates. Furthermore, because $L^*_{\max}$ is essentially arbitrary, and indeed can be positive, negative or zero, we have calculated absolute bound gaps rather than the relative bound gaps sometimes given.) The average bound gaps were then used to calculate the average bound gap ratio $\phi = \bar{g}_{\mathrm{OJ}}/\bar{g}_{\mathrm{OB}}$. A value of $\phi > 1$ (or $\phi < 1$) indicates that our new *ordered-batch* bound $\hat{L}_{\max}$ gives a better (or worse) gap than the *ordered-job* bound $\tilde{L}_{\max}$.

Table 2: Average objective values and bound gaps for Hariri and Potts test problems

		Average objective value $\bar{L}^*_{\max}$			Average ordered-job bound gap $\bar{g}_{\text{OJ}}$			Average ordered-batch bound gap $\bar{g}_{\text{OB}}$			Average bound gap ratio $\phi = \bar{g}_{\text{OJ}}/\bar{g}_{\text{OB}}$		
n	F	Sm	Med	Lrg	Sm	Med	Lrg	Sm	Med	Lrg	Sm	Med	Lrg
60	2	1700.7	1853.9	2085.1	12.6	42.1	85.7	13.9	31.5	45.0	0.91	1.33	1.90
	4	1716.3	1972.5	2504.4	27.9	96.5	230.6	14.6	34.1	63.3	1.90	2.83	3.64
	6	1743.9	2078.2	2846.9	36.8	125.7	277.1	13.2	33.8	63.4	2.80	3.72	4.37
	8	1804.8	2197.2	3175.1	46.0	133.8	310.3	11.5	28.1	56.7	4.00	4.77	5.47
	10	1814.8	2328.2	3518.1	41.7	147.9	335.6	10.8	29.7	57.4	3.87	4.99	5.84
90	2	2531.4	2635.1	2933.6	13.7	58.4	113.0	29.0	39.8	58.9	0.47	1.47	1.92
	4	2547.2	2798.4	3361.3	32.3	124.4	285.6	18.6	47.0	87.3	1.73	2.65	3.27
	6	2597.2	2941.0	3809.0	42.7	155.1	384.5	15.8	44.0	92.1	2.70	3.52	4.17
	8	2628.0	3096.7	4083.9	54.9	194.9	441.2	14.7	43.2	83.1	3.74	4.51	5.31
	10	2615.9	3213.4	4461.3	53.8	202.0	468.9	12.4	35.5	78.6	4.32	5.70	5.97
120	2	3337.9	3482.1	3749.0	16.3	65.5	133.5	26.7	46.8	76.8	0.61	1.40	1.74
	4	3382.4	3670.4	4243.6	38.7	150.8	331.0	22.4	60.6	103.1	1.73	2.49	3.21
	6	3474.9	3774.9	4666.3	55.3	183.5	454.1	20.8	53.0	110.2	2.66	3.46	4.12
	8	3466.4	3901.1	5063.1	64.5	221.6	559.3	17.2	48.8	107.7	3.75	4.54	5.19
	10	3434.6	3994.3	5427.0	68.9	231.9	602.3	14.9	43.5	95.5	4.63	5.33	6.31

Note: Sm= small setup times; Med = medium setup times; Lrg = large setup times.

This table shows different trends for the two bounds as the number of families F is changed. We see that the *ordered-job* gap $\bar{g}_{\text{OJ}}$ typically increases as F increases, but our new *ordered-batch* gap $\bar{g}_{\text{OB}}$ shows little dependency on F. We also observe that although both bounds get worse as setup times increase, the relative improvement ϕ offered by our new bound increases with increasing setup times. This results in our new bound being much tighter for large F and large setups, with the resulting gap being up to $\phi = 6.31$ times smaller on average (see $n = 120, F = 10$, Large setups). In fact, the *ordered-batch* bound seems stronger in all cases except when there are small setup times and only two families, in which case the *ordered-job* bound does better. This is not unexpected as the *ordered-job* bound will give the optimal solution if the number of families is one or the setup times are zero. Therefore as we converge on this simple case we expect the *ordered-job* bound to become tighter.

The difference in the strength of the bounds is reflected in the node counts shown in Table 1. For example, Hariri and Potts report an average node count of 17463 for their 60-job instances. In contrast, our integer program needs to explore, on average, only 105 nodes for our 60-job instances. Our tighter *ordered-batch* bound, combined with Gurobi's efficient heuristics, enables us to explore less of the branch and bound tree to find the optimal solution and prove its optimality.

Consider the speed at which the bounds can be computed. The *ordered-batch* bound requires solving a linear program with $O(\sum_{f=1}^{F} n_f^2)$ rows and columns. Consequently, this relaxation will typically be slower to evaluate than both Baker and Magazine's bound, which only takes $O(n)$ calculations, and the more expensive Hariri and Potts *ordered-job* bound that takes $O(n \log(n))$. In particular, the *ordered-batch* bound will be slow to evaluate when there is a large number of jobs and a small number of families. So although our *ordered-batch* bound provides a tighter bound in many cases, it may be more expensive to compute.

As we commented earlier, the random instance generation approach of Hariri and Potts that we used above is biased towards small due date spreads. (We note that it gave an average r of just 0.466.) However, Baker and Magazine (2000) suggest that problems become hardest as r increases to between 10 and 20. Thus, to test our approach on these instances, we created a second set of test problems using the following method, as detailed by Baker and Magazine. Processing times of jobs are generated from the uniform integer distribution $[1, 99]$ and due dates are generated from the uniform distribution $[0, rP]$. Unlike Hariri and Potts, instead of grouping these problem sizes by setup times, Baker and Magazine fix the setup time for each family f such that $s_f = 100\ \forall f$ and vary the due date spread $r \in \{0.4, 0.6, 0.8, 1.0, 2.0, 4.0, 8.0, 16.0\}$. We initially considered the three problem sizes used by Baker and Magazine, namely, $F = 3, n_f = 12\ \forall f$; $F = 4, n_f = 8\ \forall f$; and $F = 5, n_f = 6\ \forall f$. As they did, we solved 25 random instances for each combination of F, n_f and r, but found that these all gave average solution times of less than 0.25 seconds, which we considered too small for useful analysis. We instead considered a larger $F = 10, n_f = 10\ \forall f$ problem. Solving 25 random instances of this problem for each value of r generated the solution times shown in Table 3. This table also shows the two average bounds gaps $\bar{g}_{\text{OJ}}$ and $\bar{g}_{\text{OB}}$ and their ratios for these problems.

Table 3: Experimental results when the due date spread r is varied for 25 random instances of a $F = 10, n_f = 10$ problem with $s_f = 100$

Due rate range r	0.4	0.6	0.8	1.0	2.0	4.0	8.0	16.0
Solution time (seconds)	0.695	1.999	4.626	13.865	12.975	0.367	0.095	0.070
Mean objective value $\bar{L}^*_{\max}$	4350.5	3630.3	2856.8	2274.1	544.7	36.7	-253.9	-660.3
Mean *ordered-job* bound gap $\bar{g}_{\mathrm{OJ}}$	344.96	592.52	766.96	962.08	129.84	0.36	0.20	0.00
Mean *ordered-batch* bound gap $\bar{g}_{\mathrm{OB}}$	75.36	85.09	96.79	113.58	83.30	59.48	24.51	12.21
Bound gap ratio $\phi = \bar{g}_{\mathrm{OJ}}/\bar{g}_{\mathrm{OB}}$	4.58	6.96	7.92	8.47	1.56	0.01	0.01	0.00

Table 3 shows that the most difficult due date spread for our integer program occurs when $r = 1.0$ or $r = 2.0$. (This is smaller than the $r = 10$ or $r = 20$ values Baker and Magazine suggest are most difficult, but see our comments below.) We believe small due date spreads are easier to solve because the pre-processing (see Section 4) aggregates more jobs when $d_{(i+1,f)} - d_{(i,f)}$ is small. On the other hand, when the due date spread becomes very large the problem becomes easy (and the bound gap small) because (i) the variation in due date values exceeds the variation in completion times, (ii) $L^*_{\max}$ only depends on the completion time of the job with the smallest due date, and (iii) any solution starting with this job will be optimal. Indeed, we see that the average absolute gap for the *ordered-batch* bound is highest when $r = 1.0$ at 114, and that this bound gap shrinks for both low and high values of r. This explains much of the variation in run times seen for these problems.

Table 3 shows that the *ordered-job* bound performs much better than our *ordered-batch* bound for large r. However, as discussed, these problems are quickly solved using our IP. For the problems we find hardest (i.e. $r = 1.0$), our bound is much tighter, suggesting our approach would be superior.

9.4. A large scale experiment

We conducted one final experiment. The purpose of this experiment was to demonstrate that our integer program is capable of solving huge instances when there are many families but few jobs in each family. Processing times of jobs are generated from the uniform integer distribution $[1, 100]$ and due dates are generated from the uniform distribution $[0, rP]$ with $r = 1.0$ (being the most difficult case in Table 3). We fix the setup time to be $s_f = 100$. Three different problem sizes are generated, all with $n = 1080$ jobs: $F = 540, n_f = 2\ \forall f$; $F = 360, n_f = 3\ \forall f$; and $F = 270, n_f = 4\ \forall f$. For each combination we generate and solve one random instance to provable optimality. The computation times, integer program size, optimal objective and bound gaps are recorded in Table 4 below.

Table 4: Solve times for three very large problems with 1080 jobs, $s_f = 100$ and $r = 1.0$

Problem	Solution time (s)	rows	cols	Optimal $L^*_{\max}$	*ordered-batch* bound gap g_{OB}	*ordered-job* bound gap g_{OJ}
$F = 540, n_f = 2$	4.76	3227	1613	58210	13	3967
$F = 360, n_f = 3$	70.04	3533	2134	43259	31	6800
$F = 270, n_f = 4$	114.14	4001	2689	36973	56	9607

The results in this table illustrate that our integer program performs very well when the number of job, n_f, in the family f is very small. For example, with two jobs in each family and 540 families we can solve the problem in just 4.76 seconds. The results from Hariri and Potts and Baker and Magazine and the bound gaps shown in this table suggest such a problem would be very difficult to solve using their algorithms.

The theory presented in Section 7.4 predicts the *ordered-batch* bound will be tight when the number of jobs in each family is small. However, the theory does not predict the tiny bound gaps we observe. For example, when $F = 540$ and $n_f = 2$ we can use (13) to compute a worst-case bound gap of $16,190$ which is much larger than the observed bound gap of 13. Therefore, it seems more theoretical analysis is needed to explain the exceptional performance of the *ordered-batch* bound.

10. Conclusions

We have developed a novel integer program for the single machine scheduling problem with family setups. This integer program exploits known order-based properties of optimal solutions to produce a strong formulation that outperforms the more traditional time-indexed model we tested. Our new model produces bounds that, for problems with large setup times or many families, are significantly tighter than those used in existing branch and bound algorithms. Furthermore, we have developed theory to explain why this bound is so tight. As a consequence of this tighter bound, we are able to find optimal solutions for significantly larger problems than have been solved to date. Furthermore, because we use a standard integer programming approach, we are able to exploit continuing advances in integer programming solvers while providing users with an off-the-shelf alternative to programming their own customised branch and bound code.

11. Acknowledgements

The first author is supported at Stanford by the PACCAR INC fellowship.

12. References

Afrati, F., Bampis, E., Chekuri, C., Karger, D., Kenyon, C., Khanna, S., Milis, J., Queyranne, M., Skutella, M., Stein, C., Sviridenko, M., 1999. Approximation schemes for minimizing average weighted completion time with release dates. In: Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science. IEEE Computer Society Press, Los Alamitos, CA, pp. 32–43.

Ahn, B.-H., Hyun, J.-H., 1990. Single facility multi-class job scheduling. Computer and Operations Research, 265–272.

Allahverdi, A., 2015. Third comprehensive survey on scheduling problems with setup times/costs (in press). European Journal of Operational Research.

Allahverdi, A., Gupta, J. N., Aldowaisan, T., 1999. A review of scheduling research involving setup considerations. Omega 27 (2), 219 – 239.
URL `http://www.sciencedirect.com/science/article/pii/S0305048398000425`

Allahverdi, A., Ng, C., Cheng, T., Kovalyov, M., 2008. A survey of scheduling problems with setup times or costs. European Journal of Operational Research 187 (3), 985–1032.

Avella, P., Boccia, M., D'Auria, B., 2005. Near-optimal solutions of large-scale single-machine scheduling problems. INFORMS Journal on Computing 17 (2), 183–191.
URL `http://joc.journal.informs.org/content/17/2/183.abstract`

Baker, K., Magazine, M., 2000. Minimizing maximum lateness with job families. European Journal of Operational Research 127, 126–139.

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

Bigras, L.-P., Gamache, M., Savard, G., 2008. Time-indexed formulations and the total weighted tardiness problem. INFORMS Journal on Computing 20 (1), 133–142.
URL `http://joc.journal.informs.org/content/20/1/133.abstract`

Blazewicz, J., Dror, M., Weglarz, J., 1991. Mathematical programming formulations for machine scheduling: A survey. European Journal of Operational Research 51 (3), 283 – 300.
URL `http://www.sciencedirect.com/science/article/pii/037722179190304E`

Bruno, J., Downey, P., 1978. Complexity of task sequencing with deadlines, set-up times and changeover costs. SIAM Journal on Computing 7, 23–40.

Cheng, T., Ng, C., Yuan, J., 2003. The single machine batching problem with family setup times to minimize maximum lateness is strongly NP-hard. Journal of Scheduling 6, 483–490.

COIN-OR CBC, 2015. COIN-OR CBC solver.
URL `http://optimization-suite.coin-or.org/ (accessed March 2015)`

CPLEX, 2015. CPLEX integer programming solver.
URL `http://cplex.com (accessed March 2015)`

Crauwels, H., Hariri, A., Potts, C., Wassenhove, L. V., 1998. Branch and bound algorithms for single-machine scheduling with batch set-up times to minimize total weighted completion time. Annals of Operations Research 83, 59–76.

Crauwels, H., Hariri, A., Potts, C., Wassenhove, L. V., 2010. Solving the single-machine sequencing problem using integer programming. Computers and Industrial Engineering 59, 730–735.

Dunstall, S., Wirth, A., Baker, K. R., 2000. Lower bounds and algorithms for flowtime minimization on a single machine with set-up times. Journal of Scheduling 3, 51–69.

Ghosh, B., Gupta, J., 1997. Batch scheduling to minimize maximum lateness. Operations Research Letters 21, 77–80.

Grundel, S., Ciftci, B., Borm, P., Hamers, H., 2013. Family sequencing and cooperation. European Journal of Operational Research 226 (3), 414 – 424. URL `http://www.sciencedirect.com/science/article/pii/S0377221712009149`

Gurobi, 2015. Gurobi integer programming solver. URL `http://gurobi.com (accessed March 2015)`

Hariri, A., Potts, C., 1997. Single machine scheduling with batch set-up times to minimize maximum lateness. Annals of Operations Research 70, 75–92.

Hoeffding, W., 1963. Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association 58 (301), 13–30.

Jin, F., Gupta, J., Song, S., Wu, C., 2009. Single machine scheduling with sequence-dependent family setups to minimize maximum lateness. Journal of Operational Research Society 61, 1181–1189.

Keha, A. B., Khowala, K., Fowler, J. W., 2009. Mixed integer programming formulations for single machine scheduling problems. Computers & Industrial Engineering 56 (1), 357 – 367. URL `http://www.sciencedirect.com/science/article/pii/S0360835208001265`

Li, K., Yang, S., 2009. N-identical parallel-machine scheduling research with minimizing total weighted completion times: Models, relaxations and algorithms. Applied Mathematical Modelling 33, 2145–2158.

Mason, A. J., Anderson, E. J., 1991. Minimizing flow time on a single machine with job classes and setup times. Naval Research Logistics 38, 333–350.

Monma, C., Potts, C., 1989. On the complexity of scheduling with batch setup times. Operations Research 37, 798–804.

Pan, C.-H., 1997. A study of integer programming formulations for scheduling problems. International Journal of Systems Science 28 (1), 33–41. URL `http://www.tandfonline.com/doi/abs/10.1080/00207729708929360`

Potts, C., Wassenhove, L. V., 1992. Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. Journal of the Operational Research Society 43, 395–406.

Potts, C. N., Kovalyov, M. Y., 2000. Scheduling with batching: A review. European Journal of Operational Research 120 (2), 228 – 249.
URL `http://www.sciencedirect.com/science/article/pii/S0377221799001538`

Queyranne, M., 1993. Structure of a simple scheduling polyhedron. Mathematical Programming 58, 262–285.

Queyranne, M., Schulz, A. S., 1994. Polyhedral approaches to machine scheduling, technical report 408/1994. Tech. rep., Technical University of Berlin, Berlin, Germany.

Savelsbergh, M. W. P., Uma, R. N., Wein, J., 2005. An experimental study of LP-based approximation algorithms for scheduling problems. INFORMS Journal on Computing 17 (1), 123–136.
URL `http://joc.journal.informs.org/content/17/1/123.abstract`

Schutten, J., Velde, S., Zijm, W., 1996. Single-machine scheduling with release dates, due dates and family setup times. Management Science 42, 1165–1174.

van den Akker, J., Hurkens, C., Savelsbergh, M., 2000. Time-indexed formulations for machine scheduling problems: Column generation. INFORMS Journal on Computing 12 (2), 111–124.
URL `http://joc.journal.informs.org/content/12/2/111.abstract`

Wagner, H. M., 1959. An integer linear-programming model for machine scheduling. Naval Research Logistics Quarterly 6, 131.

Webster, S., Baker, K., 1995. Scheduling groups of jobs on a single machine. Operations Research 43, 692–703.

Woeginger, G. J., 1998. A polynomial-time approximation scheme for single-machine sequencing with delivery times and sequence-independent batch set-up times. Journal of scheduling 1 (2), 79–87.

Zdrzałka, S., 1995. Analysis of approximation algorithms for single-machine scheduling with delivery times and sequence independent batch setup times. European Journal of Operational Research 80 (2), 371–380.

A. Proofs for Section 7

A.1. Proof of Lemma 8

In this subsection, we show for any family f and solution $\mathbf{x}$ to the OBLP we can construct a set of f-schedules Ω_f and weights $\omega_{f,\pi}$ on each f-schedule π such that (Ω_f, ω_f) covers $(\mathbf{x}, f)$. This construction is done by Algorithm A.1. Note that π-CONSTRUCTOR(p, S) is defined in Algorithm A.2.

Algorithm A.1 (Ω_f, ω_f)-constructor

```
1: function (Ω_f, ω_f)-CONSTRUCTOR(family f, solution x to OBLP)
2:     $\Omega_f \leftarrow \emptyset$
3:     $\hat{\mathbf{x}} \leftarrow \mathbf{x}$
4:     repeat
5:         $S \leftarrow \{b \in \mathcal{B}_f : \hat{x}_b > 0\}$
6:         Let $p = \arg\min_{b \in S} \hat{x}_b$
7:         $\pi \leftarrow \pi$-CONSTRUCTOR$(p, S)$
8:         $\omega_{f,\pi} \leftarrow \hat{x}_p$
9:         $\hat{x}_b \leftarrow (\hat{x}_b - I(b \in \pi)\omega_{f,\pi})$
10:        $\Omega_f \leftarrow \Omega_f \cup \{\pi\}$
11:    until $S = \emptyset$
12:    return $\Omega_f, \omega_f$
13: end function
```

Lemma A.1. *At each iteration of Algorithm A.1 the following equations hold:*

(i) $\hat{x}_b = x_b - \sum_{\pi \in \Omega_f} I(b \in \pi)\omega_{f,\pi}, \ \forall b \in \mathcal{B}_f$

(ii) $\sum_{b \in \mathcal{B}_f : i_b \leq k \leq j_b} \hat{x}_b = 1 - \sum_{\pi \in \Omega_f} \omega_{f,\pi}, \ \forall k \in \{1, ..., n_f\}$

Proof. (i) holds by applying induction to line 9 of Algorithm A.1. (ii) holds by summing both sides of (i) over $b \in \mathcal{B}_f : i_b \leq k \leq j_b$ and simplifying the right hand side using the fact that $\sum_{b \in \mathcal{B}_f : i_b \leq k \leq j_b} x_b = 1$ from Constraint 2 in the OBLP and $\sum_{b \in \mathcal{B}_f : i_b \leq k \leq j_b} I(b \in \pi) = 1$ from Definition 4 of a f-schedule. □

Algorithm A.2 π-constructor

```
function π-CONSTRUCTOR(batch $p$, set of batches $S$)
    $\pi \leftarrow \{p\}$
    while $j_\pi < n_f$ do
        find some $q \in S$ s.t. $j_q = i_\pi + 1$
        $\pi \leftarrow \pi \cup \{q\}$
    end while
    while $i_\pi > 1$ do
        find some $q \in S$ s.t. $i_q = j_\pi + 1$
        $\pi \leftarrow \pi \cup \{q\}$
    end while
    return $\pi$
end function
```

Before we prove the validity of Algorithm A.2 in Lemma A.2 we introduce the concept of a partial f-schedule in Definition 6. We can think of Algorithm A.2 as at each iteration expanding a partial f-schedule π until it is a complete f-schedule i.e. $i_\pi = 1$, $j_\pi = n_f$.

Definition 6. *We say that $\pi \subseteq \mathcal{B}_f$ is a partial f-schedule if for each job $k = i_\pi, ..., j_\pi$ the job k is contained in exactly one batch $b \in \pi$.*

Lemma A.2. *Given input from Algorithm A.1, Algorithm A.2 terminates in at most n_f iterations with an f-schedule π.*

Proof. Consider the partial schedule π at a given iteration of Algorithm A.2. In this proof, we show if $j_\pi < n_f$ then there exists some $\hat{x}_b > 0$ such that $j_\pi + 1 = i_b$. We can then add this b to π and continue the algorithm.

Let $u \in \pi$ be the batch such that $j_u = j_\pi$. Let $\theta = \sum_{b \in \mathcal{B}_f : i_b \leq k \leq j_b} \hat{x}_b$, from Lemma A.1 part (ii) we know that $\theta = \sum_{b \in \mathcal{B}_f : i_b \leq k \leq j_b} \hat{x}_b = 1 - \sum_{\pi \in \Omega_f} \omega_{f,\pi}$.

Using $k = j_\pi$ yields:

$$\begin{aligned}
\theta &= \sum_{b \in \mathcal{B}_f : i_b \le j_\pi \le j_b} \hat{x}_b \\
&= \sum_{b \in \mathcal{B}_f : j_b = j_\pi} \hat{x}_b + \sum_{b \in \mathcal{B}_f : i_b \le j_\pi < j_b} \hat{x}_b \\
&= \sum_{b \in \mathcal{B}_f : j_b = j_\pi} \hat{x}_b + \sum_{b \in \mathcal{B}_f : i_b < j_\pi + 1 \le j_b} \hat{x}_b \\
&\ge \hat{x}_u + \sum_{b \in \mathcal{B}_f : i_b < j_\pi + 1 \le j_b} \hat{x}_b
\end{aligned}$$

Re-arranging:

$$\sum_{b \in \mathcal{B}_f : i_b < j_\pi + 1 \le j_b} \hat{x}_b \le \theta - \hat{x}_u \tag{A.1}$$

Using $k = j_\pi + 1$ yields:

$$\begin{aligned}
\theta = \sum_{b \in \mathcal{B}_f : i_b \le j_\pi + 1 \le j_b} \hat{x}_b &= \sum_{b \in \mathcal{B}_f : j_\pi + 1 = i_b} \hat{x}_b + \sum_{b \in \mathcal{B}_f : i_b < j_\pi + 1 \le j_b} \hat{x}_b \\
&\le \sum_{b \in \mathcal{B}_f : j_\pi + 1 = i_b} \hat{x}_b + \theta - \hat{x}_u
\end{aligned}$$

where the last line comes from applying (A.1). Now, finally:

$$0 < \hat{x}_u \le \sum_{b \in \mathcal{B}_f : j_\pi + 1 = i_b} \hat{x}_b$$

Therefore there exists some $\hat{x}_b > 0$ such that $j_\pi + 1 = i_b$.

By a symmetrical argument if $i_\pi > 1$ then there exists some $\hat{x}_b > 0$ such that $i_\pi - 1 = j_b$. □

Lemma A.3. *After at most $|\mathcal{B}_f|$ outer iterations Algorithm A.1 terminates with an (Ω_f, ω_f) that covers $(\mathbf{x}, f)$.*

Proof. Consider $S \leftarrow \{b \in \mathcal{B}_f : \hat{x}_b > 0\}$. At each iteration the batch $p = \arg\min_{b \in S} \hat{x}_b$ must be removed from the set S, hence the set size must decrease by at least one. It immediately follows that after at most $|S| \le |\mathcal{B}_f|$ iterations the algorithm terminates.

Since the algorithm terminates with $\hat{x}_b = 0$ for all $b \in \mathcal{B}_f$ from Lemma A.1 part (i) it follows that $x_b = \sum_{\pi \in \Omega_f} I(b \in \pi) \omega_{f,\pi}$ and therefore that (Ω_f, ω_f) covers $(\mathbf{x}, f)$. □

The next Lemma follows from Lemma A.2 and Lemma A.3.

Lemma 8. *For any family f and solution $\mathbf{x}$ to the SOBLP we can construct an (Ω_f, ω_f) that covers $(\mathbf{x}, f)$ in $O(|\mathcal{B}_f|^2 n_f)$ time.*

Proof. From Lemma A.2 that Algorithm A.2 takes at most n_f iterations and each iteration requires at most $O(|\mathcal{B}_f|)$ time. Therefore Algorithm A.2 requires at most $O(n_f|\mathcal{B}_f|)$ time. Furthermore from Lemma A.3, Algorithm A.1 makes at most $|\mathcal{B}_f|$ calls to Algorithm A.2. Hence the total time required for Algorithm A.1 is $O(|\mathcal{B}_f|^2 n_f)$. □

A.2. Proof of Theorem 3

Before we prove Lemma A.4 we need to clarify how we randomly round solutions, i.e. how X_b^{R} and L_b^{R} are generated. Algorithm A.3 describes how we take a fractional solution to our LP and round it to a 'nearby' integer solution.

Algorithm A.3 An algorithm that randomly rounds a solution to OBLP

```
function RANDOM-ROUNDER(solution $\mathbf{x}$)
    for each family $f$ do
        $(\Omega_f, \omega_f) \leftarrow \Omega_f$-CONSTRUCTOR( $f$, $\mathbf{x}$ )
        $\Pi_f \sim P_{(\Omega_f,\omega_f)}$
    end for
    for each batch $b$ do
        $X_b^{\mathrm{R}} \leftarrow I(b \in \Pi_f)$
        $L_b^{\mathrm{R}} \leftarrow \sum_{r=1}^{b} \rho_r X_r^{\mathrm{R}} - \delta_b$
    end for
    return $X_b^{\mathrm{R}}, L_b^{\mathrm{R}}$
end function
```

Because (Ω_f, ω_f) covers $(\mathbf{x}, f)$ we know that $x_b = E[X_b^{\mathrm{R}}]$ which implies $L_b^{\mathrm{LP}} = E[L_b^{\mathrm{R}}]$. We are now ready to prove Lemma A.4.

Lemma A.4. *With probability at most $\frac{1}{B^2}$:*

$$L_b^{\mathrm{R}} \geq L_b^{\mathrm{LP}} + \sqrt{\log(B) \sum_{f=1}^{F} \tau_f^2}$$

where L_b^{R} is the maximum lateness of batch b in a schedule generated by Algorithm A.3.

Proof. Consider the random variable:

$$W_{b,f}^{\mathrm{R}} = \sum_{r \in \mathcal{B}_f : r \leq b} \rho_r X_r^{\mathrm{R}}$$

We can write the maximum lateness of the batch b in our random rounding as:

$$L_b^{\mathrm{R}} = \sum_{f \in F} W_{b,f}^{\mathrm{R}} - \delta_b$$

Since $W_{b,1}^{\mathrm{R}}, ..., W_{b,F}^{\mathrm{R}}$ are independent for fixed b, we can apply Hoeffding's inequality (Hoeffding, 1963) to L_b^{R} using the fact that $0 \leq W_{f,b}^{\mathrm{R}} \leq \tau_f$ yielding:

$$P(L_b^{\mathrm{R}} \geq E[L_b^{\mathrm{R}}] + \epsilon) \leq e^{-\frac{2\epsilon^2}{\sum_{f=1}^{F} \tau_f^2}}$$

Setting $\epsilon = \sqrt{\log{(B)} \sum_{f=1}^{F} \tau_f^2}$ and $L_b^{\mathrm{LP}} = E[L_b^{\mathrm{R}}]$ it follows that:

$$P\left(L_b^{\mathrm{R}} \geq L_b^{\mathrm{LP}} + \sqrt{\log{(B)} \sum_{f=1}^{F} \tau_f^2} \right) \leq e^{-2\log{(B)}} = \frac{1}{B^2}$$

□

Theorem 3. *With probability at least* $1 - \frac{1}{B}$*:*

$$L_{\max}^{\mathrm{R}} \leq L_{\max}^{\mathrm{LP}} + \sqrt{\log(B) \sum_{f=1}^{F} \tau_f^2}$$

where $L_{\max}^{\mathrm{R}}$ *is the maximum lateness of the schedule generated by Algorithm A.3.*

Proof. Follows by applying a union bound to Lemma A.4. □

A.3. Proof of Theorem 4

Before we can prove Theorem 4, we need to formalize how we compute our heuristic schedule $\mathbf{x}'^{\mathrm{R}}$; this is given by Algorithm A.4.

Algorithm A.4 Algorithm for computing the schedule $\mathbf{x}'^{\mathrm{R}}$

```
$P' \leftarrow$ PROBLEM-SPLITTER(P)
$\mathbf{x}^{\mathrm{LP}} \leftarrow$solve SOBLP for $P'$
$\mathbf{x}'^{\mathrm{R}} \leftarrow$ RANDOM-ROUNDER($\mathbf{x}^{\mathrm{LP}}$)
```

Algorithm A.5 takes an instance of a minimizing maximum lateness with family setup times on a single-machine problem and divides the families to produce a new instance P' with at most $n^{1/3}$ jobs in each family.

Algorithm A.5 Algorithm for splitting large families into many smaller families

```
function PROBLEM-SPLITTER(Problem P)
    construct a new (empty) problem instance $P'$
    $q \leftarrow 1$
    for each family $f$ in the original problem $P$ do
        $\sigma_f \leftarrow \left\lceil \frac{n_f}{n^{1/3}} \right\rceil$                    ▷ number of splits of family $f$
        $\eta_f \leftarrow \left\lfloor \frac{n_f}{\sigma_f} \right\rfloor$         ▷ maximum number of jobs in each new family
        for $r = 1$ to $\sigma_f$ do
            Consider the jobs $(r-1)\eta_f + 1, (r-1)\eta_f + 2, \ldots, \min\{r\eta_f, n_f\}$
            Add these jobs to a new family $q$ in the new instance $P'$
            $q \leftarrow q + 1$
        end for
    end for
    return $P'$                                  ▷ return new problem instance
end function
```

Lemma A.5. *The new instance P' produced by Algorithm A.5 has $n'_q \leq n^{1/3}$ for all $q \in \{1, ..., F'\}$ and $L'^*_{\max} \leq L^*_{\max} + s_{\max} n^{2/3}$*

Proof. To show $n'_q \leq n^{1/3}$ consider $n'_q \leq \eta_f = \left\lfloor \frac{n_f}{\lceil n_f/n^{1/3} \rceil} \right\rfloor \leq \left\lfloor \frac{n_f}{n_f/n^{1/3}} \right\rfloor \leq n^{1/3}$. Consider any schedule for P. Using the same ordering of the jobs we can construct an identical schedule for P' except for at most $F' - F$ additional setups. Therefore $L'^*_{\max} \leq L^*_{\max} + s_{\max}(F' - F)$. We show that $F' - F \leq n^{2/3}$ by observing that $F' - F \leq \sum_{f=1}^{F} (\sigma_f - 1) \leq \sum_{f=1}^{F} n_f / n^{1/3} = n^{2/3}$. □

Theorem 4. *With probability at least $1 - \frac{1}{B}$:*

$$L'^{\mathrm{R}}_{\max} \leq L^*_{\max} + n^{2/3}(1 + \sqrt{2\log(n)})(s_{\max} + p_{\max})$$

where $L'^{\mathrm{R}}_{\max}$ is the maximum lateness of the schedule generated by Algorithm A.4.

Proof. Now by Theorem 3 with probability at least $1 - \frac{1}{B}$:

$$\begin{aligned}
L'^{\mathrm{R}}_{\max} &\leq L'^{*}_{\max} + \sqrt{\log(n^2) \sum_{f=1}^{F} \tau_f^2} \\
&\leq L'^{*}_{\max} + (s_{\max} + p_{\max}) \sqrt{2\log(n) \sum_{f=1}^{F} n_f^2} \\
&\leq L'^{*}_{\max} + (s_{\max} + p_{\max}) \sqrt{2\log(n) n^{1/3} \sum_{f=1}^{F} n_f} \\
&= L'^{*}_{\max} + (s_{\max} + p_{\max}) \sqrt{2\log(n) n^{4/3}} \\
&= L'^{*}_{\max} + (s_{\max} + p_{\max}) n^{2/3} \sqrt{2\log(n)}
\end{aligned}$$

Since $L'^{*}_{\max} \leq L^{*}_{\max} + s_{\max} n^{2/3}$ it follows that:

$$L'^{\mathrm{R}}_{\max} \leq L^{*}_{\max} + (s_{\max} + p_{\max}) n^{2/3} (\sqrt{2\log(n)} + 1)$$

□

B. Time Indexed Model

This section describes a time indexed formulation for this problem.

We start by discretising time into time periods $1, 2, ..., T$, where we put $T = \sum_{f=1}^{F} \sum_{k=1}^{n_f} (s_f + p_{(k,f)})$ to ensure an optimal solution exists in which all jobs are processed by time T. As before, we assume that we have enumerated all possible B batches, and for each batch (i_b, j_b, f_b), $b = 1, 2, ..., B$ we introduce binary variables $x_{b,t}$, $t = 1, 2, ..., T - \rho_b + 1$ where ρ_b is the total processing time of batch b. We then let $x_{b,t} = 1$ if batch b starts in period t, and zero otherwise. Note that if $x_{b,t} = 1$, then this batch will occupy the machine in periods t, $t+1$, ..., $t + \rho_b - 1$, and will have a batch lateness given by $t + \rho_b - \delta_b$. These variables allow us to formulate our time-indexed batch integer program (TIBIP) as follows:

$$\text{TIBIP: min} \quad \hat{L}_{\max} \tag{B.1}$$

$$\text{s.t.} \quad (t+\rho_b-\delta_b)x_{b,t} \le \hat{L}_{\max} \qquad \forall\, b \in \{1,...,B\}, t \in \{1,2,...,T-\rho_b+1\} \tag{B.2}$$

$$\sum_{b:i_b \le k \le j_b, f_b=f} \sum_{t=1}^{T-\rho_b+1} x_{b,t} = 1 \qquad \forall\, f \in \{1,...,F\}, k \in \{1,2,...,n_f\} \tag{B.3}$$

$$\sum_{b=1}^{B} \sum_{t:t \le s \le t+\rho_b-1} x_{b,t} \le 1 \qquad \forall\, s \in \{1,2,3,...,T\} \tag{B.4}$$

$$x_{b,t} \in \{0,1\} \qquad \forall\, b \in \{1,2,...,B\},\ t \in \{1,2,...,T\} \tag{B.5}$$

$$\hat{L}_{\max} \in \mathbb{R} \tag{B.6}$$

Constraint (B.2) calculates $\hat{L}_{\max}$ for the solution, while (B.3) ensures each job appears in exactly one batch and (B.4) ensures only one batch is being processed during each time period. The rest of the formulation ensures the solution is a set of batches with minimal maximum lateness.