Utilizing Constraint Optimization for Industrial Machine Workload Balancing

Benjamin Kovács
Universität Klagenfurt, Austria

Pierre Tassel
Universität Klagenfurt, Austria

Wolfgang Kohlenbrein
Kostwein Holding GmbH, Klagenfurt, Austria

Philipp Schrott-Kostwein
Kostwein Holding GmbH, Klagenfurt, Austria

Martin Gebser
Universität Klagenfurt, Austria
Technische Universität Graz, Austria

Abstract

Efficient production scheduling is an important application area of constraint-based optimization techniques. Problem domains like flow- and job-shop scheduling have been extensive study targets, and solving approaches range from complete and local search to machine learning methods. In this paper, we devise and compare constraint-based optimization techniques for scheduling specialized manufacturing processes in the build-to-print business. The goal is to allocate production equipment such that customer orders are completed in time as good as possible, while respecting machine capacities and minimizing extra shifts required to resolve bottlenecks. To this end, we furnish several approaches for scheduling pending production tasks to one or more workdays for performing them. First, we propose a greedy custom algorithm that allows for quickly screening the effects of altering resource demands and availabilities. Moreover, we take advantage of such greedy solutions to parameterize and warm-start the optimization performed by integer linear programming (ILP) and constraint programming (CP) solvers on corresponding problem formulations. Our empirical evaluation is based on production data by Kostwein Holding GmbH, a worldwide supplier in the build-to-print business, and thus demonstrates the industrial applicability of our scheduling methods. We also present a user-friendly web interface for feeding the underlying solvers with customer order and equipment data, graphically displaying computed schedules, and facilitating the investigation of changed resource demands and availabilities, e.g., due to updating orders or including extra shifts.

2012 ACM Subject Classification Applied computing → Command and control

Keywords and phrases application, production planning, production scheduling, linear programming, constraint programming, greedy algorithm, benchmarking

Digital Object Identifier 10.4230/LIPIcs.CP.2021.36

Supplementary Material *Dataset*: `https://github.com/prosysscience/CPWorkloadBalancing` archived at `swh:1:dir:eab82a06d181ac3ff1b5d6c5e11f9d8d9700b0bd`

Funding This work was partially funded by KWF project 28472, cms electronics GmbH, FunderMax GmbH, Hirsch Armbänder GmbH, incubed IT GmbH, Infineon Technologies Austria AG, Isovolta AG, Kostwein Holding GmbH, and Privatstiftung Kärntner Sparkasse.

1 Introduction

High customization and flexibility of modern production processes increase the need for efficient and performant scheduling methods in order to optimize the utilization of required equipment and resources [13]. Well-studied problem domains like flow- and job-shop scheduling [7, 19, 24], or even tardiness minimization for jobs running on a single machine [11],

turn out to be NP-hard [23]. This elevated complexity makes constraint-based optimization techniques based on integer linear programming (ILP) [4], constraint programming (CP) [22], answer set programming (ASP) [17], or Boolean satisfiability (SAT) [5] attractive means for tackling scheduling tasks, while problem size and specifics of industrial domains necessitate customizations when adapting such solving methods from lab to real-world environments [14].

Industrial domains impose particular challenges due to irregularities and exceptions, such as maintenance downtimes or extra shifts, uncertainties and variances, e.g., due to machine breakdowns or manual handling times, as well as dynamics and policies, as evolving from production order updates or customer service duties. While ILP and CP solvers proved to be highly effective on benchmark scheduling problems [3, 9, 18, 21], which refer to abstract domain models that do not incorporate the specifics encountered in industrial practice, local dispatching [16], machine learning [25], and metaheuristic methods [27] are devised for reactive decision making but cannot guarantee (near-)optimal quality of online schedules.

In this paper, we devise and empirically study optimization techniques for scheduling specialized manufacturing processes of Kostwein Holding GmbH in the build-to-print business. Unlike with large-scale assembly line production, each product is made and customized to order, and lot size one is a frequent scenario. This goes along with the manual steps of preparing machines and tools for particular production tasks in order to process daily work plans at the discretion of experienced engineers. Hence, scheduling consists of deciding which production tasks shall be performed at each workday such that resource bottlenecks are projectively avoided and customer orders are completed in time as good as possible. Moreover, we have to take dynamic factors like varying processing times and production order updates into account, which can make schedules obsolete and necessitate quick changes.

In order to address these challenges, we furnish several approaches for scheduling pending production tasks to one or more workdays, considering that long processes may exceed daily machine capacities, for performing them. The four main contributions of our work are:

- We provide an abstract formulation of our production scheduling problem from industry and contribute an instance set based on real production data.
- We propose a greedy custom algorithm as well as ILP and CP models, enabling constraint optimization by state-of-the-art ILP and CP solvers like Gurobi[1] and Google OR-tools[2].
- We empirically investigate our scheduling methods in realistic setups and show that constraint-based optimization is highly beneficial for further improving greedy solutions.
- We present a user-friendly web interface for launching solvers and inspecting their results to facilitate rescheduling, e.g., with updated orders or extra shifts, and decision making.

As a result, we demonstrate the practical applicability of our scheduling methods to instances of industrial size and relevance. In particular, our devised techniques support a timely reaction to deviations, such as process delays and customer order updates, as well as an upfront identification of resource bottlenecks, thus assisting production managers to take appropriate measures like increasing machine capacities by including extra shifts or delegating some of the pending production tasks to external suppliers.

The rest of this paper is organized as follows. Section 2 introduces an abstract formulation of the production scheduling problem at Kostwein Holding GmbH. In Section 3, we present a greedy custom algorithm as well as ILP and CP models for solving the problem. We evaluate the devised techniques on instances extracted from real production data in Section 4. Section 5 describes the web interface used to deploy our scheduling methods in industrial practice. Finally, conclusions and future work are discussed in Section 6.

[1] https://www.gurobi.com/products/gurobi-optimizer/
[2] https://developers.google.com/optimization/

2 Problem formulation

The specialized manufacturing processes at Kostwein Holding GmbH go along with manual steps like preparing machines and tools for particular production tasks, where experienced engineers organize the workflows at their operating units. Given such independent and non-prescribed task execution, fine-grained sequencing of tasks competing for a resource, e.g., performed in flow- and job-shop scheduling, is out of scope and scheduling consists of deciding which of the pending production tasks shall be performed at each workday within the available machine capacities. The overall goal is thus to balance the workload of machines over several weeks from a global perspective such that bottlenecks are projectively avoided, customer orders can be completed in time, and as few additional resources as possible have to be utilized otherwise, e.g., by including extra shifts or delegating pending tasks to external suppliers. In the following, we formally specify the application scenario we are dealing with.

We consider a set M of *machines* and a set J of *jobs*, where each job $j \in J$ is a sequence $t^j_1, \ldots, t^j_{n_j}$ of *tasks* to be successively performed on *days* denoted by integers $D = \{0, \ldots, h\}$. Every job j has an associated *earliest start* day $e_j \in D$ such that $1 \leq e_j$, a *deadline* $d_j \in D$, and a *weight* $w_j \in \mathbb{N}$ used for penalizing tardiness. While day 0 can be viewed as the date of today and is admitted as deadline for jobs that should have been accomplished already, the positive earliest start days express that tasks require some lead time and can only be scheduled from tomorrow on or even later, e.g., in case raw materials need to be supplied first. The machines $m \in M$ are characterized by daily *capacities* $q_{m,k} \in \mathbb{Q}^+ \cup \{0\}$ for $1 \leq k \leq h$, where $q_{m,k} = 0$ means that m is *unavailable* on day k, such as on weekends or maintenance.

Each task $t = t^j_i$ of some job $j \in J$ is further characterized by the following attributes:

- the *machine* $m_t \in M$ to be used for processing t,
- the *processing time* $p_t \in \mathbb{Q}^+$ required for performing t,
- the number $g_t \in \mathbb{N}$ of *gap days* that must lie in-between the day of performing $t = t^j_i$ and t^j_{i-1} if $1 < i \leq n_j$, while we take $g_t = 0$ as fixed when $t = t^j_1$ has no preceding task, and
- a *coupling flag* $c_t \in \{0, 1\}$, where $c_t = 1$ indicates that $t = t^j_i$ has to be performed directly after t^j_{i-1} for $1 < i \leq n_j$, while $c_t = 0$ does not impose any such condition and is always the case when $t = t^j_1$.

Note that the machine to process a task is fixed, which reflects that specialized products are customized to order, so that the specification of jobs may involve CAD design as well as CNC programming and task allocation cannot easily be automated or adjusted. The possible gap days between a task and its predecessor are included to leave time for intermediate steps like specialized processes performed by external suppliers or transports between separate manufacturing sites. In general, we assume that operating units organize their daily work independently in order to perform their pending tasks efficiently, which precludes specific assumptions about the sequencing of tasks on a machine and time slots of processing within a day. Hence, a task $t = t^j_i$ must be performed $g_t + 1$ or more days later than its predecessor t^j_{i-1} (if any) and cannot be scheduled to the same day. On the other hand, the coupling flag $c_t = 1$ expresses that there must not be any delay between processing t^j_{i-1} and t^j_i, apart from days $k \in D$ such that $q_{m_t,k} = 0$ signals unavailability of the machine m_t. We use such coupling for modeling long production tasks that exceed the daily capacity of their machine and are thus broken up into parts to be processed directly after each other, which circumvents unintended and inoperative preemptive scheduling of (long) tasks.

A *schedule* is an assignment $a : T \to D$ of tasks $T = \{t^j_i \mid j \in J, 1 \leq i \leq n_j\}$ to days such that, for every job j, $1 < i \leq n_j$, and $t = t^j_i$, we have that

- $e_j \leq a(t_1^j)$,
- $a(t_{i-1}^j) + g_t < a(t_i^j)$, and
- if $c_t = 1$, $q_{m_t,k} = 0$ for each day $k \in D$ with $a(t_{i-1}^j) < k < a(t_i^j)$,

while the capacities $q_{m,k}$ of machines $m \in M$ for days $1 \leq k \leq h$ have to be respected, i.e., $\sum_{t \in T, m_t = m, a(t) = k} p_t \leq q_{m,k}$ must hold. That is, machine capacities, earliest start days of jobs, and requirements about the days between successive tasks impose hard constraints on feasible schedules. Clearly, the scheduling *horizon* h, i.e., the number of days to which tasks can be scheduled, must be large enough to allow for allocating all tasks within the available machine capacities, and in Section 3.1 we present a greedy algorithm to determine a sufficient horizon. Also note that the rational numbers for processing times and machine capacities are merely considered for a convenient representation of timeframes, e.g., in terms of fractions of hours or minutes, while they can always be scaled to integers without loss of precision, so that off-the-shelf ILP and CP solvers can be applied to the corresponding models described in Section 3.2 and Section 3.3.

The deadlines d_j and weights w_j for jobs $j \in J$ are used to assess the *quality* of schedules by a weighted sum $\sum_{j \in J, a(t_{n_j}^j) > d_j} w_j \cdot \left(\omega \cdot (a(t_{n_j}^j) - d_j) + \omega'\right)$ subject to *penalties* $\omega, \omega' \in \mathbb{N}$ per day a job is delayed or per delayed job, respectively. Both penalties are multiplied by job weights to incorporate priorities, the lower the weighted sum the better the quality of a schedule is, and the numbers taken for ω and ω' allow for balancing specific objectives obtained when either penalty is set to zero: if $\omega' = 0$, the total weighted tardiness of a schedule is to be minimized, or the weights of delayed jobs should be minimal in case $\omega = 0$.

Our scheduling problem is a specific version of the Resource-Constrained Project Scheduling Problem (RCPSP) [15], which considers the scheduling of activities under precedence and resource constraints such that particular objectives are optimized. The broad RCPSP framework embraces plenty problem variants with diverse features and solving approaches proposed in the literature. Our application can here be categorized as a single-mode [8], partially renewable [6], cumulative [20] resource allocation task with release dates and deadlines [10] subject to time-based objectives [2]. Machine unavailability days, usually standing for weekends or bank holidays, constitute a specific phenomenon of our scenario leading to variable time periods from start to completion for long production tasks, which we model by splitting tasks stretching over several days up into several coupled parts.

2.1 Example

Table 1 provides the input parameters of an example problem instance with four jobs and three machines. The earliest start days e_j, deadlines d_j, and weights w_j of the jobs $j \in \{1, 2, 3, 4\}$ are listed in Table 1a, showing that the first job cannot be started before day 2 and the others immediately on the first workday. Considering a horizon of five days, the daily machine capacities are given in Table 1b. Note that the positive capacities $q_{m,k}$ for $m \in \{1, 2, 3\}$ are uniform, i.e., $q_{1,k} = 8$, $q_{2,k} = 20$, and $q_{3,k} = 4$, while $q_{m,k} = 0$ signals unavailability of a machine m on day k otherwise. Currently we build on such uniform capacity patterns for production scheduling at Kostwein Holding GmbH, since we statically split long processes into coupled tasks such that the processing time of all but the last part amounts to the uniform capacity of the allocated machine. The tasks t_1^1 and t_2^1 of the first job, listed together with the other jobs' tasks in Table 1c, constitute a corresponding example, where both tasks are to be processed by machine 1, the processing time 8 of t_1^1 matches $q_{1,1} = q_{1,2} = q_{1,4} = q_{1,5} = 8$, and $c_t = 1$ for $t = t_2^1$ indicates that the second part obtained by splitting a long task of 10 time units is coupled. Clearly, a coupled task like $t = t_2^1$

Table 1 Input parameters of an example problem instance.

(a) Earliest start days, deadlines, and weights of jobs.

Job j	Earliest e_j	Deadline d_j	Weight w_j
1	2	4	3
2	1	4	2
3	1	2	2
4	1	1	2

(b) Daily capacities of machines.

Machine m	Day 1 $q_{m,1}$	2 $q_{m,2}$	3 $q_{m,3}$	4 $q_{m,4}$	5 $q_{m,5}$
1	8	8	0	8	8
2	20	0	0	20	20
3	4	4	4	4	4

(c) Machines, processing times, and requirements of production tasks.

Job j	Task t	Machine m_t	Processing time p_t	Gap days g_t	Coupled c_t
1	1	1	8	0	0
1	2	1	2	0	1
2	1	2	10	0	0
2	2	1	2	0	0
2	3	3	3	1	0
3	1	2	0.5	0	0
4	1	3	2.5	0	0

always comes with $g_t = 0$ gap days, as conflicting requirements to postpone and proceed with processing t would be imposed otherwise. Unlike that, $g_t = 1$ for $t = t_3^2$, i.e., the third task of the second job, means that at least one day must lie in-between performing t_2^2 and t_3^2, e.g., for transport to a different manufacturing site.

Partial schedules considering only the first or second job, respectively, are displayed in Figure 1a. Regarding the first job and its two coupled tasks in the upper part, t_1^1 is scheduled to the earliest start day $e_1 = 2$ and occupies the full capacity $q_{1,2} = 8$ of machine 1 on day 2. The coupled task t_2^1 must be scheduled to the next day on which machine 1 is available, which is day 4 in view of $q_{1,3} = 0$, and 6 time units of $q_{1,4} = 8$ are left for processing any other jobs' tasks by machine 1 on the same day. Turning to the second job and its three tasks in the lower part, t_1^2 is scheduled to day $e_2 = 1$ and utilizes half of the capacity $q_{2,1} = 20$ of machine 2. The successor task t_2^2 is processed by machine 1 directly on the next day 2, occupying 2 time units of the capacity $q_{1,2} = 8$. Given the gap days $g_t = 1$ for the remaining task $t = t_3^2$, the earliest day for performing t_3^2 is 4, and it takes 3 time units of the capacity $q_{3,4} = 4$ of machine 3. Scheduling t_3^2 to day 4 is required for finishing the second job within its deadline $d_2 = 4$, while a penalty weighted by $w_2 = 2$ would apply if delaying t_3^2 to day 5.

An entire schedule for the example instance in Table 1 is shown in Figure 1b. Here the three tasks of the second job are scheduled as discussed before, so that the second job is finished within its deadline. The short tasks t_1^3 and t_1^4 of the third and fourth job are additionally processed by machine 2 or 3, respectively, on the earliest start day $e_3 = e_4 = 1$. This is possible because the sum of processing times 10 and 0.5 of t_1^2 and t_1^3 stays within the capacity $q_{2,1} = 20$ of machine 2, and $q_{3,1} = 4$ also yields the availability of machine 3 to process t_1^4 for the required 2.5 time units. Different from the previous partial schedule for the first job, its first task t_1^1 cannot be scheduled to day 2 anymore because t_2^2 takes part of the capacity $q_{1,2} = 8$ of machine 1, while the full capacity would be required for processing t_1^1. Given that machine 1 is only available again on day 4, t_1^1 is performed then, and its coupled task t_2^1 on the next day 5. That is, the first job is finished one day later than its deadline $d_1 = 4$, which in view of the weight $w_1 = 3$ leads to the quality $3 \cdot (\omega \cdot (5-4) + \omega') = 3 \cdot (\omega + \omega')$ w.r.t. the penalties ω, ω' of the schedule displayed in Figure 1b.

Job 1	Day 1	Day 2	Day 3	Day 4	Day 5
Machine 1	/ 8	8 / 8	/ 0	2 / 8	/ 8
Machine 2	/ 20	/ 0	/ 0	/ 20	/ 20
Machine 3	/ 4	/ 4	/ 4	/ 4	/ 4

earliest start is day two

coupling

Job 2	Day 1	Day 2	Day 3	Day 4	Day 5
Machine 1	/ 8	2 / 8	/ 0	/ 8	/ 8
Machine 2	10 / 20	/ 0	/ 0	/ 20	/ 20
Machine 3	/ 4	/ 4	/ 4	3 / 4	/ 4

one rest day before task three

(a) Scheduling two jobs separately based on earliest availability of machines.

All jobs	Day 1	Day 2	Day 3	Day 4	Day 5
Machine 1	/ 8	2 / 8	/ 0	8 / 8	2 / 8
Machine 2	10 / 20	/ 0	/ 0	/ 20	/ 20
Machine 3	2.5 / 4	/ 4	/ 4	3 / 4	/ 4

(b) Schedule assigning the tasks of all four jobs to workdays.

Figure 1 Schedules for the example instance in Table 1.

3 Solving approaches

Our main goal is to utilize constraint optimization for solving the production scheduling problem specified in the previous section. This, however, requires the choice of a sufficient horizon in terms of workdays, so that all jobs can be scheduled and their tardiness inspected in order to assess the solution quality. To this end, we start by proposing a greedy algorithm able to quickly produce a sensible custom solution. Beyond the option to timely screen the effects of and react to deviations in resource availabilities and demands, we use greedy solutions to derive a feasible scheduling horizon along with strict limits on the completion of jobs. Exhaustive optimization can then be performed by applying state-of-the-art solvers to the ILP and CP models also presented in this section, where greedy solutions help to warm-start the optimization and converge to high-quality schedules in shorter solving time.

3.1 Greedy algorithm

We have devised a greedy algorithm to heuristically determine a sensible custom solution that yields a feasible scheduling horizon and can also be used to parameterize the constraint optimization performed by ILP and CP solvers. The basic idea is to proceed day by day to greedily schedule pending tasks, whose predecessors (if any) have been processed before, according to some priority until all tasks are scheduled. Letting $a(t_0^j) = e_j - 1$ and $f_i^j = \sum_{i<i'\leq n_j, t'=t_{i'}^j} (g_{t'} + 1)$ for each job $j \in J$ and $1 \leq i \leq n_j$, the *priority* function we use for pending tasks $t = t_i^j$ and days $k \in \mathbb{N}$ is calculated as follows:

$$priority(t_i^j, k) = \begin{cases} -\infty & \text{if } k \leq a(t_{i-1}^j) + g_t \\ \infty & \text{if } a(t_{i-1}^j) + g_t < k \text{ and } c_t = 1 \\ w_j \cdot \frac{\omega+\omega'}{\exp(d_j-(k+f_i^j))} & \text{if } a(t_{i-1}^j) + g_t < k,\ c_t = 0, \text{ and } k + f_i^j \leq d_j \\ w_j \cdot \frac{\omega}{\exp(n_j-i+1)} & \text{otherwise} \end{cases}$$

Algorithm 1 Greedy task scheduling.

```
T ← {t_1^j | j ∈ J};
k ← 0;
while T ≠ ∅ do
    k ← k + 1;
    S ← list of tasks t ∈ T in decreasing order of priority(t, k);
    foreach t ∈ S such that priority(t, k) ≠ −∞ do
        if p_t ≤ q_{m_t,k} then
            a(t) ← k;
            q_{m_t,k} ← q_{m_t,k} − p_t;
            T ← T \ {t};
            if i < n_j for t = t_i^j then T ← T ∪ {t_{i+1}^j};
        end
    end
end
```

The distinguished priority $-\infty$ is taken for (successor) tasks t that cannot be scheduled to the day k in view of the (positive) number g_t of required gap days. In turn, a coupled successor task t for which $c_t = 1$ must be scheduled to the next day on which its allocated machine is available, and in this case we use the distinguished priority ∞.

When no hard constraints forbid or force $t = t_i^j$ to be scheduled, we approximate the feasibility of finishing the job j within its deadline d_j according to the condition $k + f_i^j \leq d_j$, which applies if and only if the gap days and workdays needed for successor tasks of t are still left after the day k. If so, we optimistically assume the availability of allocated machines and consider the sum $\omega + \omega'$ of penalties as cost to safe by processing t on day k. To also reflect the criticality of jobs, we reduce this cost exponentially based on the number $d_j - (k + f_i^j)$ of remaining buffer days by which pending tasks of j can be further postponed within the deadline d_j. The outcome is then scaled by the weight w_j to incorporate the importance of j.

The case that remains is that the job j can certainly not be finished within its deadline d_j, so that the penalty $w_j \cdot \omega'$ applies no matter whether $t = t_i^j$ is scheduled to day k or later. This means that only the penalty ω is of further interest, we reduce it exponentially according to the number $n_j - i + 1$ of pending tasks of j, and again scale the outcome by the weight w_j. Among jobs that will certainly be delayed, the priority calculation thus prefers those with fewer remaining tasks, which are presumably easier to complete soon. Arguably, such a scheme may seem unbalanced and at risk to delay jobs needing more work for even longer, but our empirical investigation of greedy heuristics led to best schedules with the described prioritization strategy, and constraint optimization later goes for improvements.

Algorithm 1 outlines our greedy method by pseudo-code, whose central part is to traverse the pending tasks $t \in T$ per day k in decreasing order of priority. When the daily capacity $q_{m_t,k}$ of the allocated machine m_t suffices to process t, we schedule t and subtract its processing time p_t from the machine capacity before checking whether other tasks of lower priority can be performed in addition. If a scheduled task $t = t_i^j$ has the successor t_{i+1}^j, the latter is added to the set T of pending tasks and can be processed from day $k + 1$ on, where the greedy scheduling proceeds when no further task can be performed within the available machine capacities on day k. Given our assumption of uniform positive capacities for days on which machines are available, the splitting of longer tasks into coupled parts with processing times up the capacity of their allocated machine, and weekly repeating machine capacity patterns (except for occasional holidays, maintenance, or extra shifts), Algorithm 1 will eventually succeed to schedule all tasks and thus yield a sufficient scheduling horizon.

Reconsidering our example instance discussed in Section 2.1, all tasks that can be processed on day 1, i.e., t_1^2, t_1^3, and t_1^4, are scheduled greedily, as shown in Figure 1b. The pending tasks on day 2, which matches the earliest start $e_1 = 2$ of job 1, are $T = \{t_1^1, t_2^2\}$, both tasks compete for machine 1, and we have to inspect their priorities to decide whether to process t_1^1 or t_2^2. As displayed in Figure 1a, it is feasible to finish both tasks within their common deadline $d_1 = d_2 = 4$, so that we have to consider the gap days and workdays needed for successor tasks: $f_1^1 = 0+1 = 1$ and $f_2^2 = 1+1 = 2$ in view of t_2^1 or t_3^2, respectively. Together with the weights $w_1 = 3$ and $w_2 = 2$, this yields $\mathit{priority}(t_1^1, 2) = 3 \cdot (\omega+\omega')/e \leq 2 \cdot (\omega+\omega')/1 = \mathit{priority}(t_2^2, 2)$, where the priority of t_2^2 for day 2 is strictly greater whenever $\omega + \omega' \neq 0$. Hence, we greedily schedule t_2^2 to day 2, and then the remaining tasks t_1^1, t_2^1, and t_3^2 to the next days on which they can be processed w.r.t. the availability of machine 1 and the gap day required before t_3^2. This reproduces the schedule shown in Figure 1b by means of our greedy scheduling strategy.

3.2 ILP model

In order to fix a feasible scheduling horizon to be investigated by means of constraint-based optimization techniques, we take advantage of the greedy solution determined by our heuristic strategy. For each job $j \in J$, we restrict the processing of its tasks to the latest day $r_j = a(t_{n_j}^j) + b$, where $a(t_{n_j}^j)$ is the completion day of j in the greedy solution and $b \in \mathbb{N}$ is some constant. This yields the *range* from the earliest start day e_j until r_j as period of workdays to which tasks of j can possibly be scheduled, and globally leads to $h = \max\{r_j \mid j \in J\}$ as sufficient scheduling horizon. For our experiments in Section 4, we use $b = 10$ to admit up to 10 days later job completion than in the greedy solution, with the hope that any such local degradations allow for schedules of better overall quality. Notably, restricting the range of days for performing jobs based on the greedy solution avoids infeasibility due to enforcing too tight deadlines and may also benefit optimization performance by not overstretching the scheduling horizon in view of potentially far deadlines. Moreover, the derived range r_j implies $l_j = \max\{0, r_j - d_j\}$ as *limit* on the number of days by which a job j can be delayed beyond its deadline d_j in the worst case.

Our ILP model for task scheduling is shown in Figure 2, starting with a summary of instance parameters as introduced in Section 2, augmented by auxiliary functions u_m mapping a day k to the number of days up to k on which the machine $m \in M$ is available, the range r_j for job $j \in J$ derived from a greedy solution, and its limit l_j on delayed completion. The decision variables include Booleans $a_{t_i^j,k}$ to indicate that the task t_i^j is scheduled to a day k in the period from the earliest start e_j to the range r_j of its job j, a numerical variable z_j whose natural value up to l_j provides the delay days of j, and a Boolean z'_j signaling that the completion of j is delayed by at least one day. Hence, the objective function (7) matches the weighted sum $\sum_{j \in J, a(t_{n_j}^j) > d_j} w_j \cdot \big(\omega \cdot (a(t_{n_j}^j) - d_j) + \omega'\big)$ subject to penalties $\omega, \omega' \in \mathbb{N}$.

The first constraint (1) expresses that each task t_i^j must be scheduled to exactly one day between the earliest start e_j and range r_j of j. Daily machine capacities $q_{m,k}$ are checked by the constraint (2), making sure that the sum of processing times p_t over all tasks t processed by machine m on day k does not exceed $q_{m,k}$. The constraint (3) addresses the gap days $g_{t_i^j}$ that must lie in-between the predecessor t_{i-1}^j and a task t_i^j with $1 < i$. For example, we obtain $\sum_{1 \leq k \leq 5} k \cdot a_{t_2^2,k} + 1 < \sum_{1 \leq k \leq 5} k \cdot a_{t_3^2,k}$ for the tasks t_2^2 and t_3^2 of the instance discussed in Section 2.1, where $g_{t_3^2} = 1$ indicates that t_3^2 cannot be scheduled directly to the next day after performing t_2^2. Coupled tasks t_{i-1}^j and t_i^j are handled by the constraint (4), requiring that t_i^j is processed on the next day such that its allocated machine $m_{t_i^j}$ is available, where coefficients $u_{m_{t_i^j}}(k)$ map the availability days k of $m_{t_i^j}$ to consecutive natural numbers. Regarding

Parameters

$k \in D \setminus \{0\}$	day / set of days
$m \in M$	machine / set of machines
$q_{m,k} \in \mathbb{Q}^+ \cup \{0\}$	capacity of machine m for day k
$j \in J$	job / set of jobs
$e_j \in D \setminus \{0\}$	earliest start day of job j
$d_j \in D$	deadline of job j
$w_j \in \mathbb{N}$	weight of job j in objective function
$t_i^j \in T$	i-th task of job j / set of tasks
$m_t \in M$	allocated machine of task t
$p_t \in \mathbb{Q}^+$	processing time of task t
$g_t \in \mathbb{N}$	gap days of task t
$c_t \in \{0,1\}$	coupling flag of task t
ω, ω'	penalty for each delay day / each delayed job
$u_m : D \to D$	function defined by $k \mapsto \lvert\{1 \leq k' \leq k \mid q_{m,k'} \neq 0\}\rvert$
$r_j \in D \setminus \{0\}$	range specifying the latest day to complete job j
$l_j \in \mathbb{N}$	limit on the delay days of job j beyond its deadline d_j

Decision variables

$a_{t_i^j,k} \in \{0,1\}$	1 if task t_i^j is scheduled to day k with $e_j \leq k \leq r_j$, 0 otherwise
$z_j \in \{0,\ldots,l_j\}$	delay days of job j, limited by l_j
$z'_j \in \{0,1\}$	1 if job j is delayed, 0 otherwise

Constraints

$\sum_{k \in D\setminus\{0\}} a_{t_i^j,k} = 1 \quad \forall t_i^j \in T$	task assignment (1)
$\sum_{t \in T, m_t = m} p_t \cdot a_{t,k} \leq q_{m,k} \quad \forall m \in M, \forall k \in D \setminus \{0\}$	machine capacities (2)
$\sum_{k \in D\setminus\{0\}} k \cdot a_{t_{i-1}^j,k} + g_{t_i^j} < \sum_{k \in D\setminus\{0\}} k \cdot a_{t_i^j,k} \quad \forall t_i^j \in T, 1 < i$	gap days (3)
$\sum_{k \in D\setminus\{0\}} u_{m_{t_i^j}}(k) \cdot a_{t_{i-1}^j,k} + 1 = \sum_{k \in D\setminus\{0\}} u_{m_{t_i^j}}(k) \cdot a_{t_i^j,k} \quad \forall t_i^j \in T, c_{t_i^j} = 1$	coupled tasks (4)
$\sum_{k \in D\setminus\{0\}} k \cdot a_{t_{n_j}^j,k} - d_j \leq z_j \quad \forall j \in J$	delay days (5)
$z_j \leq l_j \cdot z'_j \quad \forall j \in J$	delayed jobs (6)

Objective function

$\min\ \omega \cdot \sum_{j \in J} w_j \cdot z_j + \omega' \cdot \sum_{j \in J} w_j \cdot z'_j$	weighted sum of delay days and delayed jobs (7)

Figure 2 ILP model for task scheduling.

the coupled tasks t_1^1 and t_2^1 of our example in Section 2.1, this scheme gives the constraint $1 \cdot a_{t_1^1,1} + 2 \cdot a_{t_1^1,2} + 2 \cdot a_{t_1^1,3} + 3 \cdot a_{t_1^1,4} + 4 \cdot a_{t_1^1,5} + 1 = 1 \cdot a_{t_2^1,1} + 2 \cdot a_{t_2^1,2} + 2 \cdot a_{t_2^1,3} + 3 \cdot a_{t_2^1,4} + 4 \cdot a_{t_2^1,5}$, in which the coefficients do not increase for day 3 in view of the unavailability of machine $m_{t_2^1} = m_{t_1^1} = 1$. The remaining constraints (5) and (6) impose lower bounds on the variables z_j and z'_j, reflecting the delay days or delayed completion, respectively, of a job j. For the first job of the instance in Section 2.1 with its deadline $d_1 = 4$, we obtain the specific

constraints $\sum_{1 \leq k \leq 5} k \cdot a_{t_2^1,k} - 4 \leq z_1$ and $z_1 \leq 1 \cdot z'_1$. This necessitates $z_1 = z'_1 = 1$ when $a_{t_2^1,5}$ signals that the last task t_2^1 of the job is processed on day 5, as done according to the schedule in Figure 1b, so that the cost $3 \cdot (\omega + \omega')$ is included in the objective function (7). Any further constraints imposing upper bounds and thus fixing the values of z_j and z'_j would be redundant, since minimization of the objective function also aims at assigning smallest feasible values and optimal solutions for the ILP model in Figure 2 readily give best schedules.

Recalling the role of a greedy solution, we use it for restricting the days to process jobs, thus reducing the representation size of our ILP model and targeting solvers towards better schedules in the neighborhood of the greedy solution, while even better schedules that entirely differ may be excluded. Given the high complexity of our industrial scheduling domain with thousands of tasks to be scheduled over several weeks, as empirically studied in Section 4, we make this trade-off and do not insist on schedules of theoretically best quality. However, we observed that admitting up to 10 days later job completion than in the greedy solution gives loose ranges for our instances, where constraint optimization yields schedules such that by far most jobs are completed earlier than in the greedy solution. The quality of schedules obtainable in reasonable solving time is also higher than with further relaxed ranges that increase likewise the representation size and the search space of instances. That is, we could not empirically confirm potential theoretical advantages due to extended limits on the completion of jobs, and thus up to 10 more days than in the greedy solution appear sufficiently cautious to us. Moreover, our experiments demonstrate that warm-starting solvers with a greedy solution, giving an initial hint on promising task assignments, significantly improves their optimization performance.

3.3 CP model

Given that linear constraints over finite-domain variables are supported by CP solvers (and rational coefficients can be scaled to integers without loss of precision), our CP model for task scheduling is primarily a syntactic reformulation of the constraints and objective function in Figure 2. However, rather than taking Booleans $a_{t_i^j,k}$ to represent that a task t_i^j is scheduled to day k, we use interval variables $a_{t_i^j}$ with associated duration 1 within the period from the earliest start e_j to the range r_j of the job j. This enables a convenient modeling of the constraint (2) for machine capacities in terms of the *cumulative* global constraint [12]. Since *cumulative* assumes the capacity $q_{m,k}$ of a machine $m \in M$ to be the same on each day k in the scheduling horizon, we use the uniform positive capacity on availability days of m as constant threshold, and model unavailability on a day k by adding a fixed task t taking the full capacity of m as processing time p_t. The solutions and objective function values then correspond one-to-one between our ILP and CP models, where either a Boolean $a_{t_i^j,k}$ or a variable assignment $a_{t_i^j} = k$ indicates the day k for performing a task t_i^j.

Our motivation for devising two models of similar functionality is that decision variables, linear constraints, and objective functions can be conveniently expressed in both formats, so the extra effort for modeling is modest, while the respective state-of-the-art solvers feature complementary constraint-based optimization techniques. A major advantage of ILP solvers is the estimation of solution quality based on the duality gap to relaxed real-valued solutions for a model [1]. Support of global constraints with dedicated propagation methods is a particular strength of CP solvers, where we make use of *cumulative* to limit machine capacities more compactly than by separate linear constraints for each day in the scheduling horizon.

4 Experimental analysis

We empirically evaluate the optimization performance and solution quality achieved with our scheduling methods on problem instances extracted from production data by Kostwein Holding GmbH, operating in the build-to-print business. The used instance sets and solver settings are described first, and then we present the results of our empirical evaluation.

4.1 Experimental setup

The production data supplied by Kostwein Holding GmbH contains a complete list of customer orders taken as snapshot from the company's ERP system. While these orders may have substantial lead times with planned delivery dates several months ahead, the workflows can possibly change during the production process due to customer requests, and new orders regularly come in between one data export and another. Hence, a production schedule incorporating all present orders will have a horizon of several months, yet be subject to revision in a few days at latest, so that filtering orders and their planned workflows to focus on a shorter scheduling horizon up to a few weeks is advisable in practice. A list of daily machine capacities constitutes the second kind of input, which usually follow a weekly pattern apart from bank holidays, planned maintenance, and occasionally extra shifts on weekends to manage peak loads. This matches our current assumption of uniform machine capacities on availability days in order to split long tasks into coupled parts occupying a machine for days.

The problem instances for our experiments are based on six customer order lists along with production workflows exported at different weeks. Following the idea that schedules should focus on the near to mid-term future, we extracted the jobs whose earliest start days lie within the next two, four, or six weeks, respectively, thus obtaining three instance sets with six realistic scenarios of roughly same size in each. In fact, the extracted jobs amount to about 6000 tasks per 2-weeks period, so that the average number of tasks to be scheduled is around 6000, 12000, or 18000 depending on the instance set.

In preliminary experiments, we compared the ILP solvers Gurobi[1] and IBM CPLEX Optimizer[3] as well as the CP solvers Google OR-tools[2] and IBM ILOG CP Optimizer[4]. Both the two ILP and the two CP solvers showed comparable performance on small problem instances, with a slight tendency in favor of Gurobi or Google OR-tools, respectively, when the instance size grows. We thus run Gurobi and Google OR-tools in our systematic experiments.

We conducted our experiments on a machine equipped with two Intel Xeon 6138 CPUs, providing 40 cores and offering 80 parallel threads. While Google OR-tools (version 8.2.8710) is configured to exploit all 80 threads, Gurobi (version 9.1.1) runs 32 threads, which is the default recommended by the developers, as more threads can in some cases deteriorate the optimization performance. The penalties for each delay day or delayed job, respectively, are fixed to $\omega = 1$ and $\omega' = 3$, so that entirely avoiding a delay counts more than reducing the delay length just by single days. Aiming at few delayed jobs makes practical sense because the delays point out bottlenecks that may a posteriori be resolved by including extra shifts or delegating critical production tasks to external suppliers. We report average objective function values along with the standard deviation relative to greedy solutions, determined by means of the heuristic algorithm in Section 3.1, for time limits of 120, 600, and 1800 seconds when the solvers are warm-started with greedy solutions. Without warm-start, we restrict

[3] https://www.ibm.com/analytics/cplex-optimizer/
[4] https://www.ibm.com/analytics/cplex-cp-optimizer/

Table 2 Objective function values relative to the greedy solution for warm-started solvers with 120s, 600s, and 1800s solving time limit, based on six instances per number of weeks, where respective gaps are additionally given for ILP. Results without warm-start, for 1800s solving time limit, include the number of instances for which a solution was found. No run terminated before the time limit.

Weeks		Greedy	CP (solved by Google OR-tools)				
	Time limit, start	N/A	120s, warm	600s, warm	1800s, warm	1800s, no warm	
	Number of tasks	Objective	Objective			Objective	Solved
2	6543 ± 1020	1.00	0.84 ± 0.04	0.65 ± 0.08	0.53 ± 0.11	0.59 ± 0.11	5
4	12204 ± 1128	1.00	0.96 ± 0.01	0.87 ± 0.03	0.78 ± 0.03	-	0
6	17387 ± 455	1.00	0.98 ± 0.00	0.94 ± 0.01	0.89 ± 0.02	-	0

Weeks	ILP (solved by Gurobi)							
	120s, warm		600s, warm		1800s, warm		1800s, no warm	
	Objective	Gap	Objective	Gap	Objective	Gap	Objective	Solved
2	0.55 ± 0.11	0.40 ± 0.18	0.41 ± 0.16	0.25 ± 0.10	0.35 ± 0.18	0.18 ± 0.08	2.00 ± 0.00	1
4	0.88 ± 0.08	0.55 ± 0.13	0.73 ± 0.03	0.43 ± 0.09	0.66 ± 0.07	0.25 ± 0.10	-	0
6	1.00 ± 0.02	0.60 ± 0.05	0.99 ± 0.03	0.55 ± 0.05	0.99 ± 0.03	0.52 ± 0.05	-	0

the comparison to 1800 seconds because the optimization performance declines dramatically and plenty runs do not even return a feasible solution within the solving time limit. For the ILP solver Gurobi, which reports the duality gap to relaxed real-valued solutions, we also indicate average gaps and the standard deviation relative to the quality of greedy solutions. Our instance sets and instructions for running the compared solvers are available in the supplementary material.

4.2 Experimental results

Table 2 summarizes the results of our empirical evaluation. The instance sets based on jobs with the earliest start day up to two, four, or six weeks in the future are listed in separate rows, and average objective function values together with further measurements (where applicable) for solvers and their setups are given in respective columns. The average number of tasks for the instance sets including jobs starting differently many weeks ahead is provided first, and the normalized quality 1.00 of greedy solutions is then indicated for reference.

For both the CP solver Google OR-tools and the ILP solver Gurobi, where results for the latter are displayed below the former, we observe that the solution quality improves substantially with increasing solving time limit when the solvers are warm-started with greedy solutions. However, Gurobi has a significant edge on Google OR-tools for the instance sets with jobs starting up to two or four weeks ahead, yielding 18% and 12% better quality of schedules relative to the greedy solution with 1800s time limit for both solvers. These percentages increase even more when shorter solving time limits are taken, such as 29% difference between the best solutions of Gurobi and Google OR-tools in 120s for the 2-weeks instances, and still 14% in 600s for the 4-weeks instances. Unlike that, Google OR-tools performs better than Gurobi on the 6-weeks instances, where its solutions are of 10% better quality with 1800s time limit. We checked that Gurobi here deals with roughly 500,000 linear constraints, and we conjecture that the handling of the *cumulative* global constraint by Google OR-tools is advantageous for instances of such large size. The box plot of average objective function values in Figure 3 also illustrates these clear trends visually.

Regarding the duality gaps provided by Gurobi, they range from 18% for 2-weeks instances to 52% for 6-weeks instances, so that the best schedules found within 1800s solving time limit cannot be claimed optimal, but constitute a trade-off between solving time and solution quality. This indicates that provably optimal schedules for our instances of industrial size

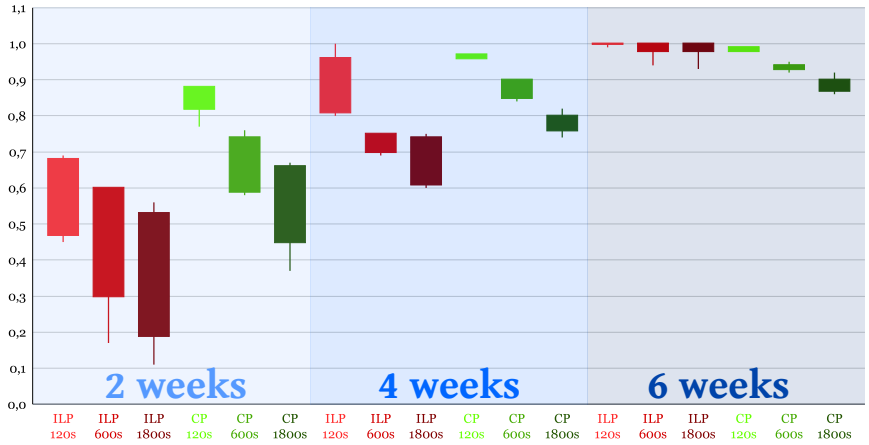


Figure 3 Plotted objective function values for ILP and CP solvers relative to the greedy solution.

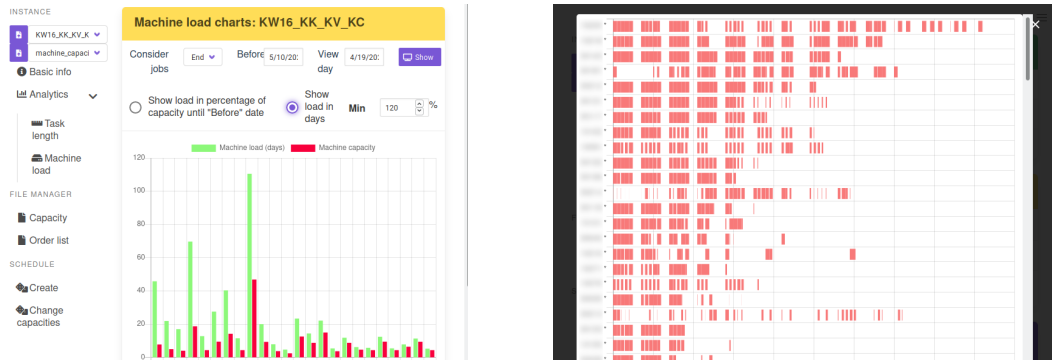
are beyond reach, and compromises have to be made. Limiting the considered jobs to those starting at most two to four weeks in the future already yields substantial improvements by the ILP solver Gurobi in comparison to greedy solutions deemed sensible. Moreover, the poor results of Gurobi and Google OR-tools without warm-start in Table 2, whose runs fail to return any feasible solution except for some 2-weeks instances, emphasize that a heuristic algorithm and constraint-based optimization techniques form a worthwhile combination to come to high-quality schedules in reasonable solving time.

5 Web interface

The presented scheduling methods are integrated as back-ends of a web application, whose user interface is illustrated in Figure 4, supporting the analysis of production planning scenarios at Kostwein Holding GmbH. To this end, production managers can upload customer order lists including the workflows of manufacturing processes exported from the company's ERP system. A second file provides the daily machine capacities by calendar dates, and the screenshot in Figure 4a indicates such input files in the upper left menu.

Before running solvers to compute schedules, the jobs can be filtered based on their earliest start days or deadlines to restrict the scope of the considered problem instance. As displayed in Figure 4a, our web interface allows for visually inspecting instance properties like the length distribution of tasks to be scheduled and the accumulated workloads of machines, which is helpful to spot critical resources and potential bottlenecks independently of specific production schedules. For example, peak loads of the allocated machines may be rebalanced by modifying the planned workflows and delegating tasks to alternative resources able to process them, or extra shifts on weekends may be included to temporarily increase the machine capacity and compensate the additional working hours by free days at another time. Taking appropriate measures to rebalance high workloads requires specific human experience about the involved manufacturing processes and can thus not be performed automatically in a meaningful way, yet our web application aims to support decision making by facilitating the exploration of possible scenarios like, e.g., the effects of increasing machine capacities.

Once a problem instance has been configured, the main functionality, however, consists of picking back-end solvers and settings to perform the task scheduling. In particular, the penalties ω and ω' for delay days or delayed jobs, respectively, can be adjusted, solving time limits be fixed, and for Gurobi also a duality gap below which the optimization is stopped can



(a) Machine loads.

(b) Visualization of greedy schedule.

Figure 4 User interface of the web application.

be given. As the experiments in Section 4 show, warm-starting Gurobi or Google OR-tools with a greedy solution virtually always benefits their optimization performance, so that the list of solver settings to launch one after the other, which can be compiled through the web interface, should usually include the warm-start option in each of its entries.

While solvers are run, our interface provides feedback about the optimization progress, including the objective function value of the best solution found so far and also the current duality gap for Gurobi. When a run is finished, the best schedule found can be visualized by a day chart, as exemplarily shown for a greedy solution in Figure 4b. Here the machines are sorted in decreasing order of their workloads, recurring idle periods of two days represent weekends, yet idleness of highly loaded machines on other days is presumably due to shortcomings of our greedy strategy and can be improved by constraint-based optimization. While the day charts give an overview of the distribution of machine workloads, the detailed schedules with the days for performing each task are available as editable spreadsheets.

We are currently about to deploy the web application at Kostwein Holding GmbH on a regular basis, with two main use cases in mind: strategic production scheduling for several weeks, resembling the scopes of the 2-weeks and 4-weeks instance sets in Section 4, to be run over night as well as reactive rescheduling during a day, where the number of jobs to consider will be much smaller to give quick feedback and possibly even optimal short-term schedules.

6 Conclusion

Our paper presents an industrial production scheduling problem and proposes three dedicated solving methods. We have devised a greedy algorithm to come up with a feasible custom solution quickly. Constraint optimization by state-of-the-art solvers can benefit the production scheduling process based on the provided ILP and CP models. Notably, we consult greedy solutions to derive feasible ranges of days for performing production tasks, while the deadlines given for jobs may be too tight for allocating all tasks within the available machine capacities. The deadlines are used to assess the quality of schedules in terms of delay days and delayed jobs, where production managers can then decide on measures to resolve resource bottlenecks.

Regarding the optimization performance, our experiments on problem instances of industrial size and relevance indicate that provably optimal schedules for thousands of production tasks are beyond reach. Nevertheless, the ILP solver Gurobi and the CP solver Google OR-tools successfully exploit the hints by greedy solutions taken to warm-start them and then manage to substantially improve the solution quality in reasonable solving time. While

some care must be taken about the representation size of instances, where around 12000 tasks in our 4-weeks instance set constitute a limit that should better not be exceeded, our empirical results demonstrate the clear advantages of combining a greedy algorithm with constraint-based optimization techniques. The synergy of the greedy and constraint-based methods thus proves to be a practically successful approach. Also taking into account that customer orders and production schedules are frequently subject to revision in our application scenario, longer scheduling horizons than considered in our experiments are of little practical interest, so that the developed scheduling methods scale well to realistic problem sizes.

For applying our scheduling methods in industrial practice, we have developed a user-friendly web interface through which production data and machine capacity profiles can be uploaded and filtered to conveniently specify problem instances. The interface also allows for configuring the penalties used within the objective function for assessing the quality of solutions as well as parameterizing the solvers to run for the optimization of schedules. Both instance properties and returned production schedules can be visually inspected for an accumulated overview of their key features. Since appropriate measures to resolve resource bottlenecks, such as increasing machine capacities, reallocating or delegating tasks, require expert knowledge that is beyond the scope of production scheduling, the web interface is meant to support production managers in exploring possible scenarios and making decisions.

We are currently in the trial phase of confronting our web application regularly with the real production data at Kostwein Holding GmbH, where the evaluation is performed by business experts who are not supposed to need in-depth understanding of the supplied solving methods. The goal is to gather user experience and practical feedback whether the provided functionality and performance are serviceable in the production scheduling process and help to complete customer orders without running into resource bottlenecks. In this respect, our scheduling methods and the encapsulating web application contribute prototypes for experimentation and the further refinement of requirements, where a few immediately compelling directions of future work are discussed in the remainder of this paper.

6.1 Future work

There are a number of opportunities to improve the performance and extend the applicability of the presented scheduling methods. The first consideration is that our greedy algorithm is still ad hoc and based on limited experiments with a handful of heuristics. Arguably, the instances of our scheduling problem are related to each other, as rescheduling with partially overlapping jobs is frequently needed in practice. Hence, there is a good chance that machine learning methods can be trained to typical resource demands and availabilities, and thus lead to better custom solutions than our greedy algorithm with manually selected heuristics. As one particularly promising approach, we are investigating natural evolution strategies [26] for training neural networks to provide the priority for greedy task scheduling. It is then an interesting question we did not explore yet whether warm-starting ILP and CP solvers with feasible solutions of better quality further improves their optimization performance.

Long production tasks that occupy a machine for several days are currently split into coupled parts with fixed processing times, based on the assumption of uniform machine capacities on availability days. This working hypothesis has been adopted to keep the initial modeling approaches simple, yet sacrifices flexibility regarding the machine capacity profiles that can be handled properly. Extending our constraint models to support a dynamic splitting mechanism, where the number and processing times of coupled tasks adjust to the available machine capacities, may allow for addressing richer application scenarios. For example, such

features would enable an automatic allocation of extra shifts declared as optional in the input, e.g., for switching between one- and two-shift operation modes based on demands, which at the moment requires the separate inspection of eligible machine capacity profiles.

A third direction of future work for tuning the optimization performance and achieving tighter (near-)optimality guarantees in terms of a small duality gap is to study worthwhile problem decompositions. For example, we may narrow down constraint-based optimization to tasks processed by highly loaded machines and use gap days as abstractions of skipped tasks. The abstracted tasks would then be inserted again in a post-processing phase.

References

1 Edward Anderson. A review of duality theory for linear programming over topological vector spaces. *Journal of Mathematical Analysis and Applications*, 97(2):380–392, 1983. doi:10.1016/0022-247X(83)90204-4.

2 Francisco Ballestín, Vicente Valls, and Sacramento Quintanilla. Due dates and RCPSP. In Joanna Józefowska and Jan Weglarz, editors, *Perspectives in Modern Project Scheduling*, volume 92 of *International Series in Operations Research & Management Science*, pages 79–104. Springer, 2006. doi:10.1007/978-0-387-33768-5_4.

3 Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*, volume 39 of *International Series in Operations Research & Management Science*. Springer, 2001. doi:10.1007/978-1-4615-1479-4.

4 Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and O. Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, 1971. doi:10.1007/BF01584074.

5 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009. doi:10.3233/978-1-58603-929-5-3.

6 Jan Böttcher, Andreas Drexl, Rainer Kolisch, and Frank Salewski. Project scheduling under partially renewable resource constraints. *Management Science*, 45(4):543–559, 1999. doi:10.1287/mnsc.45.4.543.

7 Jacques Carlier and Éric Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989. doi:10.1287/mnsc.35.2.164.

8 Ripon Chakrabortty, Ruhul Sarker, and Daryl Essam. Single mode resource constrained project scheduling with unreliable resources. *Operational Research*, 20(3):1–35, 2020. doi:10.1007/s12351-018-0380-7.

9 Giacomo Da Col and Erich Teppan. Industrial size job shop scheduling tackled by present day CP solvers. In Thomas Schiex and Simon de Givry, editors, *Proceedings of the Twenty-fifth International Conference on Principles and Practice of Constraint Programming (CP'19)*, volume 11802 of *Lecture Notes in Computer Science*, pages 144–160. Springer, 2019. doi:10.1007/978-3-030-30048-7_9.

10 Laure Drezet and Jean-Charles Billaut. A project scheduling problem with labour constraints and time-dependent activities requirements. *International Journal of Production Economics*, 112(1):217–225, 2008. doi:10.1016/j.ijpe.2006.08.021.

11 Jianzhong Du and Joseph Leung. Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, 15(3):483–495, 1990. doi:10.1287/moor.15.3.483.

12 Steven Gay, Renaud Hartert, and Pierre Schaus. Simple and scalable time-table filtering for the cumulative constraint. In Gilles Pesant, editor, *Proceedings of the Twenty-first International Conference on Principles and Practice of Constraint Programming (CP'15)*, volume 9255 of *Lecture Notes in Computer Science*, pages 149–157. Springer, 2015. doi:10.1007/978-3-319-23219-5_11.

13 Iiro Harjunkoski. Deploying scheduling solutions in an industrial environment. *Computers & Chemical Engineering*, 91:127–135, 2016. doi:10.1016/j.compchemeng.2016.03.029.

14 Iiro Harjunkoski, Christos Maravelias, Peter Bongers, Pedro Castro, Sebastian Engell, Ignacio Grossmann, John Hooker, Carlos Méndez, Guido Sand, and John Wassick. Scope for industrial applications of production scheduling models and solution methods. *Computers & Chemical Engineering*, 62:161–193, 2014. doi:10.1016/j.compchemeng.2013.12.001.

15 Sönke Hartmann and Dirk Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010. doi:10.1016/j.ejor.2009.11.005.

16 Oliver Holthaus. Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics*, 48(1):87–105, 1997. doi:10.1016/S0925-5273(96)00068-0.

17 Vladimir Lifschitz. *Answer Set Programming*. Springer, 2019. doi:10.1007/978-3-030-24658-7.

18 Leilei Meng, Chaoyong Zhang, Yaping Ren, Biao Zhang, and Chang Lv. Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem. *Computers & Industrial Engineering*, 142:Article ID 106347, 2020. doi:10.1016/j.cie.2020.106347.

19 Muhammad Nawaz, Emory Enscore, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983. doi:10.1016/0305-0483(83)90088-9.

20 Klaus Neumann and Christoph Schwindt. Project scheduling with inventory constraints. *Mathematical Methods of Operations Research*, 56:513–533, 2003. doi:10.1007/s001860200251.

21 Yves Pochet and Laurence Wolsey. *Production Planning by Mixed Integer Programming*. Springer, 2010. doi:10.1007/0-387-33477-7.

22 Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006. doi:10.5555/2843512.

23 Yuri Sotskov and Natalia Shakhlevich. NP-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics*, 59(3):237–266, 1995. doi:10.1016/0166-218X(95)80004-N.

24 Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993. doi:10.1016/0377-2217(93)90182-M.

25 Bernd Waschneck, André Reichstaller, Lenz Belzner, Thomas Altenmüller, Thomas Bauernhansl, Alexander Knapp, and Andreas Kyek. Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP*, 72:1264–1269, 2018. doi:10.1016/j.procir.2018.03.212.

26 Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(1):949–980, 2014. URL: http://dl.acm.org/citation.cfm?id=2638566.

27 Fatos Xhafa and Ajith Abraham, editors. *Metaheuristics for Scheduling in Industrial and Manufacturing Applications*, volume 128 of *Studies in Computational Intelligence*. Springer, 2008. doi:10.1007/978-3-540-78985-7.