A New MIP Model for Parallel-Batch Scheduling with Non-Identical Job Sizes

Sebastian Kosch and J. Christopher Beck

Department of Mechanical & Industrial Engineering
University of Toronto, Toronto, Ontario M5S 3G8, Canada
{skosch,jcb}@mie.utoronto.ca

Abstract. Parallel-batch machine problems arise in numerous manufacturing settings from semiconductor manufacturing to printing. They have recently been addressed in constraint programming (CP) via the combination of the novel `sequenceEDD` global constraint with the existing `pack` constraint to form the current state-of-the-art approach. In this paper, we present a detailed analysis of the problem and derivation of a number of properties that are exploited in a novel mixed integer programming (MIP) model for the problem. Our empirical results demonstrate that the new model is able to outperform the CP model across a range of standard benchmark problems. Further investigation shows that the new MIP formulation improves on the existing formulation primarily by producing a much smaller model and enabling high quality primal solutions to be found very quickly.

1 Introduction

Despite the widespread application of mixed integer programming (MIP) technology to optimization problems in general and scheduling problems specifically,[1] there is a significant body of work that demonstrates the superiority of constraint programming (CP) and hybrid approaches for a number of classes of scheduling problems [1–5]. While the superiority is often a result of strong inference techniques embedded in global constraints [6–8], it is sometimes due to problem-specific implementation in the form of specialized global constraints [4] or instantiations of decomposition techniques [1–3]. The flexibility of CP and decomposition approaches which facilitates such implementations is undoubtedly positive from the perspective of solving specific problems better. However, the ability to create problem-specific approaches is in some ways in opposition to the compositionality and model-and-solve "holy grail" of CP [9]: to enable users to model and solve problems without *implementing* anything new at all.

Our overarching thesis is that, in fact, MIP technology is closer to this goal than CP, at least in the context of combinatorial optimization problems. In our investigation of this thesis, we are developing MIP models for scheduling

[1] For example, of the 58 papers published in the *Journal of Scheduling* in 2012, 19 use MIP, more than any other single approach.

problems where the current state of the art is customized CP or hybrid approaches. Heinz et al. [10] showed that on a class of resource allocation and scheduling problems, a MIP model could be designed that was competitive with the state-of-the-art logic-based Benders decomposition. This paper represents a similar contribution in different scheduling problem: a parallel-batch processing problem which has previously been attacked by MIP, branch-and-price [11], and CP [4] with the latter representing the state of the art.

We propose a MIP model inspired by the idea of modifying a canonical feasible solution. The definition of our objective function in this novel context is not intuitive until we reason algorithmically about how constraints and assignments interact – a strategy often used in designing metaheuristics. Indeed, we suggest that the analogy between branching on independent decision variables and making moves between neighbouring schedules should be explored in more detail for a range of combinatorial problems.

In the next section we present the formal problem definition and discuss existing approaches. In Section 3 we prove a number of propositions that allow us to formally propose a novel MIP model for the problem in Section 4. Section 5 presents our empirical results, demonstrating that the performance of the new model is superior to the existing CP model, both in terms of mean time to find optimal solutions and in terms of solution quality when optimal solutions could not be found within the time limit.

2 Background

Batch machines with limited capacity exist in many manufacturing settings in forms such as ovens [12], autoclaves [4], and tanks [13]. In this paper, we tackle the problem of minimizing the maximum lateness, $L_{\max}$, in a single machine parallel batching problem where each job has an individual due date and size.

We use the following notation: a set $\mathcal{J}$ of n jobs is to be assigned to a set of n batches $\mathcal{B} = \{B_1, \ldots, B_n\}$. Batches can hold multiple jobs or remain empty. Each job j has a processing time, p_j, a size, s_j, and a due date, d_j. Jobs can be assigned to arbitrary batches, as long as the sum of the sizes of the jobs in a batch does not exceed the machine capacity, b.

The single machine processes one batch at a time. Each batch B_k has a *batch start date* S_k, a *batch processing time*, defined as the longest processing time of all jobs assigned to the batch, $P_k = \max_{j \in \mathcal{B}_k}(p_j)$, and a *batch completion date*, which must fall before the start time of the next batch, $C_k = S_k + P_k \leq S_{k+1}$.

The lateness of a job j, L_j, is the completion time of its batch C_k less its due date d_j. The objective function is to minimize the maximum lateness over all jobs, $L_{\max} = \max_{j \in \mathcal{J}}(L_j)$. Since we are interested in the maximum lateness, only the earliest-due job in each batch matters and we define it as the *batch due date* $D_k = \min_{j \in \mathcal{B}_k}(d_j)$.

The problem can be summarized as $1|p\text{-}batch; b < n; non\text{-}identical|L_{\max}$ [4, 11], where $p\text{-}batch; b < n$ represents the resource's parallel-batch nature and

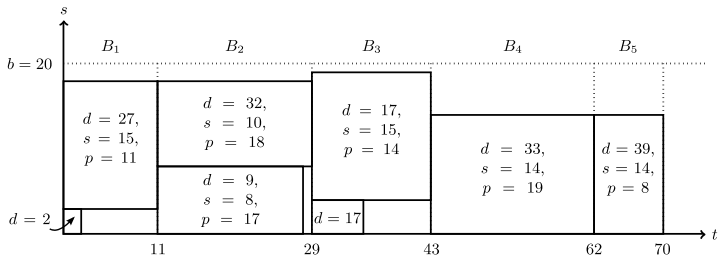

Fig. 1. An optimal solution to an example problem with eight jobs (values for s_j and p_j are not shown for the two small jobs in batches 1 and 3, respectively).

its finite capacity. A version with identical job sizes was shown to be strongly NP-hard by Brucker et al. [14]; this problem, therefore, is no less difficult.

Figure 1 shows a solution to a sample problem with eight jobs and resource capacity $b = 20$. The last batch has the maximum lateness $L_5 = C_5 - D_5 = 70 - 39 = 31$.

2.1 Reference MIP model

The problem is formally defined by MIP Model 2.1, used by Malapert et al. [4] for comparison with their CP model (see below).

$$\text{Min.} \quad L_{\max} \tag{1}$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{B}} x_{jk} = 1 \qquad \forall j \in \mathcal{J} \tag{2}$$

$$\sum_{j \in \mathcal{J}} s_j x_{jk} \leq b \qquad \forall k \in \mathcal{B} \tag{3}$$

$$p_j x_{jk} \leq P_k \qquad \forall j \in \mathcal{J}, \forall k \in K \tag{4}$$

$$C_{k-1} + P_k = C_k \qquad \forall k \in \mathcal{B} \tag{5}$$

$$(d_{\max} - d_j)(1 - x_{jk}) + d_j \geq D_k \qquad \forall j \in \mathcal{J}, \forall k \in \mathcal{B} \tag{6}$$

$$C_k - D_k \leq L_{\max} \qquad \forall k \in \mathcal{B} \tag{7}$$

$$D_{k-1} \leq D_k \qquad \forall k \in \mathcal{B} \tag{8}$$

$$x_{jk} \in \{0,1\}, C_k \geq 0, P_k \geq 0, D_k \geq 0 \qquad \forall j \in \mathcal{J}, \forall k \in \mathcal{B} \tag{9}$$

Model 2.1. Reference MIP model

The decision variables, x_{jk}, are binary variables where $x_{jk} = 1$ iff job j is assigned to batch k. Constraints (2) ensure that each job j is assigned to exactly one batch k. Constraints (3) ensure that no batch exceeds the machine capacity,

b. Constraints (4) define each batch's processing time, P_k, as the maximum processing time of the jobs j assigned to it. Constraints (6) implement the definition of D_k while ensuring that for empty batches k, $D_k = d_{\max}$. Constraints (5) define each batch's completion time, C_k, as that of the previous batch, plus the batch's processing time. Constraints (7) define the objective value $L_{\max}$. Constraints (8) sort the batches by due date, based on a well-known dominance rule: there exists an optimal solution with batches scheduled in earliest-due-date-first order (EDD). This stems from the fact that if all jobs are already assigned, the problem reduces to a polynomial-time solvable single machine problem $(1|D_k|L_{\max})$ [15].

2.2 Previous Work

Malapert et al. [4] present a CP formulation of the problem (see Model 2.2) which relies on two global constraints: `pack` [16], which constrains the job-to-batch assignments such that no capacity limits are violated, and `sequenceEDD`, which enforces the EDD order over the batches. The implementation of the latter constraint is the main contribution of the paper and is primarily responsible for the strong performance. `sequenceEDD` includes a set of pruning rules that update the lower and upper bounds on $L_{\max}$ and on the number of batches. Based on these bounds, other assignments are then eliminated from the set of feasible assignments.

$$
\begin{aligned}
\text{Min.} \quad & L_{\max} && && (10)\\
\text{s.t.} \quad & \texttt{maxOfASet}(P_k, B_k, [p_j]_k, 0) && \forall k \in \mathcal{B} && (11)\\
& \texttt{minOfASet}(D_k, B_k, [d_j]_k, d_{\max}) && \forall k \in \mathcal{B} && (12)\\
& \texttt{pack}(B_k, A_j, S_k, M, s_j) && && (13)\\
& \texttt{sequenceEDD}(B_k, D_k, P_k, M, L_{\max}) && && (14)
\end{aligned}
$$

Model 2.2. CP model proposed by Malapert et al.

Constraints (11) define P_k as the maximum of the set of processing times $[p_j]_k$ belonging to the jobs assigned to batch B_k, with a minimum value of 0 (note that the notation is adapted from Malapert et al. to match that in this paper). Constraints (12) define D_k as the minimum of the due dates $[d_j]_k$ associated with the set of jobs assigned to the batch B_k, with a maximum of $d_{\max}$, the largest due date among all given jobs. Constraint (13) implements the limited batch capacity b. It uses propagation rules incorporating knapsack-based reasoning, as well as a dynamic lower bound on the number of non-empty batches M [4, 16]. Note that this constraint handles the channeling between the set of jobs assigned to batch B_k, and the assigned batch index A_j for each job j. The limited capacity is enforced by setting the domain of the batch loads, S_k, to $[0, b]$. Constraint (14)

ensures that the objective value $L_{\max}$ is equal to the maximum lateness of the batches scheduled according to the EDD rule.

The problem has also been addressed with a detailed branch-and-price algorithm [11], which is described in [4] as follows: each batch is a column in the column generation master problem. A solution of the master problem is a feasible sequence of batches. The objective of the subproblem is to find a batch which improves the current solution of the master problem. Malapert et al. [4] showed that their CP model was significantly faster than the branch-and-price algorithm which itself was more efficient than the reference MIP model.

Other authors have examined similar problems: Azizoglu & Webster [17] provide an exact method and a heuristic for the same problem, but minimize makespan ($C_{\max}$) instead of $L_{\max}$, similar to the work by Dupont and Dhaenens-Flipo [18]. Exact methods have been proposed for multi-agent variants with different objective functions by Sabouni and Jolai [19], for makespan minimization on single batch machines by Kashan et al. [20], and for makespan minimization on parallel batch machines with different release dates [21]. An extensive review of MIP models in batch processing is given by Grossmann [13].

3 Exploiting the Problem Structure

In this section, we make a series of observations about the parallel-batch scheduling problem that allow us to develop a novel MIP formulation.

3.1 The single-EDD schedule and assigning jobs to earlier batches

We can exploit the EDD rule to eliminate $\frac{1}{2}(n^2-n)$ of the n^2 potential job-batch assignments a priori.

We first re-index all jobs in non-decreasing due date (and in non-decreasing processing time in case of a tie). For the remainder of this paper, consider all jobs to be indexed in this way. We then define the *single-EDD* schedule, in which each batch B_k contains the *single* job j matching its index (i.e., $x_{jk} = 1$ iff $j = k$), such that EDD is always satisfied. We refer to j as the *host job* of batch B_k, while other jobs assigned to B_k, if any, are *guest jobs*.

Lemma 1. *Consider a schedule S in which B_k is the earliest-scheduled batch such that its host job j is assigned to a later-scheduled batch B_{k+m}. In this schedule, B_k is either empty or $D_k = d_j$ (even though $j \notin B_k$).*

Proof. If B_k is non-empty, $D_k \geq d_j$: since B_k is defined as the earliest scheduled batch whose host job is scheduled later, B_k cannot host jobs due before d_j. But if $D_k > d_j$ then EDD is violated, as $D_{k+m} = d_j$. Thus, B_k is empty, or $D_k = d_j$ (due to other jobs with due date equal to d_j assigned to B_k). □

Proposition 1. *There exists an optimal solution in which job j is assigned to batch B_k, $j \leq k$.*

Proof. Consider again schedule S. Since $D_{k+m} = d_j$, EDD requires that no batch $B_q, k \leq q < k+m$, is due after d_j. By Lemma 1, we only need to consider the following two cases:

1. B_k is empty, so $P_k = 0$. Since EDD is not violated, we know that $D_q = d_j \ \forall \ B_q, k \leq q \leq k+m$. We can assign all jobs from B_{k+m} to B_k, such that $P_{k+m} = 0$. $L_{\max}$ will stay constant, as the completion time of the last-scheduled of all batches due at d_j does not change.
2. B_k is non-empty and due at $D_k = d_j$ (although $j \notin B_k$), due to at least one job g from a later-scheduled batch for which $d_g = d_j$, which is assigned to B_k. In this case, since $D_k = d_j = D_{k+m}$ and since EDD is not violated, *all* batches B_q where $k \leq q \leq k+m$ must be due at d_j. But then we can re-order these batches such that their respective earliest-due jobs are once again assigned to their single-EDD indices. The jobs in B_{k+m} (including j) will be assigned to B_k as a result. $L_{\max}$ is not affected by this re-assignment, as the completion time of the last-scheduled batch due at d_j does not change.

□

We thus introduce the following constraint to exclude solutions in which jobs are assigned to later batches than their single-EDD batches.

$$x_{jk} = 0 \quad \forall \{j \in \mathcal{J}, k \in \mathcal{B} | j < k\} \tag{15}$$

We can also show that in every non-empty batch B_k, the earliest-due job j must be the host job. This means that when batch B_k's host job is assigned to an earlier batch, no other jobs can be assigned to B_k; a batch that is *hostless* must be empty. This requirement rests on the following proposition.

Proposition 2. *There exists an optimal solution that has no hostless, non-empty batches.*

Proof. Consider an EDD-ordered schedule in which batch B_j is the last-scheduled batch which is hostless but non-empty: instead of its host job, only a set G of later-due guest jobs is assigned to B_j ($j \notin G$).

The earliest-due job $g \in G$ must have the same due date as batch B_{j+1}: if it is due later, EDD is violated; if it is due earlier, G is not a set of later-due guest jobs. Job g's own host batch B_g (which is hostless) cannot itself be due later than $d_g = D_{j+1}$ – this would require B_g to have guest jobs from later batches, but we defined B_k as the last batch with this property – therefore, $D_q = D_{j+1}$ for all batches $B_q, j \leq q \leq g$.

Then we can re-assign the guest jobs G from B_j into B_g, such that g is again host job in its own single-EDD batch. This re-assignment has no impact on $L_{\max}$ since it makes $P_j = 0$, resulting in the same completion time of the set of all batches with batch due date $D = D_{j+1}$. □

The above proposition translates to the following constraint:

$$x_{kk} \geq x_{jk} \quad \forall \{j \in \mathcal{J}, k \in \mathcal{B} | j > k\} \tag{16}$$

This observation allows us to define the due date of *all* batches to be the due date of their respective host jobs: $D_k = d_j, \forall\{j \in \mathcal{J}, k \in \mathcal{B} | j = k\}$. This rule holds even for empty batches B_k: $P_k = 0$, so $C_k = C_{k-1}$; but $D_{k-1} \leq D_k$ due to this rule, so $L_{k-1} \geq L_k$ and thus L_k has no impact on $L_{\max}$.

3.2 Reformulating the objective function

We can formulate each batch's lateness, L_k, as its lateness in the single-EDD schedule, modified by the assignment of jobs into and out of batches $B_h, h \leq k$.

We first define $\mathcal{B}^\star \subseteq \mathcal{B}$ as the set of batches B_k which, given any EDD schedule, are the last-scheduled among all batches with due date D_k, since we can make the following observation.

Proposition 3. *Given a set of batches with equal due date in a schedule, we only need to consider the lateness of the one scheduled last.*

Proof. In an EDD ordering, the lateness of the batch scheduled last is greater than (or equal to, in the case of an empty batch) the lateness values of all other batches sharing its due date as it has the latest completion date. □

This fact allows us to reduce the number of constraints defining $L_{\max}$, as we only need to consider batches $\mathcal{B}^\star$ as potential candidates for $L_{\max}$.

To simplify the following exposition, we define the term *move* as the re-assignment of a job j from its single-EDD batch B_k to an earlier batch $B_h, h < k$, such that $x_{jk} = 0$ and $x_{jh} = 1$ and j is a guest job in B_h. Any schedule can thus be understood as a set of such moves, executed in arbitrary order starting from the single-EDD schedule. To define the objective function, we consider the change in $L_{\max}$ associated with individual moves.

Consider any EDD schedule, such as the one in Figure 2(a). Moving a job j from its single-EDD batch $B_{k=j}$ into an earlier batch B_e has the following effect:

- the lateness of all batches $B_i, i \geq k$ is reduced by p_j (Figure 2(b)),
- the lateness of all batches $B_h, h \geq e$ is increased by $\max(0, p_j - P_e)$, where P_e is the processing time of batch B_e before j is moved into it (Figure 2(c)).

In any batch, only the host job's lateness is relevant to $L_{\max}$. In other words, the lateness of batch B_k equals the lateness of job $j = k$, unless the job was moved into an earlier batch (in which case $P_k = 0$ due to Proposition 2 and $L_k = L_{k-1}$). Therefore, we can understand the lateness of batch B_k as its lateness in the single-EDD schedule, written as $L_{k,\text{single}}$, modified by the summed effect that all moves of other jobs into and out of batches $h \leq k$ have on the completion time of B_k:

$$L_k = L_{k,\text{single}} + \sum_{h \leq k} \underbrace{P'_h - p_h(2 - x_{hh})}_{T_h} \quad \forall k \in \mathcal{B}^\star \tag{17}$$

$$P'_k \geq p_j x_{jk} \quad \forall\{j \in \mathcal{J}, k \in \mathcal{B} | j \geq k\} \tag{18}$$

$$P'_k \geq p_j \quad \forall\{j \in \mathcal{J}, k \in \mathcal{B} | j = k\} \tag{19}$$

Fig. 2. Moving a job in a single-EDD schedule. Job 5 (marked "$p_5 = 10$") is moved from its single-EDD batch 5 into the earlier batch 3. This changes the lateness of job 7 (marked $\star$) from $L_{7,\text{single}}$ to $L_{7,\text{single}} - 10 + 6 = L_{7,\text{single}} - 4$.

where $P'_k = \max(P_k, p_k) \forall k \in \mathcal{B}$ as defined in constraints (18) and (19).

For every batch $B_k \in \mathcal{B}^\star$, consider the possible scenarios for all batches $B_h, h \leq k$:

- Batch B_h holds its host job. Then $x_{hh} = 1$ and the summand T_h evaluates to $P'_h - p_h$. If B_h has guest jobs, then $P'_h - p_h > 0$ if any of them are longer than the host job; if all guests are shorter, $P'_h = p_h$ and $T_h = 0$.
- Batch B_h is hostless and thus empty. We require $T_h = -p_h$ in accordance with Figure 2(b). To achieve this, we state in constraints (19) that P'_h never drops below the length of its host job, even when $P_h = 0$. With this in effect, the minimization objective enforces $P'_h = p_h$ and $T_h = P'_h - 2p_h = -p_h$.

Thus, we add to L_k the increase in processing time due to guests, $\max(0, P'_h - p_h)$, for every non-empty batch B_h. We subtract from L_k the host job processing time p_h for every empty batch B_h. This is congruent with Figure 2 above.

The net sum of these additions and subtractions to and from $L_{k,\text{single}}$ adjusts the lateness of batch k to its correct value given the values of x_{jk}.

Proposition 4. *Constraints* (17)–(19) *correctly define* L_k *for each batch* B_k.

Proof. By induction on k, our base case is the lateness of the first batch ($k = 1$). It is clear that the lateness of B_k is equal to its single-EDD lateness, plus $\max(P_1 - p_1, 0)$ since guest jobs may cause $P_1 > p_1$:

$$L_1 = L_{1,\text{single}} + \underbrace{P'_1 - p_1(2 - x_{11})}_{=\max(P_1 - p_1, 0)}. \tag{20}$$

Our induction hypothesis is that the proposition holds for any batch B_k:

$$L_k = L_{k,\text{single}} + \sum_{h \leq k} P'_h - p_h(2 - x_{hh}) \tag{21}$$

To show how an expression for L_{k+1} then follows, we relate L_{k+1} to L_k:

$$L_{k+1} = L_k + \underbrace{P_{k+1}}_{C_{k+1}-C_k} - \underbrace{(d_{k+1} - d_k)}_{D_{k+1}-D_k} \tag{22}$$

The difference $L_{k+1} - L_k$ can also be written in terms of single-EDD lateness values and processing time adjustments due to guests or hostlessness, all of which are expressed in known terms:

$$P_{k+1} - (d_{k+1} - d_k) = L_{k+1,\text{single}} - L_{k,\text{single}} + \begin{cases} \max(P_{k+1} - p_{k+1}, 0) & x_{k+1,k+1} = 1 \\ -p_{k+1} & x_{k+1,k+1} = 0 \end{cases}. \tag{23}$$

The conditional expression is equivalent to $P'_{k+1} - p_{k+1}(2 - x_{k+1,k+1})$. We can now rewrite (22) for L_{k+1} and arrive at

$$\begin{aligned} L_{k+1} = &\left[L_{k,\text{single}} + \sum_{h \le k} P'_h - p_h(2 - x_{hh}) \right] \\ &+ L_{k+1,\text{single}} - L_{k,\text{single}} + P'_{k+1} - p_{k+1}(2 - x_{k+1,k+1}), \end{aligned} \tag{24}$$

which, after cancelling out $L_{k,\text{single}}$ terms, becomes

$$L_{k+1} = L_{k+1,\text{single}} + \sum_{h \le k+1} P'_h - p_h(2 - x_{hh}) \tag{25}$$

and agrees with (21). Since (25) follows from (21), and the latter is true for the base case of $k = 1$, (17) is true for all k. □

Note also that in the case of an empty batch $B_k \in \mathcal{B}^\star$, if $B_{k-1} \notin \mathcal{B}^\star$, $d_k = d_{k-1}$ and $x_{kk} = 0$, so $L_k = L_{k-1}$ as evident from (24); if $B_{k-1} \in \mathcal{B}^\star$, $d_k > d_{k-1}$, and thus $L_k < L_{k-1}$. as $d_k = d_{k-1}$ and $x_{kk} = 0$ if B_k is empty.

3.3 Additional lazy constraints

Lazy constraints [22] are also used in the model. Lazy constraints are constraints based on the specific problem instance. Large numbers of them are generated prior to solving, but they are not immediately used in the model. Instead, they are checked against whenever an integral solution is found, and only those that are violated are added to the LP model. In practice, only few of the lazy constraints are used in the solution process. Nevertheless, they can noticeably improve solving time in some cases.

Symmetry-breaking rule This rule creates an explicit, arbitrary preference for certain solutions. Consider two schedules S_1 and S_2. Both schedules contain batches B_h and B_k, both of which are holding their respective host jobs only.

Two jobs j and i are now assigned as the only guests to the two batches; furthermore $\max(p_i, p_j) \leq \min(p_h, p_k)$, $\max(s_h, s_k) + \max(s_j, s_i) \leq b$ and $\min(d_j, d_i) \geq \max(d_h, d_k)$. If $j \in B_h$ and $i \in B_k$ in schedule S_1 and vice versa in S_2, then the constraint renders S_2 infeasible.

$$2(4 - x_{hh} - x_{kk} - x_{jh} - x_{jk} - x_{ih} - x_{ik} + \sum_{\substack{g \\ g \neq j \\ g \neq i}} (x_{gh} + x_{gk})) \geq x_{jk} + x_{ih} \quad \begin{array}{c} \forall \{j, i \in \mathcal{J}, \\ h, k \in \mathcal{B} \\ \mid h < k < j < i \wedge \\ [p_q \leq p_r \wedge b - s_r \geq s_q \\ \forall q \in \{j, i\}, \\ \forall r \in \{h, k\}]\} \end{array} \tag{26}$$

The left-hand side of the equation evaluates to zero exactly when the above conditions are met, which in turn disallows the assignment given on the right. For all other job/batch pairings, the left side evaluates to at least two, which places no constraint on the right hand side.

This kind of symmetry-breaking rule can be extended to $m > 2$ batches, with the number of constraints growing combinatorially with m. Since it takes a constant but appreciable time to generate these constraints prior to solving, we have in our trials kept to the simplest variant shown here, and limited their use to problem instances with $n \geq 50$ jobs.

Dominance rule on required assignments A schedule is not uniquely optimal if a job j is left in its single-EDD batch although there is capacity for it in an earlier batch. This constraint can be expressed logically as: if a job j can be safely assigned to B_k without violating the capacity constraint, then j must be assigned to any earlier batch, or B_k must be empty (or both).

The left side of the above *if-then* statement is written as $(1.0 + b - s_j - \sum_{\substack{i=k \\ i \neq j}}^{n_j} s_i x_{ik})/b$, which evaluates to 1.0 or greater iff s_k plus the sizes of guest jobs in k sum to less than $b - s_j$. The constraint is written as follows:

$$2 - x_{jj} - x_{kk} \geq \left(1.0 + b - s_j - \sum_{\substack{i=k \\ i \neq j}}^{n_j} s_i x_{ik} \right) / b \quad \begin{array}{c} \forall \{j \in \mathcal{J}, k \in \mathcal{B} \\ \mid j > k \wedge p_k \geq p_j \\ \wedge s_k + s_j \leq b\} \end{array} \tag{27}$$

As with the rule above, we have found that only more difficult problems with $n \geq 50$ benefit from these constraints.

4 A New MIP Model

The full novel MIP model we are proposing is defined in Model 4.1.

$$\text{Min.} \quad L_{\max} \tag{28}$$

$$\text{s.t.} \quad \sum_k x_{jk} = 1 \qquad \forall j \in \mathcal{J} \tag{29}$$

$$\sum_j s_j x_{jk} \leq b \qquad \forall k \in \mathcal{B} \tag{30}$$

$$P'_k \geq p_j x_{jk} \qquad \forall \{j \in \mathcal{J}, k \in \mathcal{B} | j \geq k\} \tag{31}$$

$$P'_k \geq p_j \qquad \forall \{j \in \mathcal{J}, k \in \mathcal{B} | j = k\} \tag{32}$$

$$x_{kk} \geq x_{jk} \qquad \forall \{j \in \mathcal{J}, k \in \mathcal{B} | j > k\} \tag{33}$$

$$L_{\max} \geq L_{k,\text{single}} + \sum_{h \leq k} P'_h - p_h(2 - x_{hh}) \qquad \forall k \in \mathcal{B}^\star \tag{34}$$

$$x_{jk} = 0 \qquad \forall \{j \in \mathcal{J}, k \in \mathcal{B} | j < k\} \tag{35}$$

$$(*) \quad \begin{aligned} &2(4 - x_{hh} - x_{kk} - x_{jh} - x_{jk} - x_{ih} - x_{ik} \\ &\quad + \sum_{\substack{g \\ g \neq j \\ g \neq i}} (x_{gh} + x_{gk})) \geq x_{jk} + x_{ih} \end{aligned} \qquad \begin{aligned} &\forall \{j, i \in \mathcal{J}, \\ &h, k \in \mathcal{B} \\ &| \, h < k < j < i \wedge \\ &[p_q \leq p_r \wedge b - s_r \geq s_q \\ &\forall q \in \{j, i\}, \\ &\forall r \in \{h, k\}]\} \end{aligned} \tag{36}$$

$$(*) \quad 2 - x_{jj} - x_{kk} \geq \left(1.0 + b - s_j - \sum_{\substack{i=k \\ i \neq j}}^{n_j} s_i x_{ik}\right) / b \qquad \begin{aligned} &\forall \{j \in \mathcal{J}, k \in \mathcal{B} \\ &| j > k \wedge p_k \geq p_j \\ &\wedge s_k + s_j \leq b\} \end{aligned} \tag{37}$$

Model 4.1. The new MIP model. Constraints marked $(*)$ are lazy constraints.

Constraints (29) and (30) are uniqueness and capacity constraints: batches have to remain within capacity b, and every job can only occupy one batch. Constraints (31) and (32) define the value of P'_k for every batch k as the longest p of all jobs in k, but at least p_k. This is required in (34), which follows the explanation above. Constraints (33) ensure that no job is moved into a hostless batch, i.e. in order to move job j into batch k ($x_{jk} = 1$), job k must still be in batch k ($x_{kk} = 1$). Constraints (35) implement the requirement that jobs are only moved into earlier batches. Constraints (36) and (37) implement the additional lazy constraints described above.

5 Empirical comparison

We empirically compared the performance of the CP model by Malapert et al. and Model 4.1. Both models were run on 120 benchmark instances as in Malapert et al. (i.e. 40 instances of each $n_j = \{20, 50, 75\}$). The benchmarks are generated as specified by Daste [11], with a capacity of $b = 10$ and values for p_j, s_j and d_j distributed as follows: $p_j = U[1, 99]$, $s_j = U[1, 10]$, and $d_j = U[0, 0.1] \cdot \tilde{C}_{\max} + U[1, 3] \cdot p_j$. $U[a, b]$ is a uniform distribution between a and b, and $\tilde{C}_{\max} = \frac{1}{bn} \cdot \left(\sum_{j=1}^{n_j} s_j \cdot \sum_{j=1}^{n_j} p_j\right)$ is an approximation of the time required to process all jobs.

The MIP benchmarks were run using CPLEX 12.5 [23] on an Intel i7 Q740 CPU (1.73 GHz) and 8 GB RAM in single-thread mode, with CPLEX parameters `Probe = Aggressive` and `MIPEmphasis = Optimality` (the latter for $n = 20$ only). The CP was implemented using the Choco solver library [24] and run on the same machine using the same problem instances.[2] Solving was aborted after a time of 3600 seconds (1 hour).

The reference MIP model solves fewer than a third of the instances within the time limit. The branch-and-price model [11] is reported to perform considerably worse than CP [4]. Therefore, neither of the two is included here.

5.1 Results

The overview in Table 1 shows aggregated results that demonstrate the performance and robustness of our new model. As shown in Figure 3, our MIP model performs better overall on instances with $n_j = 20$ and $n_j = 75$, while MIP and CP perform similarly well on intermediate problems ($n_j = 50$).

Wherever an optimal solution was not found, the improved MIP model achieved a significantly better solution quality: out of the 40 instances with $n_j = 75$, 22 were solved to optimality by both CP and MIP, 13 were solved to optimality by the MIP only, and 5 were solved by neither model within an hour. A comparison of solution quality where no optimal schedule was found confirms the robustness of the improved MIP model: as Figure 4 illustrates, the

[2] The authors would like to extend a warm thank-you to Arnaud Malapert for both providing his code and helping us run it.

n_j	optimal soln. found by	instances	solving time [s]		absolute gap	
			CP	MIP	CP	MIP
20	both models	40/40	0.42	0.04	0	0
50	both models	40/40	5.67	4.16	0	0
75	both models	22/40	49.30	52.88	0	0
	CP model only	0/40	—	—	—	—
	MIP model only	13/40	> 3600	139.86	323.46	0
	neither model	5/40	> 3600	> 3600	310.40	25.00

Table 1. Summary of empirical results. Values are geometric means for solving time and arithmetic means for absolute gaps. No relative gaps are given due to negative lower bounds.

Mean (120 instances)	Reference MIP		Improved MIP		Reduction	
	Rows	Cols	Rows	Cols	Rows	Cols
Before presolve	7415.14	3033.81	4291.48	2882.76	−42.1%	−5.0%
After presolve	2209.34	1513.06	754.57	687.13	−65.8%	−54.6%
Reduction	−70.2%	−50.1%	−82.4%	−76.1%	—	—

Table 2. Average numbers of variables and constraints in reference and improved MIP models before and after processing by CPLEX's presolve routines.

gap ($\mathrm{UB}(L_{\max})-\mathrm{LB}(L_{\max})$) is consistently larger in the CP model. This means that even with very difficult problems, our model will often give near-optimal solutions more quickly than Malapert et al.'s CP model.

We further found that the lazy constraints introduced in Section 3.3 did not yield *consistent* benefits across problems; in fact, they doubled and tripled solving times for some instances. On average, however, adding the lazy constraints resulted in a speed gain on the order of about 10%, especially for larger problems ($n \geq 50$).

6 Discussion

One likely contribution to the new model's performance is its reduced size compared to the reference MIP model: while the reference model required $3n^2 + 8n$ constraints over $n^2 + 3n$ variables, our model uses fewer than $2.5n^2 + 2.5n$ constraints over $n^2 + n$ variables.[3] In addition, CPLEX's presolve methods are more effective on our model (see Table 2), reducing its size further.

Figure 5 shows the evolution of bounds (i.e. best feasible solutions and tightest LP solutions at the nodes) for the three models over the first few seconds. It is based on the logs for the 40 instances with $n = 75$, which contain only

[3] The word "fewer" here arises from the problem-specific cardinality of $\mathcal{B}^\star$.

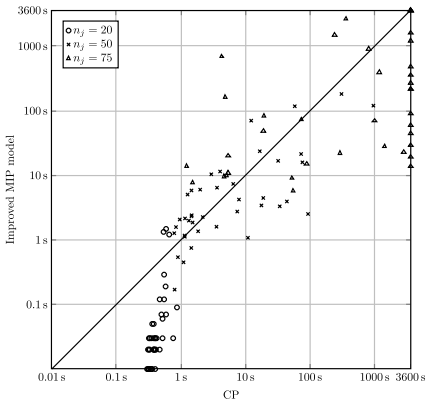

Fig. 3. Performance comparison over 120 instances, each represented by one data point. Horizontal/vertical coordinates correspond to solving time by CP model and improved MIP model, respectively. Note that 18 instances were not solved to optimality within an hour by either the CP model or both models.

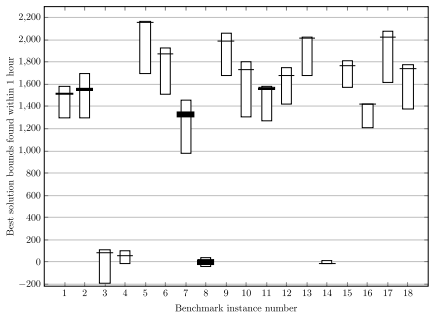

Fig. 4. Comparison of solution quality for the 18 instances that were not solved to optimality within an hour by either the CP model or both models. White bars represent the LB-UB gap achieved by the CP model, black bars the LB-UB gap achieved by the improved MIP model (straight line where solved to optimality).

irregular timestamps. While the new MIP model is better than the reference model on both the lower and upper bounds, the largest gain appears to derive from the latter, indicating that unlike what is commonly observed, the improved MIP model benefits not from a tighter relaxation but from being more amenable to the solver's primal heuristics.

The upper bound for the improved MIP model matches that of the CP model. CP is often able to find high quality solutions faster than MIP. However, the improved model removes that advantage on our test problems.

Making moves in MIP modeling One of the novelties of the new MIP model, as well as much of its inspiration, is the consideration of *moves* from the canonical single-EDD solution. The effects of the assignment variables on the objective function can be considered discretely, allowing us to reason about them algorithmically even though they constitute a declarative model. The concept of moves is common in local search techniques including Large Neighbourhood Search (LNS) [25] where moves correspond to the removal and re-insertion of jobs from and into the schedule, similar to our reasoning in Section 3.2.

This line of reasoning presents several interesting directions for future work including (i) using the idea of moves from a canonical solution to develop MIP and CP models for other optimization problems and (ii) the derivation of dominance rules to restrict LNS moves on large problems and thereby expand the size of the neighbourhoods that can be explored.

Models vs. global constraints Our results show that the novel MIP model is an improvement over previous approaches, demonstrating that at least in

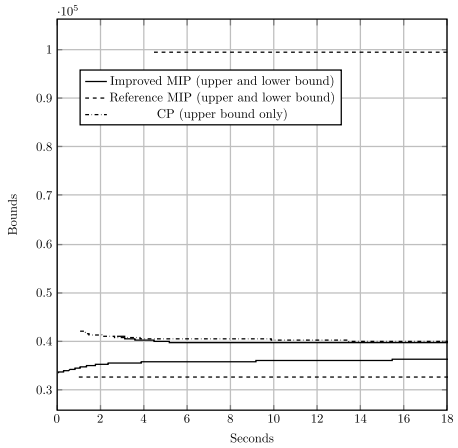

Fig. 5. Evolution of upper and lower bounds. Left cutoffs indicate the approximate mean time at which the respective bound was first found.

this case, the performance of a specialized global constraint implementation can indeed be matched and exceeded by a comparatively simple mathematical formulation. Mathematical models have the general benefit of being more readily understandable, straightforward to implement and reasonably easy to adapt to new, similar problems.

A global constraint is most valuable when it is the encapsulation of a problem structure that occurs across a number of interesting problem types. It can then be used far beyond its original context. However, with the flexibility to define arbitrary inference operations comes the temptation to develop problem-specific global constraints and to trade the ideal of re-usability for problem solving power. We believe that the collection of global constraints in CP is mature enough that most problem-specific efforts are now best placed on exploring novel ways to exploit problem structure using existing global constraints. To this end, one of our current research efforts is the development of a CP model exploiting the propositions proved in this paper without needing novel global constraints.

7 Conclusion

In this paper, we addressed an existing parallel-batch scheduling problem for which CP is the current state-of-the-art approach. Inspired by the idea of moves from a canonical solution, we proved a number of propositions allowing us to create a novel MIP model that, after presolving, is less than half the size of the previous MIP model. Empirical results demonstrated that, primarily due to the ability to find good feasible solutions quickly, the new MIP model was able to out-perform the existing CP approach over a broad range of problem instances both in terms of finding and proving optimality and in terms of finding high quality solutions when the optimal solution could not be proved.

References

[1] Hooker, J.: A hybrid method for planning and scheduling. Constraints **10** (2005) 385–401

[2] Beck, J.C., Feng, T.K., Watson, J.P.: Combining constraint programming and local search for job-shop scheduling. INFORMS Journal on Computing **23**(1) (2011) 1–14

[3] Tran, T.T., Beck, J.C.: Logic-based benders decomposition for alternative resource scheduling with sequence-dependent setups. In: Proceedings of the Twentieth European Conference on Artificial Intelligence (ECAI2012). (2012) 774–779

[4] Malapert, A., Guéret, C., Rousseau, L.M.: A constraint programming approach for a batch processing problem with non-identical job sizes. European Journal of Operational Research **221** (2012) 533–545

[5] Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.: Solving RCPSP/max by lazy clause generation. Journal of Scheduling **16**(3) (2013) 273–289

[6] Baptiste, P., Le Pape, C.: Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. Constraints **5**(1-2) (2000) 119–139

[7] Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-based Scheduling. Kluwer Academic Publishers (2001)

[8] Vilím, P.: Edge finding filtering algorithm for discrete cumulative resources in $O(kn\ log\ n)$. In: Principles and Practice of Constraint Programming-CP 2009. Springer (2009) 802–816

[9] Freuder, E.C.: In pursuit of the holy grail. Constraints **2** (1997) 57–61

[10] Heinz, S., Ku, W.Y., Beck, J.C.: Recent improvements using constraint integer programming for resource allocation and scheduling. In Gomes, C., Sellmann, M., eds.: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Volume 7874 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 12–27

[11] Daste, D., Gueret, C., Lahlou, C.: A branch-and-price algorithm to minimize the maximum lateness on a batch processing machine. In: Proceedings of the 11^{th} International Workshop on Project Management and Scheduling (PMS), Istanbul, Turkey. (2008) 64–69

[12] Lee, C.Y., Uzsoy, R., Martin-Vega, L.A.: Efficient algorithms for scheduling semiconductor burn-in operations. Oper. Res. **40**(4) (July 1992) 764–775

[13] Grossmann, I.E.: Mixed-integer optimization techniques for the design and scheduling of batch processes. Technical Report Paper 203, Carnegie Mellon University Engineering Design Research Center and Department of Chemical Engineering (1992)

[14] Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M.Y., Potts, C.N., Tautenhahn, T., van de Velde, S.L.: Scheduling a batching machine. Journal of Scheduling **1**(1) (1998) 31–54

[15] Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems. 2nd edn. Prentice-Hall (2003)

[16] Shaw, P.: A constraint for bin packing. In Wallace, M., ed.: Proceedings of the 10^{th} International Conference on Principles and Practice of Constraint Programming (CP 2004). Volume 3258 of Lecture Notes in Computer Science. (2004) 648–662

[17] Azizoglu, M., Webster, S.: Scheduling a batch processing machine with non-identical job sizes. International Journal of Production Research **38**(10) (2000) 2173–2184

[18] Dupont, L., Dhaenens-Flipo, C.: Minimizing the makespan on a batch machine with non-identical job sizes: an exact procedure. Computers & Operations Research **29**(7) (2002) 807–819
[19] Sabouni, M.Y., Jolai, F.: Optimal methods for batch processing problem with makespan and maximum lateness objectives. Applied Mathematical Modelling **34**(2) (2010) 314–324
[20] Kashan, A.H., Karimi, B., Ghomi, S.M.T.F.: A note on minimizing makespan on a single batch processing machine with nonidentical job sizes. Theoretical Computer Science **410**(27-29) (2009) 2754–2758
[21] Ozturk, O., Espinouse, M.L., Mascolo, M.D., Gouin, A.: Makespan minimisation on parallel batch processing machines with non-identical job sizes and release dates. International Journal of Production Research **50**(20) (2012) 6022–6035
[22] IBM ILOG: User's manual for cplex (2013)
[23] Ilog, I.: Cplex optimization suite 12.5 (2013)
[24] Choco Team: Choco: An open source java constraint programming library. version 2.1.5 (2013)
[25] Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Proceedings of the Fourth International Conference on Pricinples and Practice of Constraint Programming. Volume 1520 of Lecture Notes in Computer Science. (1998) 417–431