Minimizing Cumulative Batch Processing Time for an Industrial Oven Scheduling Problem

Marie-Louise Lackner[1]
Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI, TU Wien, Austria

Christoph Mrkvicka
MCP GmbH, Wien, Austria

Nysret Musliu
Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI, TU Wien, Austria

Daniel Walkiewicz
MCP GmbH, Wien, Austria

Felix Winter
Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI, TU Wien, Austria

Abstract

We introduce the Oven Scheduling Problem (OSP), a new parallel batch scheduling problem that arises in the area of electronic component manufacturing. Jobs need to be scheduled to one of several ovens and may be processed simultaneously in one batch if they have compatible requirements. The scheduling of jobs must respect several constraints concerning eligibility and availability of ovens, release dates of jobs, setup times between batches as well as oven capacities. Running the ovens is highly energy-intensive and thus the main objective, besides finishing jobs on time, is to minimize the cumulative batch processing time across all ovens. This objective distinguishes the OSP from other batch processing problems which typically minimize objectives related to makespan, tardiness or lateness.

We propose to solve this NP-hard scheduling problem via constraint programming (CP) and integer linear programming (ILP) and present corresponding CP- and ILP-models. For an experimental evaluation, we introduce a multi-parameter random instance generator to provide a diverse set of problem instances. Using state-of-the-art solvers, we evaluate the quality and compare the performance of our CP- and ILP-models, which could find optimal solutions for many instances. Furthermore, using our models we are able to provide upper bounds for the whole benchmark set including large-scale instances.

2012 ACM Subject Classification Computing methodologies → Planning and scheduling

Keywords and phrases Oven Scheduling Problem, Parallel Batch Processing, Constraint Programming, Integer Linear Programming

Digital Object Identifier 10.4230/LIPIcs.CP.2021.37

Supplementary Material *Software (Source Code, Benchmark Set, and Results)*:
`https://cdlab-artis.dbai.tuwien.ac.at/papers/ovenscheduling/`

Funding The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

[1] Corresponding author

1 Introduction

In the electronics industry, many components need to undergo a hardening process which is performed in specialised heat treatment ovens. As running these ovens is a highly energy-intensive task, it is advantageous to group multiple jobs that produce compatible components into batches for simultaneous processing. However, creating an efficient oven schedule is a complex task as several cost objectives related to oven processing time, job tardiness, setup costs and setup times need to be minimized. Furthermore, a multitude of constraints that impose restrictions on the availability, capacity, and eligibility of ovens have to be considered. Due to the inherent complexity of the problem and the large number of jobs that usually have to be batched in real-life scheduling scenarios, efficient automated solution methods are thus needed to find optimized schedules.

Over the last three decades, a wealth of scientific papers investigated batch scheduling problems. Several early problem variants using single machine and parallel machine settings were categorized in [19] and shown to be NP-hard; a more recent literature review can be found in [15]. Batch scheduling problems share the common goal that jobs are processed simultaneously in batches in order to increase efficiency. Besides this common goal, a variety of different problems with unique constraints and solution objectives arise from different applications in the chemical, aeronautical, electronic and steel-producing industry where batch processing machines can appear in the form of autoclaves [13], ovens [12] or kilns [25].

For example, a just-in-time batch scheduling problem that aims to minimize tardiness and earliness objectives has been recently investigated in [18]. Another recent study [25] introduced a batch scheduling problem from the steel industry that includes setup times, release times, as well as due date constraints. Furthermore, a complex two-phase batch scheduling problem from the composites manufacturing industry has been solved with the use of CP and hybrid techniques [20].

Exact methods used for finding optimal schedules on batch processing machines involve dynamic programming [2] for the simplest variants as well as CP- and mixed integer programming (MIP) models. CP-models have e.g. been proposed in [13] and in [10], where both publications consider batch scheduling on a single machine with non-identical job sizes and due dates but without release dates. A novel arc-flow based CP-model for minimizing makespan on parallel batch processing machines was recently proposed in [21]. Branch-and-Bound [1] and Branch-and-Price [17] methods have been investigated as well. As the majority of batch scheduling problems are $\mathcal{NP}$-hard, exact methods are often not capable of solving large instances within a reasonable time-limit and thus (meta-)heuristic techniques are designed in addition. These range from GRASP approaches [6] and variable neighbourhood search [3], over genetic algorithms [14, 5], ant colony optimization [4] and particle swarm optimization [26] to simulated annealing [7].

In this paper, we introduce the Oven Scheduling Problem (OSP), which is a new real-life batch scheduling problem from the area of electronic component manufacturing. The OSP defines a unique combination of cumulative batch processing time, tardiness, setup cost, and setup time objectives that needs to be minimized. To the best of our knowledge, this objective has not been studied previously in batch scheduling problems. Furthermore, we take special requirements of the manufacturing industry into account. Thus, the problem considers specialized constraints concerning the availability of ovens as well as constraints regarding oven capacity, oven eligibility and job compatibility.

The main contributions of this paper are:

- We introduce and formally specify a new real-life batch scheduling problem.
- We propose solver independent CP- and ILP-models that can be utilized with state-of-the-art solver technology to provide an exact solution approach. In addition we provide on OPL-model for CP Optimizer using interval variables.
- To generate a large instance set, we introduce an innovative multi-parameter random instance generation procedure.
- We provide a construction heuristic that can be used to quickly obtain feasible solutions.
- All our solution methods are extensively evaluated through a series of benchmark experiments, including the evaluation of several search strategies and a warm-start approach. For a sample of 80 benchmark instances, we obtain optimal results for 37 instances, and provide upper bounds on the objective for all instances.

In the following, we first provide a description of the OSP (Section 2) before we introduce the CP model (Section 3). Then we present alternative models and search strategies (Section 4). Afterwards, we introduce a random instance generator and the construction heuristic (Section 5). Finally, we present and discuss experimental results (Section 6).

2 Description of the Oven Scheduling Problem (OSP)

The OSP consists in creating a feasible assignment of jobs to batches and in finding an optimal schedule of these batches on a set of ovens, which we refer to as machines in the remainder of the paper.

Jobs that are assigned to the same batch need to have the same *attribute*; in the context of heat treatment this can be thought of as the temperature at which components need to be processed. Moreover, a batch cannot start before the *release date* of any job assigned to this batch. The *batch processing time* may not be shorter than the minimal processing time of any assigned job and must not be longer then any job's maximal processing time, as this could damage the produced components. Every job can only be assigned to a set of *eligible machines* and machines are further only available during machine-dependent *availability intervals*. Moreover, machines have a maximal *capacity*, which may not be exceeded by the cumulative size of jobs in a single batch. When determining the start and processing times of batches, *setup times* between consecutive batches must also be taken into account. Setup times depend on the ordered pair of attributes of the jobs in the respective batches and are independent of the machine assignments. In the context of heat treatment, this can be thought of as the time required to switch from one temperature to another.

The main objective of the OSP is to minimize the cumulative batch processing time, total setup times and setup costs, as well as the number of tardy jobs. As the minimization of job tardiness usually has the highest priority in practice, the tardiness objective is weighted higher than the other objectives.

In practice the cumulative batch processing time should be minimized as the cost of running an oven depends merely on the processing time of the entire oven batch and not on the number of jobs within a batch. Therefore, running an oven containing a single small order incurs the same costs as running the oven filled to its maximal capacity.

Furthermore, we note that setup costs and setup times are not necessarily correlated. In fact, cooling down an oven from a high to a low temperature might not incur any (energy) costs, but still might require a certain amount of processing time. Setup costs can also capture costs related to personnel involved in the setup operation.

Using the three-field notation introduced by Graham et. al. [8], the OSP can be classified as $\tilde{P}|r_j, \bar{d_j}, maxt_j, b_i, ST_{sd,b}, SC_{sd,b}, \mathcal{E}_j, Av_m|\text{obj}$, where $\mathcal{E}_j$ stands for eligible machines and Av_m for availability of machines. A more formal description of the problem constraints and the objective function obj is given in Section 3.

As shown by Uzsoy [22], minimizing makespan on a single batch processing machine is an $\mathcal{NP}$-hard problem. It follows that the OSP is $\mathcal{NP}$-hard as well, as minimizing makespan on a single batch processing machine can easily be expressed within an instance of the OSP.

2.1 Instance parameters of the OSP

An instance of the OSP consists of a set $\mathcal{M} = \{1, \ldots, k\}$ of machines, a set $\mathcal{J} = \{1, \ldots, n\}$ of jobs and a set $\mathcal{A} = \{1, \ldots, a\}$ of attributes as well as the length $l \in \mathbb{N}$ of the scheduling horizon. Every machine $m \in \mathcal{M}$ has a maximum capacity c_m and machine availability is further specified in the form of time intervals $[as(m,i), ae(m,i)] \subseteq [0, l]$ where $as(m,i)$ denotes the start- and $ae(m,i)$ the endtime of the i-th interval on machine m. W.l.o.g. we assume that every machine has the same number of availability intervals and denote this number by I where some of these intervals might be empty (i.e. $as(m,i) = ae(m,i)$). Moreover, availability intervals have to be sorted in increasing order (i.e. $as(m,i) \leq ae(m,i) \leq as(m,i+1)]$ for all $i \leq I - 1$).

Every job $j \in \mathcal{J}$ is specified by the following list of properties:

- A set of eligible machines $\mathcal{E}_j \subseteq \mathcal{M}$.
- An earliest start time (or release time) $et_j \in \mathbb{N}$ with $0 \leq et_j < l$.
- A latest end time (or due date) $lt_j \in \mathbb{N}$ with $et_j < lt_j \leq l$.
- A minimal processing time $mint_j \in \mathbb{N}$ with $min_T \leq mint_j \leq max_T$, where $min_T > 0$ is the overall minimum and $max_T \leq l$ is the overall maximum processing time.
- A maximal processing time $maxt_j \in \mathbb{N}$ with $mint_j \leq maxt_j \leq max_T$.
- A size $s_j \in \mathbb{N}$.
- An attribute $a_j \in \{1, \ldots, a\}$.

Moreover, an $(a \times a)$-matrix of setup times $st = (st(a_i, a_j))_{1 \leq a_i, a_j \leq a}$ and an $(a \times a)$-matrix of setup costs $sc = (sc(a_i, a_j))_{1 \leq a_i, a_j \leq a}$ are given to denote the setup times (resp. costs) incurred between a batch using attribute a_i and a subsequent batch using attribute a_j. Setup times (resp. costs) are integers in the range $[0, max_{ST}]$ (resp. $[0, max_{SC}]$), where $max_{ST} \leq l$ (resp. $max_{SC} \in \mathbb{N}$) denotes the maximal setup time (resp. maximal setup cost). Note that these matrices are not necessarily symmetric.

2.2 Example instance with six jobs

Consider the following example for an OSP instance consisting of six jobs, two attributes, two machines and a scheduling horizon of length $l = 15$. The instance parameters are summarized in the following tables and matrices:

Machine	M_1	M_2
c_m	100	150
Availability intervals $[as, ae]$	[0,6] [8,14]	[2,10] [11, 14]

Job j	1	2	3	4	5	6
$\mathcal{E}_j$	M_1	M_1 M_2	M_1	M_1 M_2	M_2	M_2
et_j	2	0	0	3	0	2
lt_j	10	10	20	20	20	
$mint_j$	3	3	3	5	5	5
$maxt_j$	3	5	5	8	8	10
s_j	40	60	30	50	50	50
a_j	2	2	1	1	1	1

$$st = \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix} \quad sc = \begin{pmatrix} 0 & 20 \\ 10 & 0 \end{pmatrix}$$

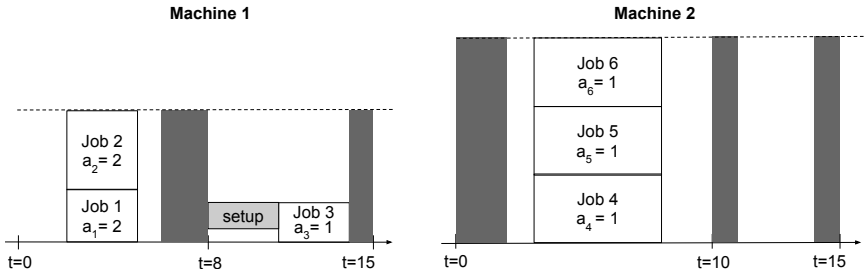

Figure 1 An optimal solution to the OSP for a small example problem.

An optimal[2] solution of this instance consists of three batches and is visualised in Figure 1. In this visualisation, the dark grey areas correspond to time intervals for which the machine is not available, the gray rectangle before the second batch on machine 1 is the setup time between the first and second batch and the dashed lines represent the machine capacities.

3 CP-model for the OSP

In this section we provide a formal definition of the OSP that will also serve as CP-model. We first explain how batches are modeled and afterwards define decision variables, objective function, and the set of constraints.

In the worst case we need as many batches as there are jobs; we thus define the set of potential batches as $\mathcal{B} = \{B_{1,1}, \ldots, B_{1,n}, \ldots, B_{k,1}, \ldots, B_{k,n}\}$ to model up to n batches for machines 1 to k. In order to break symmetries in the model, we further enforce that batches are sorted in ascending order of their start times and empty batches are scheduled at the end. That is, $B_{m,b+1}$ is the batch following immediately after batch $B_{m,b}$ on machine m for $b \leq n-1$. Clearly, at most n of the $k \cdot n$ potential batches will actually be used and the rest will remain empty.

3.1 Variables

We define the following decision variables:

- Machine assigned to job: $M_j \in \mathcal{M} \quad \forall j \in \mathcal{J}$
- Batch number assigned to job:[3] $B_j \in [n] \quad \forall j \in \mathcal{J}$
- Start times of batches: $S_{m,b} \in [0, l] \subset \mathbb{N} \quad \forall m \in \mathcal{M} \quad \forall b \in [n]$
- Processing times of batches: $P_{m,b} \in [0, max_T] \subset \mathbb{N} \quad \forall m \in \mathcal{M} \quad \forall b \in [n]$

Note that M_j and B_j determine to which batch job j is assigned ($B_{(M_j),(B_j)}$).
We additionally define the following auxiliary variables:

- Attribute of batch: $A_{m,b} \in [a] \quad \forall m \in \mathcal{M} \quad \forall b \in [n]$
- Availability interval for batch: $I_{m,b} \in [I] \quad \forall m \in \mathcal{M} \quad \forall b \in [n]$
- Number of batches per machine: $b_m \in [n] \quad \forall m \in \mathcal{M}$

[2] This solution is optimal with respect to weights $\alpha = 4$, $\beta = \gamma = 1$ and $\delta = 100$ as defined in Section 3.2.
[3] Throughout the paper, we write $[n]$ for the interval of integers $\{1, \ldots, n\}$.

3.2 Objective function

The objective function consists of four components: the cumulative batch processing time across all machines p, the number of tardy jobs t, the cumulative setup times st and the cumulative setup costs sc:

$$\begin{aligned} p &= \sum_{m\in\mathcal{M}, 1\le b\le n} P_{m,b} & t &= \left|\left\{j \in \mathcal{J} : S_{M_j,B_j} + P_{M_j,B_j} > lt_j\right\}\right| \\ st &= \sum_{\substack{m\in\mathcal{M}\\ 1\le b\le b_m-1}} st(A_{m,b}, A_{m,b+1}) & sc &= \sum_{\substack{m\in\mathcal{M}\\ 1\le b\le b_m-1}} sc(A_{m,b}, A_{m,b+1}) \end{aligned} \tag{1}$$

Normalization of cost components. In practice it is important to define an objective function that is highly flexible and configurable. We thus decided in consultation with our industrial partner to define the objective function as a linear combination of the four components. Therefore, the components p, st, sc and t need to be normalized to $\tilde{p}$, $\tilde{st}$, $\tilde{sc}$ and $\tilde{t} \in [0, 1]$:

$$\begin{aligned} \tilde{p} &= \frac{p}{avg_t \cdot n} \text{ with } avg_t = \left\lceil \frac{\sum_{j\in\mathcal{J}} mint_j}{n} \right\rceil & \tilde{t} &= \frac{t}{n} \\ \tilde{st} &= \frac{st}{\max(max_{ST}, 1) \cdot n} & \tilde{sc} &= \frac{sc}{\max(max_{SC}, 1) \cdot n} \end{aligned} \tag{2}$$

In the worst case, every batch processes a single job. In this case the total batch processing time is n times the average job processing time avg_t. The setup times and costs are bounded by the maximum setup time resp. cost multiplied with the number of jobs.[4] We take the maximum of 1 and max_{ST} resp. max_{SC} since it is possible that $max_{ST} = 0$ or $max_{SC} = 0$. The number of tardy jobs is clearly bounded by the total number of jobs.

Finally, the objective function obj is a linear combination of the four normalized components:

$$\text{obj} = (\alpha \cdot \tilde{p} + \beta \cdot \tilde{st} + \gamma \cdot \tilde{sc} + \delta \cdot \tilde{t})/(\alpha + \beta + \gamma + \delta) \quad \in [0, 1] \subset \mathbb{R} \tag{3}$$

where the weights α, β, γ and δ take integer values. Together with our industrial partner, we chose the default values to be $\alpha = 4$, $\beta = 1$, $\gamma = 1$, and $\delta = 100$, which captures the requirements of typical practical scheduling applications.

Integer-valued objective. As some state-of-the-art CP solvers can only handle integer domains, we propose an alternative objective function obj′, where we additionally multiply $\tilde{p}$, $\tilde{st}$, $\tilde{sc}$, and $\tilde{t}$ by the number of jobs and the least common multiple of avg_t, max_{ST} and max_{SC}:

$$\begin{aligned} \text{obj}' &= C \cdot n \cdot (\alpha + \beta + \gamma + \delta) \cdot \text{obj} \quad \in \mathbb{N} \\ &= \frac{\alpha \cdot C}{avg_t} \cdot p + \frac{\beta \cdot C}{\max(max_{ST}, 1)} \cdot st + \frac{\gamma \cdot C}{\max(max_{SC}, 1)} \cdot sc + \delta \cdot C \cdot t, \\ &\text{where } C = \texttt{lcm}(avg_t, \max(max_{ST}, 1), \max(max_{SC}, 1)). \end{aligned} \tag{4}$$

Preliminary experiments using the MIP solver Gurobi showed that using obj and obj′ both lead to similar results. We therefore used only obj′ in our final experimental evaluation.

[4] Actually, $st \le (n-1) \cdot max_{ST}$ and $sc \le (n-1) \cdot max_{SC}$ since no setup is necessary before the first batch. However, we want to keep the least common multiple of the denominators in equation (2) small in favor of the definition of obj′.

3.3 Constraints

In what follows, we formally define the constraints of the OSP using a high-level CP modeling notation. Most of these constraints can directly be handled by CP solvers, however we implicitly make use of constraint reification to express conditional sums and additionally use the maximum global constraint. Furthermore, we implicitly utilize the element constraint to use variables as array indices.

- Jobs may not start before their earliest start time: $S_{M_j,B_j} \geq et_j \quad \forall j \in \mathcal{J}$.
- Batch processing times must lie between the minimal and maximal processing time of all assigned jobs:

$$P_{m,b} = \max(mint_j : j \in \mathcal{J} \text{ with } B_j = b \wedge M_j = m) \quad \forall m \in \mathcal{M}, b \in [b_m]$$
$$P_{M_j,B_j} \leq maxt_j \quad \forall j \in \mathcal{J}$$

- Batches on the same machine may not overlap and setup times must be considered between consecutive batches:

$$S_{m,b} + P_{m,b} + st_{(A_{m,b},A_{m,b+1})} \leq S_{m,b+1} \quad \forall b \in [b_m - 1],$$

- Batches and the preceding setup times must lie entirely within one machine availability interval. In practice an interval for which a machine is unavailable can also represent a period for which the personnel required for the setup and running of ovens is unavailable. Therefore, setup times also need to fall completely within the associated availability interval. This is modeled with auxiliary variables $I_{m,b}$ which encode in which availability interval batch $B_{m,b}$ lies:

$$I_{m,b} = \max(i \in [I] : S_{m,b} \geq as(m,i)) \qquad \forall m \in \mathcal{M} \forall b \in [b]$$
$$S_{m,b} + P_{m,b} \leq ae(m, I_{m,b}) \qquad \forall m \in \mathcal{M} \forall b \in [b_m]$$
$$S_{m,b} - st_{(A_{m,b-1},A_{m,b})} \geq as(m, I_{m,b}) \qquad \forall m \in \mathcal{M}, \forall b \in \{2, \ldots, b_m\}.$$

- Total batch size must be less than machine capacity:

$$\sum_{j \in \mathcal{J}: M_j = m \wedge B_j = b} s_j \leq c_m \quad \forall m \in M \text{ with } b_m > 0, \forall b \in [b_m]$$

- Jobs in one batch must have the same attribute, which we model with auxiliary variables $A_{m,b}$ to set the attribute of a batch: $A_{M_j,B_j} = a_j \quad \forall j \in \mathcal{J}$.
- The assigned machine must be eligible for a job: $M_j \in \mathcal{E}_j \quad \forall j \in \mathcal{J}$.
- Set the number of batches per machine variables: $b_{M_j} \geq B_j \quad \forall j \in \mathcal{J}$
- Set variables for empty batches (i.e. batches $B_{m,b}$ with $b > b_m$):

$$S_{m,b} = l \qquad P_{m,b} = 0 \quad A_{m,b} = 1 \qquad I_{m,b} = I \quad \forall m \in \mathcal{M}, b_m < b \leq n$$

4 Alternative models and search strategies

4.1 ILP-model for the OSP

We propose a ILP-formulation, where batches are modeled similarly as in the CP-model ($B_{m,b}$ with $m \in \mathcal{M}, b \in [n]$), but we use a different set of decision variables: Binary variables $X_{m,b,j}$ encode whether job j is assigned to batch $B_{m,b}$ ($X_{m,b,j} = 1 \Leftrightarrow (B_j = b \wedge M_j = m)$), and integer variables $S_{m,b}$ and $P_{m,b}$ encode the start and processing times of batches.

To handle empty batches, we define an additional attribute with value 0 and extend the matrices of setup times $\bar{st}$ and setup costs $\bar{sc}$ so that no costs occur when transitioning from an arbitrary batch to an empty batch: $\bar{st}(a_i, a_j) = \bar{sc}(a_i, a_j) = 0$ if $a_i = 0$ or a_j=0. Moreover, we add a machine availability interval $[l, l]$ of length 0 to the list of availability intervals so that empty batches can be scheduled for this interval (the maximum number of intervals per machine therefore becomes $I + 1$). We then model the problem as follows:[5]

$$\text{Min.} \quad \text{obj'} = \tilde{\alpha} \cdot p + \tilde{\beta} \cdot st + \tilde{\gamma} \cdot sc + \tilde{\delta} \cdot t, \text{ where} \tag{5}$$

$$p = \sum_{\substack{m \in \mathcal{M} \\ 1 \le b \le n}} P_{m,b}, \qquad t = \sum_{\substack{j \in \mathcal{J}, m \in \mathcal{M} \\ 1 \le b \le n}} T_{m,b,j},$$

$$st = \sum_{\substack{m \in \mathcal{M} \\ 1 \le b \le n-1}} st_{m,b}, \qquad sc = \sum_{\substack{m \in \mathcal{M} \\ 1 \le b \le n-1}} sc_{m,b},$$

$$\text{s.t.} \sum\nolimits_{m \in \mathcal{M}, 1 \le b \le n} X_{m,b,j} = 1 \qquad \forall j \tag{6}$$

$$\sum\nolimits_{m \in \mathcal{E}_j, 1 \le b \le n} X_{m,b,j} = 1 \qquad \forall j \tag{7}$$

$$S_{m,b} \ge et_j \cdot X_{m,b,j} \qquad \forall m, \forall b, \forall j \tag{8}$$

$$mint_j \cdot X_{m,b,j} \le P_{m,b} \quad \wedge$$

$$P_{m,b} \le maxt_j \cdot X_{m,b,j} + max_T \cdot (1 - X_{m,b,j}) \qquad \forall m, \forall b, \forall j \tag{9}$$

$$S_{m,b+1} \ge S_{m,b} + P_{m,b} + st_{m,b} \qquad \forall m, \forall b \le n-1 \tag{10}$$

$$\sum\nolimits_{j \in \mathcal{J}} s_j \cdot X_{m,b,j} \le c_m \qquad \forall m, \forall b \tag{11}$$

$$a_j \cdot X_{m,b,j} \le A_{m,b} \quad \wedge$$

$$A_{m,b} \le a_j \cdot X_{m,b,j} + a \cdot (1 - X_{m,b,j}) \qquad \forall m, \forall b, \forall j \tag{12}$$

$$as(m,i) \cdot I_{m,b,i} \le S_{m,b} \quad \wedge$$

$$S_{m,b} \le ae(m,i) \cdot I_{m,b,i} + l \cdot (1 - I_{m,b,i}) \qquad \forall m, \forall b, \forall i \tag{13}$$

$$\sum\nolimits_{1 \le i \le I+1} I_{m,b,i} = 1 \qquad \forall m, \forall b \tag{14}$$

$$as(m,i) \cdot I_{m,b,i} \le S_{m,b} - st_{m,b-1} \qquad \forall m, \forall b \ge 2, \forall i \tag{15}$$

$$S_{m,b} + P_{m,b} \le ae(m,i) \cdot I_{m,b,i} + l \cdot (1 - I_{m,b,i}) \qquad \forall m, \forall b, \forall i \tag{16}$$

$$T_{m,b,j} \le X_{m,b,j} \qquad \forall j, \forall m, \forall b \tag{17}$$

$$S_{m,b} + P_{m,b} \le (X_{m,b,j} - T_{m,b,j}) \cdot (lt_j - l) + l \qquad \forall j, \forall m, \forall b \tag{18}$$

$$S_{m,b} + P_{m,b} + (1 - T_{m,b,j}) \cdot (l+1) > lt_j \qquad \forall j, \forall m, \forall b \tag{19}$$

$$\sum\nolimits_{j \in \mathcal{J}} X_{m,b,j} \ge 1 - E_{m,b} \qquad \forall m, \forall b \tag{20}$$

$$X_{m,b,j} \le 1 - E_{m,b} \qquad \forall m, \forall b, \forall j \tag{21}$$

$$S_{m,b} \ge l \cdot E_{m,b} \qquad \forall m, \forall b \tag{22}$$

$$P_{m,b} <= max_T \cdot (1 - E_{m,b}) \qquad \forall m, \forall b \tag{23}$$

$$E_{m,b} <= I_{m,b,I+1} \qquad \forall m, \forall b \tag{24}$$

$$A_{m,b} <= a \cdot (1 - E_{m,b}) \qquad \forall m, \forall b \tag{25}$$

$$E_{m,b} <= E_{m,b+1} \qquad \forall m, \forall b \le n-1 \tag{26}$$

[5] If not stated otherwise, $\forall m$ is short for $\forall m \in \mathcal{M}$, $\forall b$ for $\forall b \in [1, n]$, $\forall i$ for $\forall i \in [1, I+1]$ and $\forall j$ for $\forall j \in \mathcal{J}$.

$$st_{m,b} = st(A_{m,b}, A_{m,b+1}) \qquad \forall m, \forall b \leq n-1 \qquad (27)$$

$$sc_{m,b} = sc(A_{m,b}, A_{m,b+1}) \qquad \forall m, \forall b \leq n-1 \qquad (28)$$

$$X_{m,b,j} \in \{0,1\}, S_{m,b} \in [0,l], P_{m,b} \in [0, max_T],$$
$$A_{m,b} \in [0,a], I_{m,b,i} \in \{0,1\}, E_{m,b} \in \{0,1\},$$
$$st_{m,b} \in [0, max_{ST}], sc_{m,b} \in [0, max_{SC}] \qquad \forall m, \forall b, \forall j \qquad (29)$$

The weights of the objective function (5) are as described in equation (4) in Section 3.2. Constraint (6) ensures that every job is assigned to exactly one batch. Moreover, constraint (7) ensures that jobs can only be assigned to eligible machines. Constraint (8) specifies that a batch may not start before the earliest start of any job in the batch. The processing time of a batch is constrained by equations (9). Constraint (10) imposes additional restrictions on the starting times and ensures that the correct setup times are considered between consecutive batches. Constraint (11) ensures that the machine capacities are not exceeded for any batch.Constraint (12) ensures that jobs in the same batch have the same attribute. The binary auxiliary variables $I_{m,b,i}$ in constraint (13) encode whether batch $B_{m,b}$ is scheduled within the i-th availability interval $[as(m,i), ae(m,i)]$ of machine m. Therefore, if $I_{m,b,i} = 1$, it must hold that $as(m,i) \leq S_{m,b} \leq ae(m,i)$. The redundant constraint (14) ensures that every batch is scheduled within exactly one availability interval. Constraints (15) and (16) ensure that the entire processing time of batch $B_{m,b}$ as well as the preceding setup times $st_{m,b-1}$ (see (27)) lie within a single availability interval.

The binary auxiliary variables $T_{j,m,b}$ encode whether job j in batch $B_{m,b}$ finishes after its latest end date and is used to calculate the number of tardy jobs t. Constraint (17) ensures that $T_{j,m,b} = 1$ is only possible if job j is assigned to batch $B_{m,b}$. If $T_{j,m,b} = 0$, job j must finish before lt_j (Constraint (18)) and if $T_{j,m,b} = 1$, it must hold that $S_{m,b} + P_{m,b} > lt_j$ (Constraint (19)). The binary variables $E_{m,b}$ in equations (20) to (26) encode whether batch $B_{m,b}$ is empty or not. Constraints (20) and (21) ensure that $E_{m,b} = 1$ iff no job is scheduled for batch $B_{m,b}$. The constraints (22) to (24) set the start times, processing times, availability intervals, and attributes for empty batches. Moreover, in order to break symmetries, the list of batches $(B_{m,b})_{1 \leq b \leq n}$ per machine $m \in \mathcal{M}$ is sorted so that all non-empty batches appear first (constraint (26)). Constraint (27) defines the setup times $st_{m,b}$ and (28) the setup costs $sc_{m,b}$ between consecutive batches on the same machine. Finally, equation (29) defines the domains of all decision and helper variables.

We further investigated an alternative CP model using the Optimization Programming Language (OPL) [23]. More details for this alternative model can be found in Appendix A.

4.2 Programmed Search Strategies

We evaluated the performance of our models with the use of several programmed search strategies, which are based on variable- and value selection heuristics. For our experiments, we implemented the search strategies directly in the MiniZinc language using search annotations.

Variable Ordering: In our implemented search strategies we select at first an auxiliary variable that captures the total number of batches. For this variable we always use a minimum value first heuristic to encourage the solver to look for low cost solutions early in the search. Afterwards, we sequentially select decision variables related to a job by assigning the associated batch, machine, batch start time, and batch duration for the job (i.e., $B_1, M_1, S_{(M_1,B_1)}, P_{(M_1,B_1)}, \ldots, B_{|\mathcal{J}|}, M_{|\mathcal{J}|}, S_{(M_{|\mathcal{J}|},B_{|\mathcal{J}|})}, P_{(M_{|\mathcal{J}|},B_{|\mathcal{J}|})}$).

Variable Selection Heuristics: We use three different variable selection strategies on the set of decision variables that are related to job assignments: *input order* (select variables based on the specified order), *smallest* (select variables that have the smallest values in their domain first, break ties by the specified order), and *first fail* (select variables that have the smallest domains first, break ties by the specified order).

Value Selection Heuristics: We experimented with two different value selection heuristics for the set of variables which is related to job assignments: *min* (the smallest value from a variable domain is assigned first), and *split* (the variable domain is bisected to first exclude the upper half of the domain).

Evaluated Search Strategies: Using the previously defined heuristics we evaluated 8 different programmed search strategies:

1. *default*: Use the solver's default search strategy.
2. *search1*: Assign number of batches first, then continue with the solver's default strategy.
3. *search2*: Assign number of batches first, then continue with *input order* and *min* value selection on the job variables.
4. *search3*: Assign number of batches first, then continue with *smallest* and *min* value selection on the job variables.
5. *search4*: Assign number of batches first, then continue with *first fail* and *min* value selection on the job variables.
6. *search5*: Assign number of batches first, then continue with *input order* and *split* value selection on the job variables.
7. *search6*: Assign number of batches first, then continue with *smallest* and *split* value selection on the job variables.
8. *search7*: Assign number of batches first, then continue with *first fail* and *split* value selection on the job variables.

5 Random instance generator and construction heuristic

5.1 Construction of random instances

The random instance generator we propose is based on random instance generation procedures for related problems from the literature [14, 24]. However, as the existing variants were designed for batch scheduling problems which neither include machine eligibility constraints, machine availability times nor setup costs and times, the random generation of the associated instance parameters is a novel contribution of this paper. The list of parameters for this instance generator is given in Table 1.

Jobs. The list of n jobs is generated as follows. First, for every job j, the minimal processing time $mint_j$ is chosen using a discrete uniform distribution $U(1, max_T)$. For the maximum processing time, there are two options: either there is no upper limit on the processing time of jobs (`max_time` = `false`), in which case the maximum processing time is set to max_T for all jobs. Or, if `max_time` = `true`, the maximum processing time for a job is chosen using a discrete uniform distribution $U(mint_j, max_T)$. Next, the earliest start and latest end times are determined for every job. The earliest start time et_j is chosen similarly as in [24] according to a discrete uniform distribution $U(0, \lceil \rho \cdot Z \rceil)$ where $\rho \in [0, 1]$ and $Z = \sum mint_j$ is the total processing time of all jobs. If $\rho = 0$, all jobs are available right at the beginning and as ρ grows, the jobs are released over a longer interval. The latest end time lt_j is chosen as in [14] according to $lt_j = et_j + \lfloor U(1, \phi) \cdot mint_j \rfloor$ where $\phi \geq 1$. If $\phi = 1$, the latest end time is equal to the sum of the earliest start time and the minimum processing time, meaning that

Table 1 List of parameters of the random instance generator.

Name	Description	Values
Parameters relating to jobs		
n	number of jobs	10, 25, 50, 100
max_T	overall maximum processing time	10, 100
`max_time`	`true` if jobs have a max. processing time	`true`, `false`
ρ	determines spread of earliest start times	0.1, 0.5
ϕ	determines time from earliest start to latest end of job	2, 5
σ	determines number of eligible machines per job	0.2, 0.5
s	maximum job size	5, 20
Parameters relating to attributes		
a	number of attributes	2, 5
`s_time`	type of setup-time matrix	`constant`, `arbitrary`,
= `s_cost`	type of setup-cost matrix	`realistic`, `symmetric`
Parameters relating to machines		
k	number of machines	2, 5
$min_C = s$	lower bound for max. machine capacity (=max. job size)	5, 20
max_C	upper bound for maximum machine capacity	20, 100
τ	lower bound for the fraction of time machines are available	0.25, 0.75
max_I	max. number of availability intervals	5

all jobs must be processed immediately in order to finish on time. As ϕ grows, more time is given for every job to be completed and tardy jobs are less likely. Regarding the set of eligible machines for a job, one machine is chosen at random among all machines. Additional machines are then added to this set with probability σ each. The size s_j and attribute a_j of a job are both chosen at random between 1 and the maximum job size s or number of attributes respectively.

Attributes. The setup times and setup costs matrices can be of four different types: Constant, arbitrary, realistic and symmetric. For the type "constant", setup times/costs are all equal to a randomly chosen constant between 0 and $\lceil max_T/4 \rceil$. For the type "arbitrary", every entry is chosen independently at random between 1 and $\lceil max_T/4 \rceil$. For the type "realistic", setup times/costs between two batches of the same attribute are lower and are chosen independently at random between 0 and $\lceil max_T/8 \rceil$, whereas setup times/costs between different attributes are higher and are chosen between $\lceil max_T/8 \rceil + 1$ and $\lceil max_T/4 \rceil$. For the type "symmetric" a symmetric matrix is generated with random entries between 0 and $\lceil max_T/4 \rceil$.

Machines. The maximum machine capacity c_m is randomly chosen between min_C and max_C where the lower bound min_C is set to the maximum job size s to ensure that every job fits into every machine. For the machine availability times, we first fix the length of the scheduling horizon l. If we assume that every job is processed in a batch of its own and that all jobs are processed on the same machine, the total runtime is at most equal to the sum of all processing times Z plus n times the maximal setup time max_{st}. The parameter $\tau \in (0, 1]$ is a lower bound for the fraction of time that every machine is available. Thus, if max_{et} is the latest earliest start time, all jobs should – on average – be finished at time

$$l = max_{et} + \lceil (Z + n \cdot max_{st})/(\tau) \rceil$$

which we use to set the length of the scheduling horizon. Note that if the latest end date of a job is greater than this upper bound we simply use it instead. Now, for every one of the k machines, we pick the number of availability intervals I randomly between 1 and max_I. Every interval $[start_i, end_i]$ should be long enough to accommodate at least a single job with minimal processing time $min_T = \min(mint_j : j \in \mathcal{J})$ (plus the necessary setup times). Thus, the minimum distance between two interval start times $start_i$ and $start_{i+1}$ is $d = min_T + max_{st}$. We first pick the start time $start_1$ of the first interval: In order to guarantee that every machine is available at least a fraction τ of the time, $0 \leq start_1 \leq \lfloor l \cdot (1 - \tau) \rfloor$ must hold and in order to leave enough time for all availability intervals, it has to hold $start_1 \leq l - I \cdot d$. Next, for the start times of the remaining intervals, we pick $I - 1$ random integers between $(start_1 + d)$ and $(l - d)$ that are at least d apart. Finally, we determine the end time end_i of the i-th interval:

$$end_i = start_i + \max(d, \lceil U(\tau, 1) \cdot (start_{i+1} - start_i) \rceil)$$

5.2 Construction heuristic

We designed a simple construction heuristic that finds initial solutions for instances of our problem The heuristic starts at time 0. At every time step, the list of currently available machines and the list of remaining jobs that have already been released and can be processed on one of the machines is generated. Among these jobs, the one with the earliest due date is chosen and assigned to one of the machines if it fits into an availability interval. Once a job is scheduled, the algorithm adds other jobs that are currently available to the same batch if the job's attributes and maximal processing time as well as the machine's capacity allows so. If no job can be scheduled, the time is increased by one and the above procedure is repeated until the end of the scheduling horizon is reached or all jobs have been scheduled.

6 Experimental evaluation

Using our random instance generator, we created a large set of benchmark instances to evaluate the performance of our proposed models. First, we executed the random generator once for every possible configuration of the 15 parameter values specified in Table 1. Thereby we produced 1024 instances for each of the 16 combinations of the parameters n, k and a. Then we randomly selected 5 instances from every set of 1024 instances, creating a set of 80 instances which we used throughout our experiments. This set thus consists of 20 instances each with 10 (instances 1-20), 20 (21-40), 50 (41-60) and 100 jobs (61-80). All benchmark instances turn out to be satisfiable and solvable by the construction heuristic described in Section 5.2. This reflects our real-life industrial application, for which feasible solutions usually can be found heuristically and the main aim is to find cost-minimal schedules. Furthermore, note that in the particular real-life application scheduling scenarios consisting of roughly 50 jobs are considered to be the average use case.

We implemented both the CP- and ILP-model presented in Sections 3 and 4.1 using the high-level constraint modeling language MiniZinc [16] and used recent versions of Chuffed, OR-Tools, CP Optimizer and Gurobi. For Chuffed, OR-Tools and Gurobi, we used all 7 search strategies described in Section 4.2 and compared them with the solvers default search strategy (for CP Optimizer, search strategies are currently not supported by MiniZinc). For Chuffed, we activated the free search parameter which allows the solver to interleave between the given search strategy and its default search. Furthermore, we investigated a warm-start approach with Gurobi (for the other solvers, warm-start is currently not supported by MiniZinc): the

construction heuristic described in Section 5.2 was used to find an initial solution which was then provided to the model. Finally, the OPL-model presented in Section A was run using CP Optimizer in IBM ILOG CPLEX Studio. This results in a total of 53 different combinations of models, solvers and search strategies per instance; the time limit for every one of these combinations was set to one hour per instance. Experiments were run on single cores, using a computing cluster with 10 identical nodes, each having 24 cores, an Intel(R) Xeon(R) CPU E5–2650 v4 @ 2.20GHz and 252 GB RAM.

In the following we summarize our findings.[6] Table 2 provides an overview of the final results produced on the 80 benchmark instances with all evaluated methods. Based on initial experiments with different search strategies we selected the following search strategies solver pairings in the final experiments: chuffed-cp with *search3*, chuffed-ilp with *search2*, ortools-cp with *search6*, ortools-ilp with *search2*, and for all other solvers we used the *default* search strategy. The first column in each row denotes the evaluated solver and model. From left to right, columns 2–5 display: the number of solved instances, the number of instances where overall best cost results could be achieved, the number of obtained optimal solutions and the number of optimality proofs. Column 6 shows the number of fastest proofs. Columns 7–10 further present information for the 13 instances for which all solvers could deliver optimality proofs: The average number of nodes visited in the search process, the average runtime, the standard deviation of the number of visited nodes, and the standard deviation of the runtime.

Table 2 Overview of the final computational results based on 80 benchmark instances. Columns marked with * are based on the subset of instances for which all solvers could prove optimality.

solver	solved	best	opt	proof	fastest	avg nd*	avg rt*	std nd*	std rt*
chuffed-cp	66	25	24	14	1	3.87E+05	134.5	8.94E+05	240
chuffed-ilp	67	24	24	14	0	3.69E+05	259.8	7.75E+05	467.6
ortools-cp	72	20	20	20	0	n/a	47.2	n/a	48.5
ortools-ilp	78	20	20	20	0	n/a	12	n/a	11.9
cpopt-cp	64	37	32	14	0	3.25E+06	158.1	3.86E+06	201.1
cpopt-ilp	69	40	33	13	0	1.17E+07	487.9	1.20E+07	535.4
gurobi-cp	46	36	32	25	0	5.01E+02	30.7	4.43E+02	56.3
gurobi-ilp	65	52	37	32	23	4.74E+02	2.3	1.03E+03	3.1
gurobi-cp-ws	80	37	34	25	1	4.26E+02	26	4.04E+02	39.3
gurobi-ilp-ws	80	66	37	36	12	3.14E+02	2	5.07E+02	2.6
opl	51	22	22	19	0	1.05E+06	49.1	9.75E+05	57.3

Finding solutions. The warm-start approach for Gurobi finds solutions for all 80 instances. Even without warm-start, ortools-ilp was capable of finding solutions for 78 instances (72 for ortools-cp), followed by cpopt-ilp (69 instances) and chuffed-ilp (67 instances). Regarding the quality of found solutions, the best results were achieved by gurobi-ilp-ws (66 best results), followed by gurobi-ilp (52 best results). With cpopt-ilp and cpopt-cp as well as gurobi-cp-ws and gurobi-cp, best results could be achieved for roughly half of the instance set.

[6] The entire benchmark set as well as the detailed experimental results and the MiniZinc code are publicly available at `https://cdlab-artis.dbai.tuwien.ac.at/papers/ovenscheduling/`.

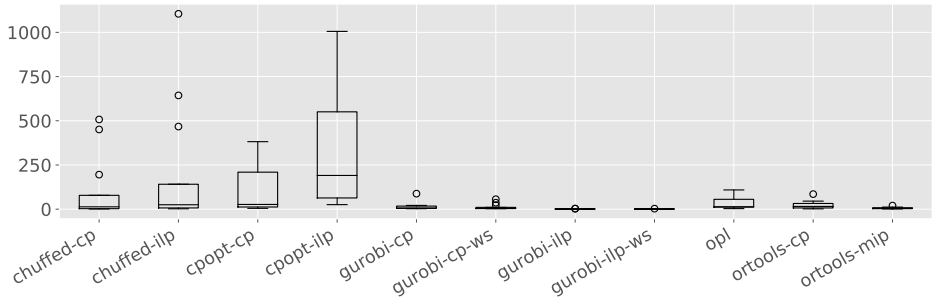

Figure 2 Comparison of proof times.

Finding optimal solutions. Using all evaluated methods, optimal solutions could be found for 37 instances: all instances with 10 jobs, 14 instances with 20 jobs, 2 with 50 jobs and one with 100 jobs. Most optimality proofs were provided by Gurobi (36 proofs with the ILP-model), followed by OR-Tools (20 proofs) and OPL (19 proofs). Even though cpopt-ilp (cpopt-cp) could only provide 13 (14) optimality proofs, it did find 33 (32) optimal solutions; chuffed-cp (chuffed-ilp) provided 14 optimality proofs and could find 24 optimal solutions. Figure 2 takes a closer look at the 13 instances for which all evaluated methods provided optimal solutions within the runtime limit (instances 1–7, 9, 10, 12, 15, 17 and 19) and compares the relative proof times for these instances. To calculate the relative proof time for an instance, we divide the absolute proof time by the overall fastest absolute proof time for that instance. Gurobi and OR-Tools deliver fastest proofs and outperform Chuffed and CP Optimizer; the OPL model lies in between. Moreover, it can be noted that Gurobi and OR-Tools perform best with the ILP-model, whereas Chuffed and CP Optimizer can provide faster proofs with the CP model.

Optimality gap. Figure 3 visualizes the overall smallest optimality gap per instance. That is, if $s(I)$ is the objective value of the overall best solution found (i.e., the minimal solution cost) and $b(I)$ is the best (i.e. maximal) dual bound found by Gurobi for instance I, the optimality gap is given by $g(I) = 1 - b(I)/s(I)$. The optimality gap generally increases with

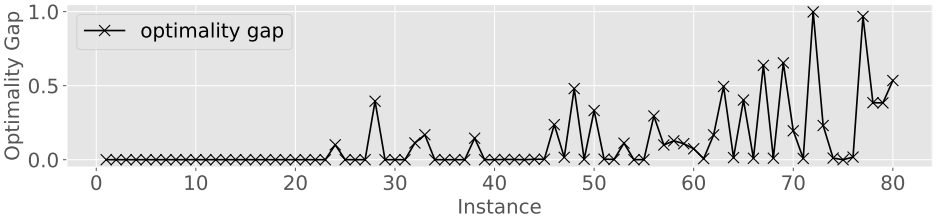

Figure 3 Overall smallest optimality gap per instance.

the number of jobs per instance: while the dual bounds are tight for all instances with 10 jobs and for most instances with 20 jobs, this is no longer the case for instances with 50 or 100 jobs. However, the size of the optimality gap is not purely determined by the number of jobs; there are 15 instances with 50 jobs or more for which the optimality gap is less than 1%.

Search strategies. The results of the comparison of search strategies for chuffed-cp, chuffed-ilp, ortools-cp and ortools-ilp can be found in Table 3. The first column in each row denotes the evaluated search strategy and the following columns contain the respective numbers of solved instances, best solution results and optimality proofs achieved by each search strategy in comparison to all other search strategies with the same solver. We can see that

Table 3 Comparison of 8 search strategies for CP- and ILP-models with Chuffed and OR-Tools.

search	chuffed-cp			chuffed-ilp			ortools-cp			ortools-ilp		
strategy	solved	best	proof	solved	best	proof	solved	best	proof	solved	best	proof
default	49	22	10	41	20	10	38	26	20	21	20	20
search1	64	27	14	64	22	14	37	27	20	25	22	20
search2	64	32	14	67	25	14	71	27	19	78	31	20
search3	66	29	14	64	25	13	71	33	20	78	31	20
search4	65	30	14	64	28	13	72	30	19	78	31	20
search5	65	30	14	67	24	14	70	28	19	78	31	20
search6	64	26	14	66	26	14	72	34	20	78	31	20
search7	65	32	14	65	24	14	71	31	19	78	31	20

using search strategies could greatly improve the number of solved instances for all four compared methods. The improvement was most significant for ortools-ilp, which could solve only 21 instances with the default strategy and 78 instances with search strategies 2–7 (all instances except 77 and 80). For Chuffed, all search strategies had a comparable impact on the number of solved instances. The number of best solutions found (in comparison to the other search/model configurations with the same solver) could be improved as well for all four methods; again, the most notable improvement was for ortools-ilp and search strategies 2–7 (20 vs. 31 best solutions). For Chuffed, the number of provided optimality proofs could also be improved using the suggested search strategies. Experiments were also run for gurobi-ilp and gurobi-cp with all search strategies, but no significant differences could be observed for this solver.

Warm-start with Gurobi. The construction heuristic could find feasible solutions for all 80 instances within few seconds. Warm-starting Gurobi with these initial solutions obtained bounds for all instances. Figure 4 allows a comparison of the solution quality per instance for the construction heuristic, gurobi-ilp and gurobi-ilp-ws. The heuristic solution could be improved by gurobi-ilp-ws for all 80 instances, thus providing 15 more results than gurobi-ilp. For 12 instances for which gurobi-ilp had previously found solutions, warm-starting could improve the solution quality. For 6 instances warm-starting led to a lower solution quality. Concerning optimality proofs, gurobi-ilp-ws was slightly worse than gurobi-ilp (32 vs. 36 proofs). To sum up, even though warm-starting found solutions that were better than those found by the heuristic, using this approach was not always advantageous.

7 Conclusion

In this paper, we introduced and formally defined the Oven Scheduling Problem and provide new instances for this problem. We propose CP- and ILP-models and investigate various search strategies. Using our models as well as a warm-start approach, we were able to find feasible solutions for all 80 benchmark instances. Provably optimal solutions could be found for nearly half of the instance set and for 50 of the 80 instances, the best optimality gap is

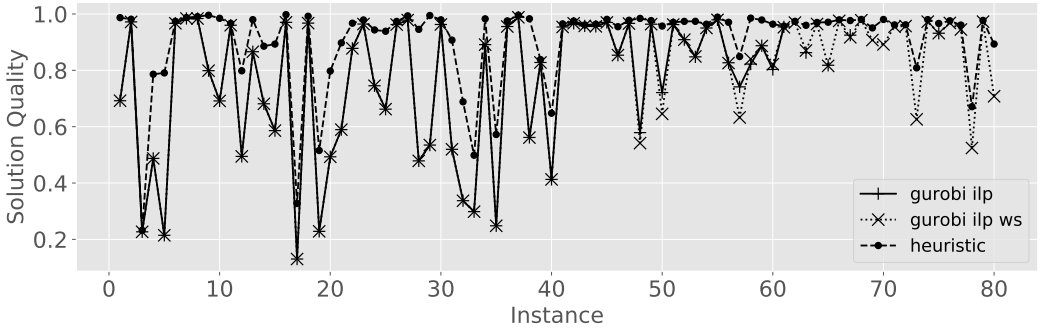

Figure 4 Solution quality per instance using the construction heuristic, gurobi-ilp without warm-start and gurobi-ilp-ws with the heuristically constructed solution for warm-start.

less than 1%. Overall, the best results could be achieved with Gurobi and OR-Tools for the ILP-model. Varying the search strategy had a major impact on the performance of the CP solvers Chuffed and Gurobi. To further improve the solution quality for large instances, we plan to develop meta-heuristic strategies based on local search or large neighborhood search. Moreover, in order to explain which parameters cause instances to be hard, an in-depth instance space analysis could be conducted.

References

1 Meral Azizoglu and Scott Webster. Scheduling a batch processing machine with incompatible job families. *Computers & Industrial Engineering*, 39(3-4):325–335, 2001.

2 Peter Brucker, Andrei Gladky, Han Hoogeveen, Mikhail Y Kovalyov, Chris N Potts, Thomas Tautenhahn, and Steef L Van De Velde. Scheduling a batching machine. *Journal of scheduling*, 1(1):31–54, 1998.

3 Eray Cakici, Scott J Mason, John W Fowler, and H Neil Geismar. Batch scheduling on parallel machines with dynamic job arrivals and incompatible job families. *International Journal of Production Research*, 51(8):2462–2477, 2013.

4 Bayi Cheng, Qi Wang, Shanlin Yang, and Xiaoxuan Hu. An improved ant colony optimization for scheduling identical parallel batching machines with arbitrary job sizes. *Applied Soft Computing*, 13(2):765–772, 2013.

5 Antonio Costa, Fulvio Antonio Cappadonna, and Sergio Fichera. A novel genetic algorithm for the hybrid flow shop scheduling with parallel batching and eligibility constraints. *The International Journal of Advanced Manufacturing Technology*, 75(5-8):833–847, 2014.

6 Purushothaman Damodaran, Mario C Vélez-Gallego, and Jairo Maya. A grasp approach for makespan minimization on parallel batch processing machines. *Journal of Intelligent Manufacturing*, 22(5):767–777, 2011.

7 Purushothaman Damodaran and Mario C. Vélez-Gallego. A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times. *Expert Systems with Applications*, 39(1):1451–1458, 2012.

8 Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

9 IBM. *IBM ILOG CPLEX Optimization studio, Getting Started with Scheduling in CPLEX Studio*, 2017.

10 Sebastian Kosch and J Christopher Beck. A new mip model for parallel-batch scheduling with non-identical job sizes. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 55–70. Springer, 2014.

11 Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250, 2018.

12 Chung-Yee Lee, Reha Uzsoy, and Louis A Martin-Vega. Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40(4):764–775, 1992.

13 Arnaud Malapert, Christelle Guéret, and Louis-Martin Rousseau. A constraint programming approach for a batch processing problem with non-identical job sizes. *European Journal of Operational Research*, 221(3):533–545, 2012.

14 Sujay Malve and Reha Uzsoy. A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families. *Computers & Operations Research*, 34(10):3016–3028, 2007.

15 Muthu Mathirajan and Appa Iyer Sivakumar. A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *The International Journal of Advanced Manufacturing Technology*, 29(9-10):990–1001, 2006.

16 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a Standard CP Modelling Language. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, Lecture Notes in Computer Science, pages 529–543, Berlin, Heidelberg, 2007. Springer.

17 N Rafiee Parsa, Behrooz Karimi, and A Husseinzadeh Kashan. A branch and price algorithm to minimize makespan on a single batch processing machine with non-identical job sizes. *Computers & Operations Research*, 37(10):1720–1730, 2010.

18 Sergey Polyakovskiy, Dhananjay Thiruvady, and Rym M'Hallah. Just-in-time batch scheduling subject to batch size. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, pages 228–235, New York, NY, USA, 2020. Association for Computing Machinery.

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

20 Tanya Y. Tang and J. Christopher Beck. CP and Hybrid Models for Two-Stage Batching and Scheduling. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Lecture Notes in Computer Science, pages 431–446, 2020.

21 Renan Spencer Trindade, Olinto CB de Araújo, and Marcia Fampa. Arc-flow approach for parallel batch processing machine scheduling with non-identical job sizes. In *International Symposium on Combinatorial Optimization*, pages 179–190. Springer, 2020.

22 Reha Uzsoy. Scheduling a single batch processing machine with non-identical job sizes. *The International Journal of Production Research*, 32(7):1615–1635, 1994.

23 Pascal Van Hentenryck. *The OPL optimization programming language*. MIT press, 1999.

24 Mario Cesar Velez Gallego. *Algorithms for scheduling parallel batch processing machines with non-identical job ready times*. PhD thesis, Florida International University, 2009.

25 Z. Zhao, S. Liu, M. Zhou, X. Guo, and L. Qi. Decomposition Method for New Single-Machine Scheduling Problems From Steel Production Systems. *IEEE Transactions on Automation Science and Engineering*, 17(3):1376–1387, 2020.

26 Hongming Zhou, Jihong Pang, Ping-Kuo Chen, and Fuh-Der Chou. 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, 2018.

A Alternative CP model using OPL

In addition to the solver independent models presented in sections 3 and 4.1, we developed an alternative CP model for IBM ILOG CPLEX Studio, since CP Optimizer is particularly well suited for scheduling problems [11]. This model is written using the Optimization Programming Language (OPL) [23] and makes use of *interval variables* for batches and

setup times between batches. It is based on the ILP model described in Section 4.1.[7] In the following, we briefly describe the decision variables and constraints that differ from the ILP model;[8] for an introduction to the used CP Optimizer concepts see [9]. We use optional interval variables for batches:

$$\text{interval } B_{m,b} \text{ optional} \subseteq [0,l] \text{ size} \in [min_T, max_T] \text{ intensity } av_m \quad \forall m \in \mathcal{M} \forall b \in [n].$$

Batches are optional since not all $k \times n$ batches will actually be used: Depending on whether any jobs are assigned to a batch or not, the batch interval variable will be present or absent. The intensity function av_m encodes the machine availability times and is modeled using *intensity step functions*; $av_m(t) = 100$ if machine m is available at time t and $av_m(t) = 0$ otherwise. Setup times between batches are also modelled using optional interval variables:

$$\text{interval } st_{m,b} \text{ optional} \subseteq [0,l] \text{ size} \in [0, max_{ST}] \text{ intensity } av_m \quad \forall m \in \mathcal{M} \forall b \in [n-1].$$

Besides these interval variables, we use the same decision variables as in Section 4.1: $X_{m,b,j} \in 0,1$ to encode whether job j is assigned to batch $B_{m,b}$, $A_{m,b} \in [0,a]$ for the attribute of batch $B_{m,b}$ and $sc_{m,b}$ for setup costs.

The following constraints are used for the batch and setup time interval variables:

- `presenceOf` ensures that a batch is present iff some job is assigned to it. Similarly, setup times are present iff the preceding and following batch are present.
- `endBeforeStart` is used to enforce the order of batches and setup times on the same machine and `endAtStart` is used to schedule setup times exactly before the following batch.
- `startOf` restricts the start time of batches to be after the earliest start date of any assigned job and `lengthOf` is used to enforce that batch processing times lie between the minimal and maximal processing time for every assigned job
- `noOverlap` is used as a redundant constraint to ensure that batches on the same machine do not overlap
- `forbidExtent` is used to guarantee that batches and setup times are scheduled entirely within one machine availability interval: whenever $B_{m,b}$ or $st_{m,b}$ is present, it cannot overlap a point t where $av_m(t) = 0$.

[7] We based the OPL model on our ILP model as it turned out that using CP Optimizer as solver via MiniZinc delivers particularly good results with the ILP model, see the results in Section 6.

[8] The full model will be made available on our website once this paper has been accepted for publication.