CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

Logic-Based Benders Decomposition for Scheduling a Batching Machine

Simon Emde
Lukas Polten
Michel Gendreau

October 2018

CIRRELT-2018-40

Bureaux de Montréal :
Université de Montréal
Pavillon André-Aisenstadt
C.P. 6128, succursale Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

Logic-Based Benders Decomposition for Scheduling a Batching Machine

Simon Emde[1, *], Lukas Polten[1], Michel Gendreau[2]

[1] Technische Universität Darmstadt, Fachgebiet Management Science / Operations Research, Hochschulstraße 1, 64289 Darmstadt, Germany

[2] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Department of Mathematics and Industrial Engineering, Polytechnique Montréal, P.O. Box 6079, Station Centre-Ville, Montréal, Canada H3C 3A7

Abstract. This paper investigates the problem of scheduling a set of jobs on a single batching machine to minimize the maximum lateness, where jobs may be subject to precedence constraints and incompatibilities. Single batching machine scheduling has many applications, but this study is particularly motivated by single crane scheduling in an automated storage and retrieval system (AS/RS): given a set of transport requests, which requests should be processed together in the same dual command cycle, and in what order should the cycles be processed? Since storage and retrieval requests may refer to the same physical item, precedence constraints must be observed. Moreover, the crane may not be capable of handling multiple storage or retrieval requests in the same cycle, hence the need to account for incompatibilities. We present a novel exact algorithm based on branch & Benders cut, which is shown to solve even large instances with more than 100 jobs to optimality in many cases. For the special case without precedence constraints and incompatibilities, it improves on several best-known upper bounds from the literature.

Keywords: Benders decomposition, single batching machine, automated storage and retrieval, precedence constraints, maximum lateness.

---

* Corresponding author: emde@bwl.tu-darmstadt.de

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2018

1 Introduction

We consider the problem of batching a set of jobs with given processing times and due dates on a single batching machine. Each batch can contain a given maximum number of jobs, which are processed in parallel such that the processing time of the entire batch equals the processing time of the longest job in the batch. Jobs can be incompatible with each other, in which case they must not be assigned to the same batch, and they may be restricted by precedence relations. The goal is to minimize the maximum lateness. In triple notation as originally introduced by Graham et al. (1979), this corresponds to the triple $[1|\text{prec}; \text{incompatible}; \text{batch}(b)|L_{\max}]$.

Batching machine scheduling problems have a multitude of applications in manufacturing industries, particularly chemical, microelectronic, and metalworking. More specifically, special cases of the problem treated in this paper, without precedence constraints and incompatibilities, have been observed in the context of semiconductor burn-in, where burn-in ovens are equivalent to batching machines (Lee et al., 1992). However, this research is originally motivated by an automated storage and retrieval system (AS/RS) we encountered at a major German machine manufacturer.

The system is a unit-load AS/RS served by one crane performing one retrieval and one storage operation in-between returns to the front-end input / output point (dual command cycle). The AS/RS is used to store parts for a closely interfaced assembly system. Parts required in final assembly are requested from the storage system with a certain lead time (between two to four hours in practice) and are presented to one of multiple order picking stations. The parts retrieved from the AS/RS are then packed into standard size bins in so-called kits, sorted to conform to the production sequence. After parts have been picked, partially empty unit-load devices may be returned to storage. Further storage requests may also ensue from arriving parts. One of the biggest challenges involved with operating an AS/RS in the context of just-in-time production is ensuring that parts required in the assembly hall be ready on time. Production and in-house part delivery schedules are fixed, and planning cycles are short, leaving little room for error. In the worst case, if an important part is critically delayed, the entire production line must be halted until the part is available. Consequently, requests should not be delayed past their due date if at all possible.

We can interpret the crane as a batching machine which can process at most two requests (jobs) at a time. Since the total shelf access time for all requests is constant and usually dominated by travel times, the longest processing time (i.e., the farthest travel time) of any request in a command cycle can be an adequate approximation of travel times. However, only storage and retrieval requests can be mixed, since the shuttle can only carry one unit-load device at a time, hence some jobs are incompatible with each other. Moreover, there may be a precedence relation between some requests (jobs), because they actually refer to the same unit-load device. E.g., one request is for fetching a specific pallet, and another is for returning it to its storage location. Obviously, the pallet cannot be returned before it is retrieved.

The contribution of this paper consists of modelling this problem as a batching machine scheduling problem, which is a generalization of the problem considered in Lee et al. (1992) and Cabo et al. (2015). We develop a logic-based Benders decomposition algorithm, which is, to the best of our knowledge, the first exact procedure for this problem. In a computational study, our algorithm com-

pares favorably to the procedures proposed by Cabo et al. (2015). We also generate new test instances from an AS/RS context to study the performance of the algorithm in the face of precedence relations and incompatibilities between jobs.

The remainder of this paper is structured as follows. In Section 2, we review the germane literature. In Section 3, we formally describe the problem and present a MIP model. We investigate a special case that is solvable in polynomial time and develop a novel algorithm based on Benders decomposition for the general case in Section 4. In a computational study (Section 5), we compare our procedure against a default solver and a state-of-the-art procedure from the literature. Finally, Section 6 concludes the paper.

2 Literature review

Batching machine scheduling has a long history, with a great number of papers published over the past decades. Pinedo (2016) gives an overview over machine scheduling problems in general. Batching machine scheduling in particular is surveyed by Potts and Kovalyov (2000) and Zhu and Wilhelm (2006); a meta-analysis of the literature is provided by Abedinnia et al. (2017).

Batching machine scheduling problems are often distinguished by the machine environment and the processing time of the batches. Regarding the former criterion, we consider a single machine environment, and regarding the second, the processing time of a batch equals the longest processing time of any job assigned to it (also referred to as a burn-in model, Brucker et al., 1998). Such machines are commonly encountered in many industrial settings, like semiconductor manufacturing (e.g., Jia et al., 2015) or chemical processing (e.g., Méndez et al., 2006).

The particular single machine batching problem tackled in this paper is a generalization of the problem presented by Cabo et al. (2015), who in turn take up a problem introduced by Lee et al. (1992), who model the burn-in operations at a semiconductor manufacturing plant: before shipping, chips must be stress tested in an oven at a given temperature. Each chip has an individual prescribed burn-in time it must spend in the oven, which may be exceeded but not underrun. Therefore, the processing (i.e., burn-in) time of a given batch of chips depends on the longest prescribed burn-in time of any of the chips in the batch. Since it is critical that chips be ready for delivery on time, scheduling objectives usually revolve around minimizing lateness. Lee et al. (1992) develop polynomial time exact procedures for some special cases of this problem (e.g., agreeable processing times and due dates), as well as prove worst-case performance bounds for some priority rules.

Cabo et al. (2015) propose a mixed-integer programming model for this problem. They solve it heuristically by decomposing it into two stages. In the first stage, the sequence of jobs is determined. In the second stage, for the given job sequence an optimal schedule is created in polynomial time via a dynamic programming based approach. The authors use this dynamic programming scheme to search heuristically in a so-called split-merge neighborhood. We report on their results in more detail in Section 5.

The problem has recently been extended by Cabo et al. (2018) to include a bi-criterial objective function to minimize both the maximum lateness and the number of batches. A MIP model and a

genetic algorithm are presented to find Pareto-optimal solutions. Similarly, Zhou et al. (2018) develop a particle swarm optimization algorithm for a single batching machine problem with lateness objective where the jobs have non-identical sizes (i.e., the maximum number of jobs per batch depends on the size of the jobs) and non-zero release dates.

Apart from applications in production, batching machine scheduling problems can also be applied to different domains. In particular, this research is inspired by an AS/RS, where the shuttle is equivalent to a batching machine and the retrieval and storage requests correspond to jobs. Decision problems concerning AS/RS are surveyed by Roodbergen and Vis (2009) and Gagliardi et al. (2012). Single crane scheduling in AS/RS has recently been surveyed by Boysen and Stephan (2016), who also note the similarities to some machine scheduling problems. The AS/RS literature almost exclusively focuses on a makespan objective; time windows – although quite common in just-in-time systems – are rarely taken into account. Among the few exceptions are Linn and Xie (1993), who present a simulation study, and Lee and Kim (1995), who assume a common due date and develop priority rule based heuristics.

3 Problem description

Let $J = \{1, \ldots, n\}$ be the set of jobs to be processed in batches on a single machine. Between some jobs, there may be precedence relations (e.g., in the AS/RS context, if the same partially empty pallet needs to be first retrieved and then returned to storage). Precedence relations are expressed by tuples $(j, j') \in E$, denoting that job $j' \in J$ must be processed in a batch after job $j \in J$. Moreover, some jobs are incompatible with each other and cannot be assigned to the same batch (e.g., two retrieval or two storage requests cannot be combined). Incompatibilities are expressed by tuples $(j, j') \in I$, indicating that job $j \in J$ and job $j' \in J$ must not be processed in the same batch. Each job $j \in J$ takes p_j time units of processing time and should be finished not later than its due date d_j. Finally, batches must not contain more jobs than the given maximum batch size b.

A schedule consists of a partition of the job set J into r subsets $B_1, \ldots, B_r$ and a permutation $\Sigma = \langle \sigma(1), \ldots, \sigma(r) \rangle$ of index set $\{1, \ldots, r\}$, indicating the order in which the batches are processed on the single machine, i.e., batch $B_{\sigma(k)}$ is the k-th batch to be processed, $\forall k = 1, \ldots, r$. Note that the number of batches r is not given but can obviously not be greater than n. We define $P(B_i) = \max_{j \in B_i}\{p_j\}$ as the processing time of batch B_i, i.e., the longest processing time of any of the jobs assigned to B_i. The completion time of batch $B_{\sigma(k)}$ is $\tau_k = \sum_{k'=1}^{k} P(B_{\sigma(k')})$. For notational simplicity, let $\pi(j) = k$ such that $j \in B_{\sigma(k)}$ be the sequence position of the batch job j is assigned to. We consider a schedule feasible if and only if it satisfies the following conditions.

- The maximum batch size is not exceeded, i.e., for each $i \in \{1, \ldots, r\}$, it must hold that $|B_i| \leq b$.
- Precedence relationships between jobs must be observed, i.e., for each tuple $(j, j') \in E$, it must hold that $\pi(j) < \pi(j')$.

- Incompatible jobs must not be assigned to the same batch, i.e., for each tuple $(j, j') \in I$, it must hold that $\pi(j) \neq \pi(j')$.

Among all feasible schedules, we seek one which minimizes the maximum lateness of jobs, i.e., we minimize

$$L_{\max} = \max_{j \in J} \{\tau_{\pi(j)} - d_j \mid \tau_{\pi(j)} > d_j\}. \tag{1}$$

Note that, in the context of AS/RS, the maximum travel time for a request in a command cycle (i.e., the batch processing time) may only be a lower bound on the actual travel time of the crane, depending on the technical capabilities and layout of the AS/RS. In the literature, the Chebyshev metric is often used to measure distances, because the S/R machine can typically move independently in the vertical and horizontal directions (Van Den Berg, 1999). However, abstracting from technical details like acceleration and deceleration of the crane (Chang et al., 1995) or using surrogate distance objectives (e.g., De Koster et al., 1999, Chen and Wu, 2005) is common in the order batching literature.

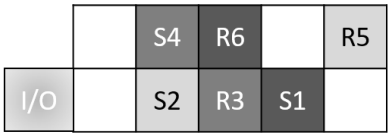
Regarding the computational complexity, single batching machine scheduling with a lateness objective is shown to be NP-hard in the strong sense even if $b = 2$ by Brucker et al. (1998). Since our problem is a generalization, the same complexity status holds.

3.1 Example of a schedule

Consider the example AS/RS schematically depicted in Figure 1b. There are $n = 6$ jobs in total, 3 storage requests (labeled S1, S2, and S4 in the figure) and 3 retrieval requests (R3, R5, and R6). The due dates and processing times are in Table 1a. Moreover, let $E = \{(2,3),(1,5),(5,4)\}$ be precedence constraints, i.e., storage request S2 must be processed before retrieval request R3, S1 before R5, and R5 before S4. Finally, only at most one storage and one retrieval request can be processed per batch, i.e., the batch size is $b = 2$ and $I = \{(S1,S2),(S1,S4),(S2,S4),(R3,R5),(R3,R6),(R5,R6)\}$.

j	1	2	3	4	5	6
p_j	10	6	8	6	12	8
d_j	28	25	21	20	19	18

(a) Example problem data.



(b) Schematic depiction of the AS/RS in the example; slots with same background shade are processed in the same command cycle.

Figure 1: An example problem.

A feasible and optimal solution consists of $r = 3$ batches $B_1 = \{S1, R6\}$, $B_2 = \{S2, R5\}$, $B_3 = \{S4, R3\}$, processed in that order (i.e., $\Sigma = \langle 1, 2, 3 \rangle$). This implies that batch 1 finishes at $\tau_1 = 10$, batch 2 at $\tau_2 = 22$, and batch 3 at $\tau_3 = 30$. Consequently, job $R5$ is late by 3 time units, job $S4$ by 10 and job $R3$ by 9, leading to an objective value of $L_{\max} = \max\{3, 9, 10\} = 10$.

3.2 MIP model

J	set of jobs, $J = \{1, \ldots, n\}$
C	set of batches, $C = \{1, \ldots, n\}$
E	set of precedence relations; $(j, j') \in E$ indicates that job j must be processed in an earlier batch than job j'
I	set of incompatible jobs; $(j, j') \in I$ indicates that the jobs j and j' can not be in the same batch
p_j	processing time of job j
d_j	due date of job j
b	maximum batch size
M	big integer
$x_{c,j}$	binary variable: 1, if job j is assigned to batch c; 0, otherwise
τ_c	continuous variable: completion time of batch c
P_c	continuous variable: processing time of batch c
$L_{\max}$	continuous variable: maximum lateness

Table 1: Notation.

Using the notation summarized in Table 1, we adapt the mixed-integer linear programming model originally proposed by Cabo et al. (2015) as follows.

$$\text{Minimize } f(\boldsymbol{x}, \boldsymbol{\tau}, \boldsymbol{P}, L_{\max}) = L_{\max} \tag{2}$$

subject to

$$\sum_{c \in C} x_{c,j} = 1 \qquad \forall j \in J \tag{3}$$

$$\sum_{j \in J} x_{c,j} \leq b \qquad \forall c \in C \tag{4}$$

$$P_c \geq p_j \cdot x_{c,j} \qquad \forall c \in C, j \in J \tag{5}$$

$$\tau_c = \tau_{c-1} + P_c \qquad \forall c \in C \setminus \{1\} \tag{6}$$

$$\tau_1 = P_1 \tag{7}$$

$$L_{\max} \geq \tau_c - d_j - M \cdot (1 - x_{c,j}) \qquad \forall c \in C, j \in J \tag{8}$$

$$\sum_{c \in C} c \cdot x_{c,j} \leq \sum_{c \in C} c \cdot x_{c,j'} - 1 \qquad \forall (j, j') \in E \tag{9}$$

$$x_{c,j} + x_{c,j'} \leq 1 \qquad \forall c \in C, (j, j') \in I \tag{10}$$

$$x_{c,j} \in \{0, 1\} \qquad \forall c \in C, j \in J \tag{11}$$

$$L_{\max} \geq 0 \tag{12}$$

Objective function (2) minimizes the maximum lateness. Constraints (3) and (4) ensure that each job is assigned to exactly one batch and no batch contains more than b jobs, respectively. Inequalities

(5) set the batch processing times, (6)-(7) set the batch completion times, and (8) set the maximum lateness. Constraints (9) enforce the precedence relations, and Inequalities (10) do not allow assigning incompatible jobs to the same batch. Finally, (11) and (12) define the domain of the variables.

4 Solution methods

4.1 Polynomially solvable special case

While single batching machine scheduling is generally NP-hard if the batch size b is restricted, there is one important special case that can be solved in polynomial time. If there are no precedence relations and incompatibilities, and all jobs have the same common due date, the problem becomes tractable. Note that common due dates are not unusual in settings with fixed planning periods. For example, Lee and Kim (1995) describe an AS/RS in a just-in-time context with such characteristics. Moreover, by setting $D = \max_{j \in J}\{d_j\}$ a solution to this special case may also serve to calculate a lower bound on the optimal objective value of the general problem.

Proposition 4.1. $[1|d_j = D; \textit{batch}(b)|L_{\max}]$ *without precedence relations and incompatibilities, i.e.,* $E = I = \emptyset$*, and with a common due date* $d_j = D$*,* $\forall j \in J$*, can be solved in* $O(n \log n)$ *time.*

Proof. Since all jobs have the same common due date, minimizing the maximum lateness is identical to minimizing the makespan. This corresponds to the problem $[1|\text{batch}(b)|C_{\max}]$, which is shown by Brucker et al. (1998) to be solvable in $O(n \log n)$ time by proceeding as follows. Sort the jobs according to the SPT (shortest processing time) rule. Put the first b jobs together in the first batch B_1 and remove these jobs from consideration. Take the next b jobs in SPT order and put them in the next batch, and so on, until all jobs are added to a batch. Note that this procedure assumes that the number of jobs is a multiple of the batch size, i.e., $n = b \cdot r$. This can always be imposed by adding dummy jobs with zero processing time. The batches can be processed in any arbitrary order as long as there are no waiting times between the batches.

Since sorting the jobs in SPT order takes $O(n \log n)$ time, the proposition holds. □

4.2 Logic-based Benders decomposition

To solve the general case, we propose an exact branch & cut procedure based on Benders decomposition (Benders, 1962). We decompose the problem into two components: in the master problem, we decide on the assignment of jobs to batches but not the sequence of these batches. As such, the master problem is essentially a relaxed version of the original problem. We formulate it as an integer programming model and solve it using a commercial black-box default solver, namely CPLEX 12.8. Whenever the solver finds an integer solution, i.e., an assignment of jobs to batches, the slave problem is solved for this given master solution. The slave problem consists of sequencing the given batches such that all precedence constraints are observed and the maximum lateness is minimal. Subsequently, feasibility and optimality cuts are generated and added to the constraint set of the master model as so-called lazy constraints. The solver then continues solving the master model with the

newly added cuts, iteratively solving the slave problem, until no more feasible, unfathomed solutions remain. The best incumbent solution at this point is optimal.

Unlike classic Benders decomposition, we do not re-solve the master model from scratch whenever a new cut is added, but instead inject cuts into the branch & cut tree as it evolves as lazy constraints. This approach is often called branch & Benders cut (Rahmaniani et al., 2017, Emde, 2017). Moreover, we do not employ classic Benders cuts but instead use combinatorial logic-based cuts in the spirit of Codato and Fischetti (2006) and Hooker (2011).

4.3 Master problem

The master problem is concerned with finding an assignment of jobs to batches. We care at this stage about the order of the batches only implicitly. To formulate this problem concisely as an integer programming model, without loss of generality, we assume that jobs are ordered according to non-increasing due date. We introduce binary variables $y_{j,j'}$, which take the value 1 if and only if job $j \in J$ is assigned to the same batch as job $j' \in J$ and the earliest due date of the batch is $d_{j'}$. Note that this implies that if $y_{j,j'} = 1$, then $d_j \geq d_{j'}$. We refer to the earliest due date of the jobs in some batch i as the *batch due date* of batch i. Moreover, we introduce auxiliary continuous variables ρ_j, which represent the batch processing time of the batch that contains job j as the most critical (earliest due date) job, i.e., the maximum processing time of any job in the batch whose batch due date is d_j. Finally, auxiliary continuous variable α is a lower bound on the lateness objective.

Since we do not consider the exact batch sequence in the master model, we cannot calculate the exact objective value, that is, the maximum lateness $L_{\max}$. It is possible to solve the master model as a pure feasibility problem, i.e., without any (meaningful) objective at all. However, without objective, the solver has no way of evaluating solutions; the solution process would hence resemble the search for the proverbial needle in a haystack. Therefore, we introduce minimizing auxiliary variable α as a surrogate objective. Variable α equals the maximum difference between the completion time of a batch, assuming that batches are processed in the order of non-decreasing batch due date, and the earliest due date of the batch. Note that in the absence of precedence constraints, an earliest due date ordering (EDD) of batches is optimal (Lawler, 1973). However, an EDD ordering of batches may be infeasible if the precedence relations do not allow it, hence α is merely a lower bound on the actual optimal lateness, which can only be determined by solving the slave problem described in Section 4.5. Formally, our master model is as follows.

$$\text{[Master] Minimize } \alpha \tag{13}$$

subject to

$$\sum_{j' \in J} y_{j,j'} = 1 \qquad \forall j \in J \tag{14}$$

$$\sum_{\substack{j\in J:\\ j<j'}} y_{j,j'} \leq (b-1)y_{j',j'} \qquad \forall j' \in J \tag{15}$$

$$y_{j,j'} \leq y_{j',j'} \qquad \forall j,j' \in J \tag{16}$$

$$y_{j,j'} = 0 \qquad \forall j,j' \in J : j > j' \vee (j,j') \in E \vee (j,j') \in I \vee (j',j) \in E \vee (j',j) \in I \tag{17}$$

$$y_{j,j''} + y_{j',j''} \leq 1 \qquad \forall j,j',j'' \in J : (j,j') \in E \vee (j,j') \in I \tag{18}$$

$$\rho_{j'} \geq p_j \cdot y_{j,j'} \qquad \forall j,j' \in J \tag{19}$$

$$\sum_{\substack{j\in J:\\ j<j'}} p_j y_{j,j'} \leq (b-1)\rho_{j'} \qquad \forall j' \in J \tag{20}$$

$$\alpha \geq \sum_{\substack{j\in J:\\ j\geq j'}} \rho_j - d_{j'} \qquad \forall j' \in J \tag{21}$$

$$y_{j,j'} \in \{0,1\} \qquad \forall j,j' \in J \tag{22}$$

$$\alpha \geq 0 \tag{23}$$

Objective function (13) minimizes the maximum difference between batch completion time in EDD order and batch due date over all batches as a substitute objective, as expressed by auxiliary variable α. Constraints (14) ensure that each job j is assigned to exactly one batch (whose batch due date is $d_{j'}$). Inequalities (15) make batches of more than b jobs impossible. Valid inequalities (16) enforce that if a job j is assigned to the same batch as a job j', then j' must also be assigned to that batch. Note that if $y_{j,j} = 1$, job j is the job with the earliest due date of a batch. For each batch, there is only one such job. This is already implicitly enforced by Constraints (15); however inequalities (16) tighten the LP relaxation. Eqs. (17) imply that the job with the earliest due date of a batch is the one for which $y_{j,j} = 1$, and that incompatible jobs or jobs that precede each other are not in the same batch. Constraints (18) make it impossible for two incompatible jobs to be assigned to the same batch. If some job j' is the job with the earliest due date in its batch, then Inequalities (19) set auxiliary variable $\rho_{j'}$ to the batch processing time of that batch. Valid inequalities (20) also set variables $\rho_{j'}$ and are redundant but tighten the LP relaxation and consequently accelerate convergence. By Inequalities (21), auxiliary variable α assumes the value of the greatest difference between the completion time of a batch (under the assumption that batches are processed in EDD order) and the most critical due date of that batch. Finally, (22) and (23) define the domain of the variables.

Note that model [Master] is more compact than the original MIP model from Section 3.2 in terms of variable count. Moreover, several complicated constraints are removed, especially the difficult "big M" Constraints (8), which pose notoriously great problems for solvers (Codato and Fischetti, 2006). The model, however, relaxes the precedence constraints; master solutions may therefore be infeasible and / or suboptimal. We describe how to derive a batch sequence from a master solution in Section 4.5.

4.4 Warm starting for the master model

Preliminary tests show that the master model presented above sometimes struggles to find a first feasible solution, especially in the face of complicated precedence relations, which are only modelled in a rudimentary fashion in the master model. Since feasibility cuts regarding precedence violations are only added in the slave problem (see next Section 4.5), this may lead to some inefficiency. Therefore, to accelerate the solution process, we warm start CPLEX using several feasible initial solutions, which we obtain via the following constructive heuristic.

Let $S = \langle s_1, \ldots, s_n \rangle$ be a permutation of the job set J, i.e., a sequence of jobs. We assign the first job in S ("first" according to the given sequence) whose predecessors have already been assigned in previous iterations to the current batch (initially batch $B_1 := \emptyset$). Then we remove this job from S and start again from the beginning. If the current batch is full (i.e., contains b jobs) or there are no assignable jobs whose predecessors have already been processed, the current batch is closed and a new batch is started. The entire procedure is outlined in Algorithm 1.

Algorithm 1: Assigning jobs to batches for given sequence S.

Input: job sequence S

$\tilde{J} := J$;
$i := 1$;
$B_1 := \emptyset$;
while $\tilde{J} \neq \emptyset$ **do**
  if $|B_i| = b$ **or** *there are no assignable jobs in* $\tilde{J}$ **then**
    $i := i + 1$;
    $B_i := \emptyset$;
  for $j := 1$ **to** $|\tilde{J}|$ **do**
    if *all predecessors of job* s_j *are already assigned to an earlier batch* **then**
      $B_i := B_i \cup \{s_j\}$;
      $\tilde{J} := \tilde{J} \setminus \{s_j\}$;
      remove job s_j from sequence S;
      break;
return *feasible solution* $\langle B_1, \ldots, B_i \rangle$;

We generate 11 different sequences and consequently 11 different warm start solutions by considering the following sequences, where ties are broken randomly.

- Sort jobs according to non-decreasing processing time.
- Sort jobs according to non-decreasing due date.
- For $\gamma = 2, \ldots, 10$, sort jobs according to non-decreasing due date, then sort each subsequence $\langle s_{n \cdot k/\gamma+1}, \ldots, s_{n \cdot (k+1)/\gamma} \rangle$, $k = 0, \ldots, \gamma - 1$, according to non-decreasing processing time.

4.5 Slave problem

The master problem determines the individual batches. The slave problem consists of finding finding the optimal $L_{\max}$ value for the given master solution by sequencing the given batches. When solving the slave problem, the assignment of jobs to batches is already known. Let $\bar{\boldsymbol{y}}$ be the current master solution. The total number of batches is $r = |\{j \in J \mid \bar{y}_{j,j} = 1\}|$. For each job $j' \in J$ where $\bar{y}_{j',j'} = 1$, the corresponding batch is $B_i = \{j \in J \mid \bar{y}_{j,j'} = 1\}$, $i = 1, \ldots, r$.

4.5.1 Feasibility cuts

The master model does not consider precedence relations beyond avoiding assigning dependent jobs to the same batch. Hence it is possible that the job-batch assignment is inherently infeasible due to cycles. For example, if job j is a predecessor of job j', and j' is a predecessor of job j'', and jobs j and j'' are assigned to the same batch, then regardless of how the batches are sequenced the solution can never be feasible. To detect such cycles, we employ Tarjan's strongly connected components algorithm (Tarjan, 1972).

Let $\tilde{J}(i, i') = \{j \in B_i \mid \bar{y}_{j,j} = 0 \wedge \exists j' \in B_{i'} : (j, j') \in E \vee (j', j) \in E\}$ be the set of jobs from batch B_i which are the predecessors or successors of at least one job in batch $B_{i'}$. Note that we need not consider jobs j where $\bar{y}_{j,j} = 1$ because cycles can only exist due to bad *combinations* of jobs. If a single job j in a batch (i.e., $\bar{y}_{j,j} = 1$) already causes a cycle, then the instance as whole is infeasible. Let $G(V, W)$ be a directed graph, where $V = \{1, \ldots, r\}$ is the set of vertices and W the set of arcs. For each of the r batches, there is one vertex in V. There is an arc $(i, i') \in W$, $i, i' \in V$, $i \neq i'$, if and only if there exist a job $j \in B_i$ and a job $j' \in B_{i'}$ such that $(j, j') \in E$.

Running Tarjan's algorithm on G, we get the set of strongly connected components $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$, where each element $\gamma_k \subseteq V$ is a linearly ordered set denoting one strongly connected component of G in reverse topological order. Since we are only interested in cycles, we can remove all elements from Γ which only contain one vertex, i.e., we are only interested in $\tilde{\Gamma} = \{\gamma \in \Gamma \mid |\gamma| > 1\}$. If $\tilde{\Gamma} = \emptyset$, the graph does not contain any cycles, and it is therefore possible to feasibly sequence the r batches from the master solution. If $\Gamma \neq \emptyset$, we add the following combinatorial feasibility cuts to the constraint set of model [Master]. Let $\gamma(l)$ be the l-th element of cycle γ as found by Tarjan's algorithm in reverse order (i.e., in correct topological order). Then,

$$\sum_{l=1}^{|\gamma|-1} \sum_{j \in \tilde{J}(\gamma(l),\gamma(l+1))} \sum_{\substack{j' \in J: \\ \bar{y}_{j,j'}=1}} (1 - y_{j,j'}) + \sum_{j \in \tilde{J}(\gamma(|\gamma|),\gamma(1))} \sum_{\substack{j' \in J: \\ \bar{y}_{j,j'}=1}} (1 - y_{j,j'}) \geq 1, \qquad \forall \gamma \in \tilde{\Gamma}. \tag{24}$$

The idea behind Cuts (24) is that at least one of the jobs that is in one of the critical batches in γ and is either the predecessor or the successor of another batch in the cycle must be reassigned.

Regarding the time complexity, Tarjan's algorithm has a worst-case performance of $O(|V| + |W|)$. In our problem, the number of vertices is bounded by $O(n)$, and the number of edges by $O(n^2)$, hence the feasibility cuts can be generated in $O(n^2)$ time.

Example (cont.): Consider the example from Section 3.1. Assume that the current master solution

is $\bar{y}_{1,4} = \bar{y}_{4,4} = \bar{y}_{2,5} = \bar{y}_{5,5} = \bar{y}_{3,6} = \bar{y}_{6,6} = 1$ (all other master variables equal 0), corresponding to three batches $B_1 = \{1, 4\}$, $B_2 = \{2, 5\}$, and $B_3 = \{3, 6\}$. Since job 1 is a predecessor of job 5, and job 5 is a predecessor of job 4, there is no feasible sequence for these three batches. The graph G corresponding to this master solution is depicted in Figure 2. Tarjan's algorithm gives us $\Gamma = \{\langle 3\rangle, \langle 2, 1\rangle\}$, i.e., there is one cycle $\gamma = \langle 1, 2\rangle$. Consequently, the following to cut is generated: $(1 - y_{1,4}) \geq 1$, enforcing that jobs 1 and 4 must not be assigned to the same batch anymore, which eliminates this particular cycle.

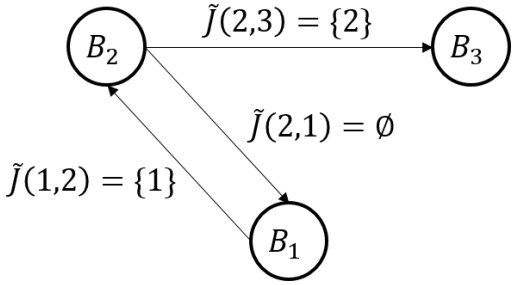


Figure 2: Graph G in the example.

4.5.2 Optimality cuts

If $\tilde{\Gamma} = \emptyset$, there is at least one sequence of batches that does not violate the precedence constraints and hence is feasible. Since the assignment of jobs to batches is already given by the master solution, finding such a sequence which minimizes the maximum lateness is equivalent to scheduling the set of batches $\{1, \ldots, r\}$ with given processing times $\tilde{p}_i = P(B_i)$ and due dates $\tilde{d}_i = \min_{j \in B_i}\{d_j\}$, $\forall i = 1, \ldots, r$, on a single machine to minimize the maximum lateness. Essentially, the batches become jobs with given $\tilde{p}_i$ and $\tilde{d}_i$, to be scheduled on a single machine. In machine scheduling notation, this corresponds to $[1|\text{prec}|L_{\max}]$. Note that at this stage we do not schedule individual jobs but entire batches of jobs.

Problem $[1|\text{prec}|L_{\max}]$ is a classic machine scheduling problem, which can be solved in $O(n^2)$ time using Lawler's algorithm (Lawler, 1973). Let $\Sigma^* = \langle \sigma(1), \ldots, \sigma(r)\rangle$ be the optimal sequence of batches as output by Lawler's algorithm, and let $L^*_{\max}$ be the corresponding optimal objective value. Moreover, let UB be the objective value of the best currently known feasible solution, i.e., the current upper bound, which can initially be set to $UB := \infty$. If $L^*_{\max} < UB$, a new best solution has been found, which is stored, and the upper bound is updated to $UB := L^*_{\max}$. Moreover, we add the following cut to model [Master].

$$\alpha \leq UB - \epsilon, \tag{25}$$

where ϵ is a sufficiently small number greater than 0. The left-hand term of Inequality (25) serves as a lower bound on the optimal objective value. By adding this cut, we exclude all solutions from consideration whose lower bound is not less than the current best upper bound.

Regardless of whether a new upper bound has been found, we determine

$$k^* = \arg\min_{k \in \{1,\ldots,r\}} \left\{ \sum_{k'=1}^{k} \tilde{p}_{\sigma(k')} - \tilde{d}_{\sigma(k)} \geq UB \right\}$$

as the first batch whose lateness matches of exceeds the upper bound. To lower the upper bound (i.e., find a better solution), at least one of the following conditions must be met.

- The batch processing time $\tilde{p}_{\sigma(k)}$ or the batch due date $\tilde{d}_{\sigma(k)}$ of at least one batch $k \in \{1, \ldots, k^*\}$ must change. Formally, let $\underline{j}_k = \arg\min_{j \in B_{\sigma(k)}} \{d_j\}$ be the job that determines the batch due date of batch $\sigma(k)$, and let $\overline{j}_k = \arg\max_{j \in B_{\sigma(k)}} \{p_j\}$ be the job that determines the batch processing time of batch $\sigma(k)$. Then $y_{\overline{j}_k, \underline{j}_k}$ must be forced to zero for at least one $k \in \{1, \ldots, k^*\}$ for this condition to hold. Note that $\overline{j}_k$ may be equal to $\underline{j}_k$.
- A job which is a predecessor of a job in the critical batch k^* is reassigned to a different batch. Formally, let $\bar{E}_k = \{j \in B_{\sigma(k)} \mid \exists j' \in B_{\sigma(k^*)} : (j, j') \in E\}$ be the set of jobs in batch $\sigma(k)$ which have a successor in the critical batch. Then, for at least one $k \in \{1, \ldots, k^* - 1\}$, $y_{j,\underline{j}_k} = 0$, for some $j \in \bar{E}_k$, must hold.
- Analogously, a job which is a successor of a job in the critical batch k^* is reassigned to a different batch. Formally, let $\underline{E}_k = \{j' \in B_{\sigma(k)} \mid \exists j \in B_{\sigma(k^*)} : (j, j') \in E\}$ be the set of jobs in batch $\sigma(k)$ which have a predecessor in the critical batch. Then, for at least one $k \in \{k^* + 1, \ldots, r\}$, $y_{j,\underline{j}_k} = 0$, for some $j \in \underline{E}_k$, must hold.

We enforce this by adding the following cut to program [Master]:

$$\sum_{k=1}^{k^*} (1 - y_{\overline{j}_k, \underline{j}_k}) + \sum_{k=1}^{k^*-1} \sum_{j \in \bar{E}_k} (1 - y_{j,\underline{j}_k}) + \sum_{k=k^*+1}^{r} \sum_{j \in \underline{E}_k} (1 - y_{j,\underline{j}_k}) \geq 1. \tag{26}$$

Example (cont.): Consider a current master solution $\bar{y}_{3,4} = \bar{y}_{4,4} = \bar{y}_{2,5} = \bar{y}_{5,5} = \bar{y}_{1,6} = \bar{y}_{6,6} = 1$ (all other master variables equal 0), corresponding to three batches $B_1 = \{3, 4\}$, $B_2 = \{2, 5\}$, and $B_3 = \{1, 6\}$. Lawler's algorithm yields the optimal sequence $\Sigma^* = \langle 3, 2, 1 \rangle$ with objective value $L_{\max} = UB = 10$. This corresponds to the solution depicted in Figure 1b. The "critical batch" sequence position is $k^* = 3$, because the third batch $B_{\sigma(3)}$ in Σ^* contains the job $j = 4$ whose due date is missed by $L_{\max} = 10$ time units. Hence, the generated cut is

$$(1 - y_{1,6}) + (1 - y_{5,5}) + (1 - y_{3,4}) + (1 - y_{2,5}) + (1 - y_{5,5}) \geq 1.$$

5 Computational study

5.1 Benchmark instances and computational environment

Since our problem is a generalization of the batching machine scheduling problem solved by Cabo et al. (2015), we reuse the same instances. This data set consists of 90 small instances ($n = 20$ jobs),

120 medium-size instances ($n = 50$), and 150 large instances ($n = 100$). Each of these three problem classes is made up of blocks of 10 instances for each combination of b and λ, where b is the maximum batch size, and λ determines the tightness of the due dates. For each job, the due date is a randomly drawn (uniform distribution) integer from the interval $[1; (\lambda/b) \cdot \sum_{j \in J} p_j]$; hence, the lower λ, the tighter the time windows. The parameter ranges used by Cabo et al. (2015) are summarized in Table 2.

symbol	description	values		
n	number of jobs	20	50	100
b	maximum batch size	2, 3, 4	2, 5, 10, 25	5, 10, 15, 25, 50
λ	tightness of due dates	0.5, 1, 1.5	0.5, 1, 1.5	0.5, 1, 1.5

Table 2: Parameter ranges of the instances from Cabo et al. (2015).

From the AS/RS context, to the best of our knowledge, there are no established test data considering precedence constraints and incompatibilities. Therefore, to generate data for the AS/RS case, which distinguish between storage and retrieval requests, we adapt the instance generation scheme of Cabo et al. (2015) as follows. For each job $j \in J$, we draw the processing time as a uniformly distributed random integer from the interval $[0; 100]$, and the due date from $[1; (\lambda/b) \cdot \sum_{j \in J} p_j]$. Moreover, for each job j, we randomly decide (0.5 probability) if it is a storage or a retrieval job. All storage jobs are pairwise incompatible with each other, as are all retrieval jobs, because we assume the shuttle can only carry one item at a time, which is the most common system configuration in practice (Boysen and Stephan, 2016).

We create precedence constraints by generating a random permutation $S = \langle s_1, \ldots, s_n \rangle$ of jobs. We consider every pair of distinct sequence positions $k, k' \in \{1, \ldots, n\}$, $k \neq k'$. If $d_{s_k} < d_{s_{k'}}$ and $k - k' > \delta \cdot n$, s_k and $s_{k'}$ switch positions. This way, by adjusting parameter δ, we can determine how much the precedence relations and the due dates disagree: a low value of δ implies that jobs with an early due date tend to have few predecessors, which makes it easier to not violate due dates, and vice versa.

For each pair of distinct jobs in the sequence, the job that comes later in the sequence is a successor of the job that comes earlier in the sequence with a certain probability, which is adjusted such that each job is expected to have either $\rho = 0.5$, 2, or 4 successors, depending on ρ. The parameters for the newly generated instances are summarized in Table 3. For each parameter constellation, we generate one instance, yielding 3 (different n) $\cdot$ 3 (different λ) $\cdot$ 2 (different δ) $\cdot$ 3 (different ρ) $= 54$ instances. They are labeled according to the scheme $n\_\lambda\_\delta\_\rho$. The instances can be downloaded using the following DOI: 10.5281/zenodo.1446439

For the instances from the literature, we compare our B&BC scheme to the best available algorithm, which is the iterated descent heuristic using a split-merge neighborhood proposed by Cabo et al. (2015) (hereafter referred to as CABO). To establish a fair comparison, we use the same metrics as the original paper to evaluate our solutions, namely the relative gap to a lower bound, calculated as $(f - LB)/LB \cdot 100\%$, where f is the objective value under consideration. Analogous to the original paper, the lower bound LB is derived by the SPT-EDD-dynamic batch schedule, which is based on

symbol	description	values
n	number of jobs	20, 100, 200
λ	tightness of due dates	0.5, 1, 1.5
δ	degree of disagreement between due dates and precedence relations	0.125, 0.5
ρ	expected number of successors per job	0.5, 2, 4

Table 3: Parameters for instance generation

disassociating due dates and processing times (Possani, 2001). Moreover, we calculate the relative improvement over a simple EDD schedule: $(f^{\text{EDD}} - f)/f \cdot 100\%$, where f^{EDD} is the objective value of the EDD schedule.

Cabo et al. (2015) implement their algorithm in C and run the tests on a system with an Intel Core i7 CPU clocked at 2.8 GHz, whereas we use an x64 PC equipped with a 4 GHz Intel i7-6700K CPU and 64 GB of RAM. We implement B&BC in C# 7.0 and use CPLEX 12.8 as a default solver to solve the mixed-integer programming models. We set a time limit of 30 CPU minutes for B&BC and CPLEX.

5.2 Computational results

5.2.1 Instances from the literature

In the first part of our computational study, we compare B&BC against the best solution method from the literature by Cabo et al. (2015), for the special case where there are no precedence constraints or incompatibilities, i.e., the classic single batching machine scheduling problem to minimize the maximum lateness. We compare B&BC to the best results from the original paper, using the same instances. Results are averaged as in the original paper.

	CABO		B&BC		
	opt. gap (%)	# optimal	opt. gap (%)	# optimal	CPU sec.
$\lambda = 1.5$	1.03	28	0	30	0.1
$\lambda = 1$	2.45	18	0	30	0.1
$\lambda = 0.5$	1.89	14	0	30	0.3
avg.	1.79	60	0	90	0.2

Table 4: Algorithmic performance on small instances from the literature ($n = 20$).

Table 4 shows the results for the small literature instances ($n = 20$ jobs). Our B&BC scheme solves all 90 small instances to optimality in negligible time (0.2 seconds on average). Cabo et al. (2015)'s heuristic is even faster (the authors report 0.016 seconds per instance), but produces an average relative optimality gap of 1.79%. Note that Cabo et al. (2015) report that they can find all optimal solutions for the 90 small instances by increasing the time limit for their heuristic to 0.48 seconds.

Table 5 lists the same data for the medium-size instances. For many of these problems, Cabo et al. (2015) were not able to obtain optimal results and consequently do not print optimality gaps in their

		CABO			B&BC			
λ	b	gap UB (%)	gap LB (%)	# opt	gap UB (%)	gap LB (%)	# opt	CPU sec.
1.5	2	23.4	63.8	4	33.4	63.8	10	0.1
1.5	5	44.4	22.1	3	44.5	22.1	10	7.5
1.5	10	32.7	134.8	1	33.4	13.5	10	1.8
1.5	25	9.6	12.6	3	10.5	11.4	10	0.6
1	2	53.7	502.7	0	55.5	468.6	10	1.1
1	5	52.2	120.3	0	56.1	102.2	6	869.4
1	10	41.4	60	0	43.5	54.6	10	37.9
1	25	9.6	30.3	0	29.8	10.0	10	0.7
0.5	2	26.8	854.5	0	27.6	850.2	10	1.1
0.5	5	39.6	304	0	41.1	294.6	0	1829.2
0.5	10	35.6	143.2	0	37.5	136.2	6	1067.3
0.5	25	12.9	49.9	0	13.2	49.6	10	0.3
	avg.	31.8	191.5	0.9	35.5	173.1	8.5	318.1

Table 5: Algorithmic performance on medium-size instances from the literature ($n = 50$).

paper. Therefore, we compare our results using the same metrics as in the original paper, namely the relative gap to the lower bound (column "gap LB") and the relative improvement over the EDD schedule (column "gap UB"). Note that for the former, a lower gap is better, whereas for the latter, a greater gap is better. We also list the number of times the proven optimal solution is found. For CABO, we reproduce the data as reported in the original paper, for B&BC a solution is proven optimal if the algorithm terminates gracefully within its 1,800 second time limit.

Regarding the results, B&BC solves most medium-size instances to optimality within a few seconds of CPU time. The only problems it somewhat struggles with are those where the batch sizes are intermediate, especially a batch size of $b = 5$ seems to be tricky. This makes sense since there are more options, combinatorially speaking, of grouping jobs in batches if the batches are neither very small nor very large. Still, even in these cases where B&BC exceeds the time limit, the solutions are better than those reported by Cabo et al. (2015) in terms of average LB / EDD gaps. Concerning the running times, the average CPU time of CABO per instance is reported as about 24 seconds, whereas the average CPU time of B&BC is about 318 seconds. Note, however, that most individual instances are actually solved by B&BC in less than 2 seconds.

		CABO			B&BC			
λ	b	gap UB (%)	gap LB (%)	# opt	gap UB (%)	gap LB (%)	# opt	CPU sec.
1.5	5	60.9	43.2	2	60.5	48.6	7	592.7
1.5	10	55.7	9.4	5	55.7	9.4	10	48.6
1.5	15	47.9	14.5	2	50.0	9.8	10	134.1
1.5	25	25	19	0	27.5	15.2	10	168.3
1.5	50	7.5	15.5	0	9.7	12.8	10	9.1
1	5	62.1	203.3	0	61.2	209.2	0	1809.8
1	10	53.1	126.8	0	49.5	144.9	0	1809.9
1	15	46.8	96.9	0	46.9	96.8	0	1802.3
1	25	31.5	60.6	0	32.5	58.3	3	1541.7
1	50	8.5	39.2	0	11.6	34.5	10	18.4
0.5	5	42.9	604.8	0	44.0	592.0	0	1813.4
0.5	10	43.9	299.5	0	43.8	300.4	0	1817.9
0.5	15	41.9	197.3	0	42.5	194.5	0	1813.8
0.5	25	31.3	117.9	0	32.0	115.9	0	1804.7
0.5	50	14.9	48.9	0	16.4	46.4	10	4.8
	avg.	38.3	126.5	0.6	38.9	125.9	4.7	1012.6

Table 6: Algorithmic performance on large instances from the literature ($n = 100$).

Regarding the large instances with $n = 100$ jobs (Table 6), B&BC manages to solve 70 out of the 150 instances to optimality within the time limit. While there are some instances that can be solved quite quickly (especially those with very large batch sizes), the average solution time is slightly less than 17 minutes. The quality of the solutions is on average slightly better than what Cabo et al. (2015) report. The average runtime of CABO is given as 8.4 minutes. This indicates that B&BC, while being quite successful at finding optimal solutions, may also serve as a passable heuristic, at least if CPU times are not supremely important.

5.2.2 New instances from AS/RS context

Since the literature has so far only looked at the problem as a single batching machine scheduling problem, we also test the more complicated generalization tailored to the AS/RS use case, where jobs can be either storage or retrieval requests such that precedence relations and incompatibilities must be observed. Note that, to create more of a challenge for B&BC, we also increase the maximum number n of jobs to 200 for these instances, double the size of the largest instances from the literature.

The results for all 54 generated instances are in Table 7, comparing CPLEX, solving the undecomposed original MIP model from Section 3.2, a constructive heuristic, and our B&BC. The constructive heuristic is the same we use to generate warm start solutions as described in Section 4.4. We use the best result from the 11 generated sequences. The time limit for CPLEX and B&BC is set to 1,800 CPU seconds, while the constructive heuristic takes negligible time for all instances. Optimal results are bold in the table. For B&BC, the table also shows the total number of generated cuts per instance.

In the AS/RS case, B&BC performs quite well overall. It finds the optimal solution in all but 14 out of the 54 instances. Even when B&BC does not prove optimality, the best upper bound is better than CPLEX's, sometimes by several orders of magnitude. In all cases, B&BC matches or improves the best CPLEX objective value. On average over all instances, B&BC yields an improvement of about 43.5% over CPLEX, calculated as $(f^{\text{CPLEX}} - f^{\text{B\&BC}})/f^{\text{CPLEX}}$, while being a lot faster: CPLEX reaches the time limit in all but five instances, B&BC only needs less than 9 minutes per instance on average. Considering only the largest instances with $n = 200$ jobs, B&BC can solve almost half of them in less than 2 minutes per instance. This clearly indicates that the B&BC scheme is quite useful for the AS/RS case, too.

6 Conclusion

We investigate the problem of scheduling a set of jobs on a single crane in an AS/RS subject to incompatibilities and precedence constraints such that the maximum lateness is minimal. This is a generalization of single batching machine scheduling. We propose a novel branch & Benders cut scheme to solve this problem.

Our computational tests show that B&BC compares favorably to the best solution method from the literature on the simpler special case without precedence relations and incompatibilities. It improves many best known upper bounds and finds optima that were heretofore unknown. On newly generated larger instances from the AS/RS context, the algorithm also performs well, solving most instances to

	CPLEX		heuristic	B&BC		
ID	f	CPU sec.	f	f	CPU sec.	# cuts
20_0.5_0.125_0.5	**316**	1801.7	402	**316**	0.2	19
20_0.5_0.125_2	**456**	1802.6	483	**456**	0.2	17
20_0.5_0.125_4	**370**	1801.9	488	**370**	0.3	145
20_0.5_0.5_0.5	**240**	1801.6	247	**240**	0.1	20
20_0.5_0.5_2	**322**	1800.8	376	**322**	0.2	18
20_0.5_0.5_4	**345**	1800.9	460	**345**	0.3	128
20_1_0.125_0.5	**173**	1800.8	264	**173**	0.2	21
20_1_0.125_2	**210**	1802.0	301	**210**	0.1	15
20_1_0.125_4	**227**	1800.6	299	**227**	0.2	122
20_1_0.5_0.5	**206**	374.2	264	**206**	0.2	18
20_1_0.5_2	**174**	1801.9	216	**174**	50.6	10163
20_1_0.5_4	**228**	1802.3	335	**228**	1.4	1426
20_1.5_0.125_0.5	**99**	255.0	221	**99**	0.1	15
20_1.5_0.125_2	**0**	0.7	72	**0**	0.1	19
20_1.5_0.125_4	**43**	7.1	127	**43**	0.2	130
20_1.5_0.5_0.5	**94**	53.5	125	**94**	0.2	133
20_1.5_0.5_2	**72**	1800.9	231	**72**	1.1	954
20_1.5_0.5_4	**132**	1801.8	132	**132**	11.2	5272
avg.	205.9	1339.5	280.2	205.9	3.7	1035.3
100_0.5_0.125_2	2533	1801.7	2464	**1910**	6.7	24
100_0.5_0.125_4	2196	1801.7	2250	**1754**	18.3	770
100_0.5_0.5_0.5	2223	1801.6	2278	**1869**	7.6	263
100_0.5_0.5_2	2072	1801.7	1764	**1338**	7.1	24
100_0.5_0.5_4	1922	1801.7	2109	**1536**	7.0	30
100_1_0.125_0.5	2117	1801.7	1981	**1399**	9.8	595
100_1_0.125_2	1232	1801.6	725	**368**	8.8	23
100_1_0.125_4	920	1801.3	901	**280**	37.7	1824
100_1_0.5_0.5	1026	1801.6	1446	504	1803.2	32683
100_1_0.5_2	1214	1801.6	954	432	1801.6	34414
100_1_0.5_4	1033	1801.4	830	246	1849.9	6804
100_1.5_0.125_0.5	1266	1801.5	1096	642	1802.8	43314
100_1.5_0.125_2	1235	1801.6	631	**139**	5.9	16
100_1.5_0.125_4	788	1801.6	621	**449**	6.6	105
100_1.5_0.5_0.5	213	1801.7	496	**0**	6.4	57
100_1.5_0.5_2	670	1801.6	478	**77**	6.0	17
100_1.5_0.5_4	373	1801.7	366	**17**	6.3	45
100_0.5_0.125_0.5	936	1801.8	1033	718	1802.3	38022
avg.	1331.6	1801.6	1245.7	**759.9**	510.8	8835.0
200_0.5_0.125_4	5643	1808.7	3708	**3200**	61.2	71
200_0.5_0.5_0.5	7635	1827.4	4057	**2817**	75.2	433
200_0.5_0.5_2	4778	1810.8	4321	3114	1810.1	20737
200_0.5_0.5_4	5558	1809.6	3514	**2724**	65.8	37
200_1_0.125_0.5	5824	1810.4	4651	3474	1811.3	14713
200_1_0.125_2	4741	1808.6	4476	2797	1814.6	19922
200_1_0.125_4	5336	1809.1	1359	**468**	79.4	87
200_1_0.5_0.5	4547	1808.9	1705	400	1843.0	2849
200_1_0.5_2	5508	1808.8	2185	1122	1817.4	27170
200_1_0.5_4	3507	1808.7	1519	376	1812.6	10454
200_1.5_0.125_0.5	5217	1808.8	1681	572	1831.2	8993
200_1.5_0.125_2	5883	1808.7	2037	702	1838.1	9044
200_1.5_0.125_4	4568	1809.0	375	**141**	44.7	16
200_1.5_0.5_0.5	4595	1811.5	376	**196**	99.0	1486
200_1.5_0.5_2	6707	1810.8	865	**16**	915.4	18445
200_1.5_0.5_4	4115	1808.7	115	**103**	52.8	325
200_3_500_4	4655	1808.9	301	**132**	419.5	9050
200_3_500_8	4455	1809.0	964	456	1810.9	30210
avg.	5181.8	1810.4	2122.7	1267.2	1011.2	9669.0

Table 7: Algorithmic performance on new instances; naming scheme: $n\_\lambda\_\delta\_\rho$. Optimal objective values are bold.

optimality and producing substantially tighter upper bounds than a default solver (CPLEX).

Given its good performance on batching problems, future research should focus on adapting B&BC to even more general problem settings. For instance, if a retrieval and a storage request refer to the same physical box, there may not only be a precedence relation between the requests, but also a minimum time lag that must pass between retrieving and then re-storing the item, to give logistics workers sufficient time to pick items from the box. More sophisticated distance metrics, like the Euclidean or Manhattan metrics, may also be considered to compute travel times of the S/R machine. Moreover, we only consider the dedicated storage case in this paper, where every item has a fixed known storage location. Our B&BC may be integrated into a holistic planning approach, which considers the problems of crane scheduling and storage assignment conjointly.

References

Abedinnia, H., Glock, C. H., Grosse, E. H., and Schneider, M. (2017). Machine scheduling problems in production: A tertiary study. *Computers & Industrial Engineering*, 111:403–416.

Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252.

Boysen, N. and Stephan, K. (2016). A survey on single crane scheduling in automated storage/retrieval systems. *European Journal of Operational Research*, 254(3):691–704.

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

Cabo, M., González-Velarde, J. L., Possani, E., and Solís, Y. Á. R. (2018). Bi-objective scheduling on a restricted batching machine. *Computers & Operations Research*, 100:201–210.

Cabo, M., Possani, E., Potts, C. N., and Song, X. (2015). Split–merge: Using exponential neighborhood search for scheduling a batching machine. *Computers & Operations Research*, 63:125–135.

Chang, D.-T., Wen, U.-P., and Lin, J. T. (1995). The impact of acceleration/deceleration on travel-time models for automated storage/retrieval systems. *IIE transactions*, 27(1):108–111.

Chen, M.-C. and Wu, H.-P. (2005). An association-based clustering approach to order batching considering customer demand patterns. *Omega*, 33(4):333–343.

Codato, G. and Fischetti, M. (2006). Combinatorial benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766.

De Koster, M., Van der Poort, E. S., and Wolters, M. (1999). Efficient orderbatching methods in warehouses. *International Journal of Production Research*, 37(7):1479–1504.

Emde, S. (2017). Optimally scheduling interfering and non-interfering cranes. *Naval Research Logistics (NRL)*, 64(6):476–489.

Gagliardi, J.-P., Renaud, J., and Ruiz, A. (2012). Models for automated storage and retrieval systems: a literature review. *International Journal of Production Research*, 50(24):7110–7125.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326.

Hooker, J. (2011). *Logic-based methods for optimization: combining optimization and constraint satisfaction*, volume 2. John Wiley & Sons, Hoboken, NJ.

Jia, W., Jiang, Z., and Li, Y. (2015). Combined scheduling algorithm for re-entrant batch-processing machines in semiconductor wafer manufacturing. *International Journal of Production Research*, 53(6):1866–1879.

Lawler, E. (1973). Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19:544–546.

Lee, C.-Y., Uzsoy, R., and Martin-Vega, L. A. (1992). Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40(4):764–775.

Lee, M.-K. and Kim, S.-Y. (1995). Scheduling of storage/retrieval orders under a just-in-time environment. *International Journal of Production Research*, 33(12):3331–3348.

Linn, R. and Xie, X. (1993). A simulation analysis of sequencing rules for asrs in a pull-based assembly facility. *International Journal of Production Research*, 31(10):2355–2367.

Méndez, C. A., Cerdá, J., Grossmann, I. E., Harjunkoski, I., and Fahl, M. (2006). State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Computers & Chemical Engineering*, 30(6-7):913–946.

Pinedo, M. (2016). *Scheduling: Theory, Algorithms, and Systems*. Springer International Publishing, Berlin, Germany.

Possani, E. (2001). *Lot streaming and batch scheduling: splitting and grouping jobs to improve production efficiency*. PhD thesis, University of Southampton.

Potts, C. N. and Kovalyov, M. Y. (2000). Scheduling with batching: A review. *European Journal of Operational Research*, 120(2):228–249.

Rahmaniani, R., Crainic, T. G., Gendreau, M., and Rei, W. (2017). The benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817.

Roodbergen, K. J. and Vis, I. F. (2009). A survey of literature on automated storage and retrieval systems. *European Journal of Operational Research*, 194(2):343–362.

Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160.

Van Den Berg, J. P. (1999). A literature survey on planning and control of warehousing systems. *IIE Transactions*, 31(8):751–762.

Zhou, H., Pang, J., Chen, P.-K., and Chou, F.-D. (2018). A modified particle swarm optimization algorithm for a batch-processing machine scheduling problem with arbitrary release times and non-identical job sizes. *Computers & Industrial Engineering*, 123:67–81.

Zhu, X. and Wilhelm, W. E. (2006). Scheduling and lot sizing with sequence-dependent setup: A literature review. *IIE Transactions*, 38(11):987–1007.