Modeling and Solving Parallel Machine Scheduling with Contamination Constraints in the Agricultural Industry

Felix Winter[1]
Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI, TU Wien, Austria

Sebastian Meiswinkel
MCP Algorithm Factory, 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 Algorithm Factory, MCP GmbH, Wien, Austria

Abstract

Modern-day factories of the agricultural industry need to produce and distribute large amounts of compound feed to handle the daily demands of livestock farming. As a highly-automated production process is utilized to fulfill the large-scale requirements in this domain, finding efficient machine schedules is a challenging task which requires the consideration of complex constraints and the execution of optional cleaning jobs to prevent a contamination of the final products. Furthermore, it is critical to minimize job tardiness in the schedule, since the truck routes which are used to distribute the products to customers are sensitive to delays. Thus, there is a strong need for efficient automated methods which are able to produce optimized schedules in this domain.

This paper formally introduces a novel real-life problem from this area and investigates constraint-modeling techniques as well as a metaheuristic approach to efficiently solve practical scenarios. In particular, we investigate two innovative constraint programming model variants as well as a mixed integer quadratic programming formulation to model the contamination constraints which require an efficient utilization of variables with a continuous domain. To tackle large-scale instances, we additionally provide a local search approach based on simulated annealing that utilizes problem-specific neighborhood operators.

We provide a set of new real-life problem instances that we use in an extensive experimental evaluation of all proposed approaches. Computational results show that our models can be successfully used together with state-of-the-art constraint solvers to provide several optimal results as well as high-quality bounds for many real-life instances. Additionally, the proposed metaheuristic approach could reach many optimal results and delivers the best upper bounds on many of the large practical instances in our experiments.

2012 ACM Subject Classification Computing methodologies → Planning and scheduling

Keywords and phrases Parallel Machine Scheduling, Contamination Constraints, Constraint Programming, Mixed Integer Quadratic Progamming, Metaheuristics, Local Search, Simulated Annealing

Digital Object Identifier 10.4230/LIPIcs.CP.2022.41

Supplementary Material *Software (Source Code, Benchmark Set, and Results)*: `https://doi.org/10.5281/zenodo.6797397`

Funding The financial support by the Austrian Federal Ministry for Digital and Economic Affairs,

[1] Corresponding author

the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

1 Introduction

In the modern agricultural industry large amounts of compound feed are produced and distributed to fulfill the demands of livestock farming. A highly-automated production environment is used to handle these large-scale requirements, where complex machinery handles the mixing and processing of the numerous ingredients of the compound feed products.

Finding efficient production schedules is a challenging task as several problem-specific constraints regarding contamination levels need to be fulfilled. Furthermore, job tardiness is a critical minimization objective in this domain as trucks are needed to distribute the compound feed products to many consumers and the associated routing is sensitive to delays in production. Currently, human planners create the production schedules either manually or basic greedy algorithms are used to produce solutions that often include a large number of tardy jobs and can hardly fulfill all constraints. Therefore, there is a strong need for novel efficient automated scheduling methods in this area.

In this paper we introduce a novel challenging real-life machine scheduling problem originating from the agricultural industry. As a set of predetermined jobs has to be scheduled on multiple machines where processing times depend on the predecessor job, the problem can be categorized as a parallel machine scheduling problem with setup times (PMSP). Finding efficient schedules for such problems is usually a challenging task and even early basic variants were shown to be NP-hard [3]. Therefore, a plethora of heuristic as well as exact solution approaches were proposed in the past and several surveys such as [4, 2] provide an overview of the related literature.

However, as practical machine scheduling problems appear in many different variations regarding the specified constraints and the objective function, complex large-scale applications are still being investigated in the recent literature. For example, [14] recently proposed a constraint programming (CP) approach to solve a resource-constrained PMSP which includes precedence constraints and aims to minimize job completion time. In [9], another variant with cyclical parallel machines originating from the agricultural industry was approached with mathematical programming as well as an adaptive variable neighborhood based metaheuristic. Another problem considering identical machine scheduling with tool requirements was recently investigated in [6]. In their paper, the authors proposed a matheuristic approach that combines a genetic algorithm together with mathematical programming to efficiently solve practical large-scale instances. In [13], large instances of a bi-objective PMSP with resource constraints during setups was tackled by introducing a novel iterated pareto greedy algorithm. The complexity of another real-life PMSP with setup times and resources originating from the manufacturing industry was analyzed in [5], and the authors proposed mixed integer programming (MIP) models to efficiently solve instances which are based on industrial data. Recently, exact and metaheuristic methods based on simulated annealing and MIP were proposed for another unique PMSP variant from the industry [12].

The problem we investigate in this paper can be compared to previously studied PMSPs as a set of given jobs with sequence-dependent processing times is scheduled to unrelated parallel machines. However, in addition to traditional PMSP constraints, a set of unique contamination level constraints needs to be fulfilled which further requires the consideration of including optional cleaning jobs in the schedule. Modeling these contamination constraints requires auxiliary variables with a continuous domain which makes it challenging to find efficient CP formulations and to the best of our knowledge such constraints have not been investigated for PMSPs in the past. The objective function for the investigated problem variant further includes a domain-specific variant of tardiness minimization, since due dates are defined on tours which are associated to groups of jobs.

In addition to formally introducing a novel real-life PMSP, we provide a set of 19 real-life benchmark instances that represent scenarios from the agricultural industry. As exact approaches to the problem we propose a direct- and an interval variable based CP model together with several programmed search strategies as well as a mixed integer quadratic programming (MIQP) formulation that include novel modeling techniques regarding the contamination constraints and cleaning jobs. Furthermore, we investigate a metaheuristic approach using simulated annealing to efficiently solve large-scale real-life problem instances which utilizes four problem specific neighborhood operators and uses randomly generated initial solutions.

An extensive experimental evaluation of all proposed approaches using the real-life benchmark instances shows that the CP approach is able to provide 9 optimal results and further produces high-quality solutions for all instances. The metaheuristic approach we propose further can reach 8 optimal results, similar upper bounds as obtained by the exact methods for the majority of instances, and two overall best upper bounds.

The remainder of the paper is structured as follows: We provide the problem description in Section 2, before we give the direct CP model in Section 3. Afterwards, a MIQP formulation and an alternative CP model using interval variables are proposed in Sections 4 & 5. In Section 6, we then introduce a metaheuristic approach based on local search. The experimental evaluation of all proposed approaches is discussed on Section 7. Finally, we give concluding remarks at the end of the paper.

2 Problem Description

The main aim of the PMSP variant we investigate is to create efficient schedules on multiple machines for a given set of jobs, where each job has to be scheduled on exactly one of its eligible machines. Furthermore, release dates (i.e. earliest start times) are specified for each job depending on the machine and the processing time of each job depends not only on the machine but also on the previously scheduled job. As several steps regarding the mixing of food products are performed within a job (each job produces a unique mix), there are complex domain-specific rules that determine sequence-dependent processing times to fulfill strict food requirements. Thus, we use sequence-dependent job times instead of setup times to specify this problem.

To avoid contamination of the produced goods maximum contamination levels of several contamination factors further must be respected whenever a job is started. Different ingredients, which are associated with the contamination factors, are used to produce unique food mixes in each job. Each ingredient causes a different contamination change regarding the individual factors, while the contamination needs to stay below a maximum to keep the food clean. The maximum level depends on the particular quantity and ingredient mix and thus is specified per factor/job pair. Further, the contamination reduction varies for each machine and factor, which is given as a reduction factor for each factor/machine pair.

The contamination levels are changed through the execution of the jobs, where each job can affect them differently. For example, a particular job could lower the level of one contamination factor and raise the level for another factor during its execution.

Additionally, optional cleaning jobs can be scheduled to reduce the contamination levels. Cleaning jobs behave similarly to regular jobs as they flush machines using a dummy mix and thereby update individual contamination levels. To fully reset all levels multiple cleaning jobs might be needed. Thus, in contrast to regular jobs they are optional and can be scheduled more than once if necessary. In the agricultural industry usually predetermined uniform lengths are used for the cleaning jobs, as it is challenging to give guarantees about contamination changes on variable lengths.

Finally, the main aim of the problem we investigate is to minimize tour tardiness, where each tour represents a truck route that is associated to a group of jobs that produce supplies for that tour. The food mixes each job produces are loaded on trucks which start delivery as soon as they are fully loaded, where food from a single job can be distributed on several tours and to multiple customers within a tour. The tour planning involves product forecasts and is not handled in our problem but predetermined in the input (a tour due date is the latest time the truck should start delivery).

Thus, the end time of the last job in the schedule that is associated to that tour is compared against the tour due date to calculate tour tardiness. The minimization objective then aggregates the tardiness over all tours. Table 1 summarizes the formal parameters of an instance to the investigated problem:

Table 1 Parameters for the parallel machine scheduling problem with contamination constraints.

Description	Parameter
Set of non-cleaning jobs	J^*
Set of cleaning jobs	S
Set of all jobs	$J = J^* \cup S$
Set of tours	T
Tours associated to each job	$jobTours_j \subseteq T \quad \forall j \in J^*$
Set of Machines	M
Processing time of job j after predecessor i on machine m	$p_{i,j,m} \in \mathbb{R}^+ \quad \forall i \in J, j \in J, m \in M$
Processing time of job i on machine m at the start of the schedule	$b_{i,m} \in \mathbb{R}^+ \quad \forall i \in J, m \in M$
Set of eligible machines per job	$E_j \subseteq M \quad \forall j \in J^*$
Tour due dates	$d_t \in \mathbb{N} \quad \forall t \in T$
Job release dates	$r_{j,m} \in \mathbb{N} \quad \forall j \in J, m \in M$
Set of contamination factors	K
Maximum contamination per factor and job	$C^{\max}_{k,j} \in \mathbb{R}^+ \quad \forall k \in K, j \in J$
Contamination volume per job	$C_{k,j} \in \mathbb{R}^+ \quad \forall j \in J, k \in K$
Initial contamination per machine	$C^0_{k,m} \in \mathbb{R}^+ \quad \forall k \in K, m \in M$
Contamination reduction multiplier per factor and machine	$RF_{k,m} \in [0,1] \quad \forall k \in K, m \in M$
Bound on the scheduling horizon	$h \in \mathbb{N}$
Bound on the contamination level	$u \in \mathbb{R}^+$
Bound on the tardiness of a tour	$v \in \mathbb{N}$

To further illustrate the investigated problem, Figure 1 visualizes a schedule for a simple toy example instance with jobs $J1 - J6$, a cleaning job C, and two machines M_1, M_2. In this example, J_1 is scheduled first on M_1 at time 0 and ends at time 2. Afterwards, J_3 is executed before C is scheduled at time 5. Finally, after a short break J_4 is started at time 8. J_4 cannot directly start after J_3 ends, as its release date is 8 in this example ($r_{J_4,M_1} = 8$). J_2 is scheduled first on M_2 and starts at time 1, as $r_{J_2,M_2} = 1$. After completion of J_2, J_5 starts just at time 4 since $r_{J_5,M_2} = 4$. Finally, J_6 is scheduled at time 6 and ends at time 9.

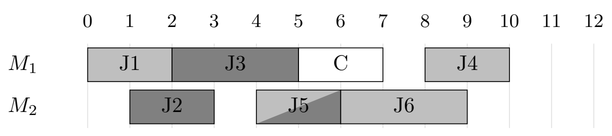

Figure 1 An example schedule for 6 jobs and a cleaning job on two machines.

Note that the jobs have processing times of 2 or 3 in the example, which are determined by the corresponding predecessor job. Therefore, the job lengths could be different if the job order was changed. The background colors of each job indicate the associated tours. In the example, jobs with a dark gray color belong to tour T_1, whereas light gray jobs belong to tour T_2. C has a white background and is not associated to any tour (which is always the case with cleaning jobs as they do not provide any demands), and J_5 is actually associated to both T_1 and T_2. Thus, production for T_1 is finished with the end of J_5 at time 6 and tour T_2 is completed at time 10. Let the tour due dates for this example be $d_{T_1} = 5$ and $d_{T_2} = 8$. Then T_1 would be late by one time unit and T_2 would be late by two units.

Figure 2 further illustrates the change in contamination levels on M_1 regarding two contamination factors $K = \{k_1, k_2\}$. Both factors start at level 0 at time point 0 but get raised through the execution of J_1 to 0.2 and 0.5. The execution of J_3 actually lowers the level of k_2 to 0.2, however, k_1 is raised to 1.5 by J_3 as it causes a strong contamination regarding that factor. C then reduces contamination for both factors down to $k_1 = 0.2, k_2 = 0$ so that J_4 can be scheduled, which again increases the contamination levels regarding both factors.

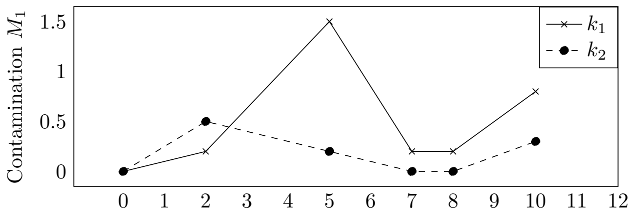

Figure 2 Contamination levels for factors k_1 and k_2 for the jobs scheduled on machine M_1 in the example schedule shown in Figure 1.

3 Constraint Programming Formulation

In this section, we propose a direct CP formulation for the PMSP with contamination constraints. The model serves as a formal specification of the problem, but can also be used as an exact solution approach together with a CP solver.

3.1 Decision Variables

The direct CP model uses decision variables that represent the job predecessors and thereby determines the complete job sequence on each machine:

- Predecessors of regular jobs: $x_j \in J \cup M \quad \forall j \in J^*$
- Cleaning job predecessors: $x_s \in J \cup M \cup \{\perp\} \quad \forall s \in S$

A predecessor either is another job or a machine, in the latter case the associated job starts the machine schedule. Cleaning jobs additionally can have their predecessor set to $\perp$ in which case they are not scheduled at all. Further, the following auxiliary variables are specified:

- Machine assignment for each job: $y_j \in M \quad \forall j \in J^*$
- Machine assignment for each cleaning job: $y_j \in M \cup \{\perp\} \quad \forall j \in S$
- Start time of each job: $start_j \in \mathbb{N} \quad \forall j \in J$
- End time of each job: $end_j \in \mathbb{N} \quad \forall j \in J$
- End time of each tour: $end_t \in \mathbb{N} \quad \forall t \in T$
- Contamination level before each job: $c_{k,j} \in \mathbb{R}^+ \quad \forall k \in K, j \in J$

These variables define the machine assignments for each job (if a cleaning job is not used its machine assignment is $\bot$) in addition to start- and end times for jobs and tours. Furthermore, the contamination states before each job are captured by a set of contamination level variables.

3.2 Constraints

In the following we formally specify the constraints of the investigated problem (Note that we implicitly make use of the element global constraint and constraint reification):

- Job start times need to be greater than or equal to the job release date:

$$start_j \geq r_{j,y_j} \quad \forall j \in J \tag{1}$$

- Tour end time variables should be greater than or equal to the associated job end times:

$$end_t \geq end_j \quad \forall j \in J, t \in jobTours_j \tag{2}$$

- Jobs can only be assigned to eligible machines:

$$y_j \in E_j \quad \forall j \in J^* \tag{3}$$

- Channel predecessor variables with start- and end time variables:

$$(x_j \in J) \Rightarrow (start_j > end_{x_j} \wedge end_j = start_j + p_{(x_j),j,(y_j)} \wedge y_j = y_{(x_j)}) \quad \forall j \in J \tag{4}$$

$$(x_j \in M) \Rightarrow (start_j \geq 0 \wedge end_j = start_j + b_{j,(y_j)} \wedge y_j = x_j) \quad \forall j \in J \tag{5}$$

$$(x_j = \bot) \Rightarrow (start_j = 0 \wedge end_j = 0 \wedge y_j = \bot) \quad \forall j \in S \tag{6}$$

- All predecessor assignments need to be different (unless they are set to $\bot$):

$$x_{(j_1)} \neq x_{(j_2)} \vee x_{(j_1)} = \bot \quad \forall j_1, j_2 \in J \text{ where } j_1 \neq j_2 \tag{7}$$

- Contamination levels must stay below the maximum values:

$$(c_{k,j} + C_{k,j}) \cdot (1 - RF_{k,(y_j)}) < C^{max}_{k,j} \quad \forall k \in K, j \in J \text{ where } x_j \in J \tag{8}$$

- Contamination levels at the beginning must be greater or equal to the initial state:

$$c_{k,j} \geq C^0_{k,(x_j)} \quad \forall k \in K, j \in J \text{ where } x_j \in M \tag{9}$$

- Contamination levels are updated based on the job sequence:

$$c_{k,j} \geq RF_{k,(y_j)} \cdot (c_{k,(x_j)} + C_{k,(x_j)}) \quad \forall k \in K, j \in J \text{ where } x_j \in J \tag{10}$$

3.3 Objective Function

As in the practical application it is better to have several tours that are a bit late than having a single tour that is late by a large amount of time, the tardiness of each tour is squared in the objective function:

$$minimize \quad \sum_{t \in T} \max\{0, end_t - d_t\}^2 \tag{11}$$

4 Mixed Integer Quadratic Programming Model

In this section, we specify a MIQP formulation of the PMSP with contamination and can be used as an exact solution approach with state-of-the-art MIQP solvers.

4.1 Decision Variables

A set of Boolean decision variables is used to determine the job sequence on all machines by capturing the job predecessors. Additionally, several auxiliary variables determine machine assignments, start- and end times, contamination levels, and tour tardiness:

- Boolean job predecessor variables that are set to 1 if and only if job i is a direct predecessor of job j on machine m (ω is a dummy job indicating the initial machine state):
$$x_{i,j,m} \in \{0,1\} \quad \forall i,j \in J'(J' = J \cup \{\omega\}), m \in M \tag{12}$$
- Boolean machine assignment variables:
$$y_{j,m} \in \{0,1\} \quad \forall j \in J, m \in M \tag{13}$$
- Start time of each job: $start_j \in \{0,\dots,h\} \quad \forall j \in J$
- End time of each job: $end_j \in \{0,\dots,h\} \quad \forall j \in J$
- End time of each tour: $end_t \in \{0,\dots,h\} \quad \forall t \in T$
- Contamination level before each job: $c_{k,j} \in [0,u] \quad \forall k \in K, j \in J$
- Tour tardiness: $tardiness_t \in \{0,\dots,v\} \quad \forall t \in T$

4.2 Constraints

The following list of linear constraints are used in the MIQP formulation:

- Each job can be used as at most one predecessor:
$$\sum_{j \in J \setminus \{i\}, m \in M} x_{i,j,m} \leq 1 \quad \forall i \in J \tag{14}$$
- Each machine can be used as at most one predecessor:
$$\sum_{j \in J} x_{0,j,m} \leq 1 \quad \forall m \in M \tag{15}$$
- Any job that is used as a predecessor also needs to have a single predecessor itself:
$$\sum_{m \in M, i \in J' \setminus \{j\}} x_{i,j,m} = \sum_{m \in M, i \in J' \setminus \{j\}} x_{j,i,m} \quad \forall j \in J \tag{16}$$
- If a job j has a predecessor i, i must have another predecessor on the same machine:
$$\sum_{k \in J' \setminus \{i\}} x_{k,i,m} \geq x_{i,j,m} \quad \forall i \in J, j \in J, m \in M \tag{17}$$
- If a job has a predecessor on a machine, it also needs to be assigned on the same machine:
$$\sum_{i \in J' \setminus \{j\}} x_{i,j,m} = y_{j,m} \quad \forall j \in J, m \in M \tag{18}$$
- If a job is a predecessor on a machine, it also needs to be assigned on the same machine:
$$\sum_{j \in J' \setminus \{i\}} x_{i,j,m} = y_{i,m} \quad \forall i \in J, m \in M \tag{19}$$
- Each job can only be assigned to eligible machines:
$$\sum_{m \in E_j} y_{j,m} = 1 \quad \forall j \in J^*, \quad \sum_{m \in E_j} y_{j,m} \leq 1 \quad \forall j \in S, \quad \sum_{m \in M \setminus E_j} y_{j,m} = 0 \quad \forall j \in J \tag{20}$$
- Each job needs a predecessor on an eligible machine (cleaning jobs may have one):
$$\sum_{i \in J, m \in E_j} x_{i,j,m} = 1 \quad \forall j \in J^*, \quad \sum_{i \in J, m \in E_j} x_{i,j,m} \leq 1 \quad \forall j \in S, \quad \sum_{i \in J, m \in M \setminus E_j} x_{i,j,m} = 0 \quad \forall j \in J \tag{21}$$

- Release dates have to be respected:

$$start_j \geq r_{j,m} - (1 - y_{j,m}) \cdot h \quad \forall j \in J, m \in M \tag{22}$$

- Channel tour end times to the job end times:

$$end_t \geq end_j \quad \forall j \in J, t \in jobTours_j \tag{23}$$

- Channel job start- and end times with the job predecessors (this also prevents cyclic predecessor assignments):

$$start_j > end_i - (1 - \sum_{m \in E_j} x_{i,j,m}) \cdot (h+1) \quad \forall i \in J', j \in J \tag{24}$$

$$end_j \geq start_j + \sum_{m \in E_j} (x_{i,j,m} \cdot p_{j,m,i}) - (1 - \sum_{m \in E_j} x_{i,j,m}) \cdot (h+1) \quad \forall i \in J', j \in J \tag{25}$$

- Contamination levels must stay below the maximum values:

$$(c_{k,j} + C_{k,j}) \cdot (1 - RF_{k,m}) < C_{k,j}^{max} + (1 - x_{i,j,m}) \cdot (1+u) \quad \forall k \in K, i \in J', j \in J, m \in E_j \tag{26}$$

- Set initial contamination levels for jobs at the start of the schedule:

$$c_{k,j} \geq C_{k,m}^{0} - (1 - x_{0,j,m}) \cdot (1+u) \quad \forall k \in K, j \in J, m \in E_j \tag{27}$$

- Set contamination levels in the sequence based on the job predecessor assignments:

$$c_{k,j} \geq RF_{k,m} \cdot (c_{k,i} + C_{k,i}) - (1 - x_{i,j,m}) \cdot (1+u) \quad \forall k \in K, i \in J, j \in J, m \in E_j \tag{28}$$

- Channel the tour tardiness variables to tour end time variables:

$$tardiness_t \geq end_t - d_t \tag{29}$$

4.3 Objective Function

The objective function aims to minimize the sum of each squared tour tardiness:

$$minimize \quad \sum_{t \in T} tardiness_t^2 \tag{30}$$

5 Alternative Constraint Programming Model using Interval Variables

In this section, we propose an alternative CP model that utilizes a widely used technique to capture the scheduling aspects of the investigated problem with optional interval variables and specialized global constraints [10, 11].

As alrady mentioned in Section 2 the practical application uses sequence-dependent processing times instead of setup times in its input parameters. However, in the interval based model we want to utilize efficient scheduling global constraints that only accept setup-time based input. Thus, we transform the sequence-dependent processing times into equivalent shorter processing and appropriate setup times in a preprocessing phase for this formulation (by taking the overall minimum processing time and calculating setup times based on the differences regarding each job predecessor).

We specify the following additional input parameters needed by this formulation:

- Setup time between two jobs on each machine: $s_{m,i,j} \in \mathbb{N} \quad \forall m \in M, i \in J', j \in J$
- Processing time (not sequence-dependent) of each job (ω has a processing time of 0): $p_j \in \mathbb{N} \quad \forall j \in J'$
- Minimum release time: $m_j = \min\{r_{j,m} | m \in M\} \quad \forall j \in J$
- Contamination volume for unused job positions is 0: $C_{k,0} = 0 \quad \forall k \in K$
- Acceptable difference for floating point comparison: $\epsilon \in \mathbb{R}^+$
- All jobs associated to a tour: $tourJobs_t = \{j \in J | t \in jobTours_j\}$

5.1 Decision Variables

We use optional interval variables which formally are decision variables whose domain values are a convex interval: $\{\perp\} \cup \{[s,e) | s, e \in \mathbb{Z}, s \leq e\}$, where s and e are the start- and end times of the interval and $\perp$ is a special value indicating that the interval is not scheduled at all. The start- and end times of such a variable var can be accessed via functions $startOf(var), endOf(var)$, and the function $presenceOf(var)$ returns true if and only if an optional interval is scheduled. Furthermore, several global constraints can be defined on interval variables. For example the $alternative(var, V)$ global constraint ensures that exactly one of the interval variables in the set V must have identical start- and end times to the interval variable var. We specify interval variables (optional interval variables) with the notation $intervalVar(p, [l, b])$ ($optIntervalVar(p, [l, b])$), where p denotes the processing time and $[l, b]$ specifies the time period in which the interval may be scheduled.

In addition to the interval variables, we make use of sequence variables which capture a permutation over a given set of interval variables and thereby express the sequence of all present interval variables. Thus, a sequence variable π can be used with global constraints such as $noOverlap(\pi)$, which ensures that all intervals in the sequence do not interfere temporally. Furthermore, sequence variables can serve as arguments for functions such as $first(\pi, var)$ and $typeOfPrev(\pi, var, -1)$, where $first(\pi, var)$ ensures that the interval variable var is scheduled first in sequence π and $typeOfPrev(\pi, var, -1)$ returns the job id which is the predecessor of interval var in sequence π or -1 if var is scheduled at the start of the sequence. The following decision variables are used in the interval variable based model:

- Interval variables for jobs and optional interval variables for cleaning jobs:
$$x_j : \text{intervalVar}(p_j, [m_j, h]) \quad \forall j \in J^*, \quad x_s : \text{optIntervalVar}(p_s, [m_s, h]) \quad \forall s \in S \tag{31}$$
- Optional interval variables that model machine assignments for each job:
$$xm_{m,j} : \text{optIntervalVar}(p_j, [r_{j,m}, h]) \quad \forall j \in J', m \in M \tag{32}$$
- A sequence variable for each machine:
$$\pi_m : \text{seq}(\{xm_{m,j} | j \in J'\}, J') \quad \forall m \in M \tag{33}$$

These interval variables are sufficient to determine the full schedule. However, to model the contamination levels we further need sets of auxiliary variables that capture the positions of jobs as well as auxiliary variables with a continuous domain that determine the contamination levels in the sequence. As we designed the interval based model for the use with CPoptimizer [11], which does not support floating point variables but only dynamic floating point expressions, we cannot rely on the recursive formulation of the contamination level constraints that was used for the models proposed in sections 3 & 4. Instead, we introduce additional auxiliary variables to represent all possible job positions which can be used together with floating point dynamic expressions for the contamination constraints:

- Variables storing the job positions (0 is used if the job is not scheduled on that machine):
$$jobPos_{m,j} \in \{0, \ldots, |J|\} \quad \forall m \in M, j \in J \tag{34}$$
- Variables storing the job scheduled at each position (0 is used if no job is scheduled):
$$jobAt_{m,j} \in \{0\} \cup J \quad \forall m \in M, j \in \{1, \ldots, |J|\} \tag{35}$$
- Variables storing the job predecessors (0 if the job is not scheduled on the machine, -1 if it has no predecessor):
$$prev_{m,j} \in \{-1, 0\} \cup J' \quad \forall m \in M, j \in J \tag{36}$$
- Dynamic expressions representing the contamination levels before each job position:
$$c_{k,m,0} = C^0_{k,m}, \quad c_{k,m,i} = RF_{k,m} \cdot (c_{k,m,(i-1)} + C_{k,(jobAt_{m,i})}) \quad \forall k \in K, m \in M, i \in \{1, \ldots, |J|\} \tag{37}$$
- Dynamic expressions representing the contamination volume at each position:
$$cv_{k,m,i} = (c_{k,m,i-1} + C_{k,(jobAt_{m,i})}) \cdot (1 - RF_{k,m}) \quad \forall k \in K, m \in M, i \in \{1, \ldots, |J|\} \tag{38}$$

5.2 Constraints

The following set of constraints are specified for the interval variable based model:

- Release dates must be respected for each present job interval:

$$\text{presenceOf}(xm_{m,j}) \Rightarrow (\text{startOf}(xm_{m,j}) - s_{m,(prev_{m,j}),j} \geq r_{j,m}) \quad \forall j \in J, m \in M \tag{39}$$

- Each job can only be scheduled on one machine:

$$\text{alternative}(x_j, \{xm_{m,j} | m \in M\}) \quad \forall j \in J \tag{40}$$

- The dummy job ω is scheduled as the first job on all machines:

$$\text{first}(\pi_m, xm_{m,\omega}) \wedge \text{presenceOf}(xm_{m,\omega}) \quad \forall m \in M \tag{41}$$

- All jobs in a machine sequence must not overlap (considering also the setup times):

$$\text{noOverlap}(\pi_m, \{s_{m,i,j} | i \in J', j \in J\}) \quad \forall m \in M \tag{42}$$

- Maximum contamination levels must not be violated:

$$cv_{k,m,i} \leq C^{\max}_{k,(jobAt_{m,i})} - \epsilon \quad \forall k \in K, m \in M, i \in \{1, \dots, |J|\} \tag{43}$$

- Jobs cannot be scheduled on ineligible machines:

$$\neg\text{presenceOf}(xm_{m,j}) \quad \forall j \in J, m \in M \setminus E_j \tag{44}$$

- Channel job interval variables to job predecessor variables:

$$prev_{m,j} = \text{typeOfPrev}(\pi_m, xm_{m,j}, -1) \quad \forall m \in M, j \in J \tag{45}$$

- Channel job predecessor variables to sequence position variables:

$$(prev_{m,j} = \omega) \Leftrightarrow (jobPos_{m,j} = 1) \quad \forall m \in M, j \in J \tag{46}$$

$$(prev_{m,j} = 0) \Leftrightarrow (jobPos_{m,j} = 0) \quad \forall m \in M, j \in J \tag{47}$$

$$(prev_{m,j} = i) \Rightarrow (jobPos_{m,i} = jobPos_{m,j} - 1) \quad \forall m \in M, i, j \in J \tag{48}$$

$$(jobPos_{m,i} = jobPos_{m,j} - 1 \wedge jobPos_{m,i} \neq 0) \Rightarrow (prev_{m,j} = i) \quad \forall m \in M, i, j \in J \tag{49}$$

- Channel sequence position variables to job position variables:

$$(jobPos_{m,i} = j) \Leftrightarrow (jobAt_{m,j} = i) \quad \forall m \in M, i \in J, j \in \{1, \dots, |J|\} \tag{50}$$

5.3 Objective Function

The objective function minimizes the squared tour tardiness of all tours:

$$minimize \quad \sum_{t \in T} \max(\{0\} \cup \{\text{endOf}(x_j) - d_t | \forall j \in tourJobs_t\})^2 \tag{51}$$

6 Metaheuristic Approach

In this section, we propose a local search approach using simulated annealing for the PMSP with contamination constraints. First, we describe the solution representation, the used cost function, and the generation of initial solutions. Then, we explain the generation of neighborhood solutions and further describe how a simulated annealing based acceptance function is utilized.

We note that a local search approach based on simulated annealing that partly use similar neighborhoods for another PMSP variant was proposed in [12]. However, we further introduce two additional neighborhood operators to deal with the unique properties of the problem proposed in this paper.

6.1 Solution Representation, Cost Function & Initial Solutions

We represent candidate solutions by using arrays storing the sequence of job ids assigned to each machine. The length of each array is set to the maximum number of all jobs, and empty positions in the arrays are aligned to the end and set to a null value. Any candidate solution ensures that regular jobs are scheduled exactly once on any machine, cleaning jobs however are optional. The value of the objective function for a given candidate solution is determined by calculating the earliest possible start time for each job in the sequence (i.e. either directly after the predecessor ends or at the release date). However, as candidate solutions may also cause constraint violations, we additionally include the number of violations V in the cost function $cost(S)$ that is used to evaluate a candidate solution S:

$$cost(S) = \sum_{t \in T} \max\{0, end_t - d_t\}^2 + V \cdot M \tag{52}$$

The cost function adds the number of constraint violations V multiplied by a given factor M to the objective function, where M is set to a large value so that a single hard constraint violation becomes incomparingly more expensive than any objective value. We determine the value V by counting the number of contamination level violations together with the number of eligible machine violations. Thereby, each job scheduled on an ineligible machine is counted as a violation and the number of contamination level violations is calculated by checking if the maximum contamination level is exceeded for each job position and factor.

To randomly construct initial solutions, we shuffle the list of jobs and then simply assign one job after the other to a randomly selected eligible machine. Note that this construction procedure may produce infeasible solutions as contamination level constraints may appear, however, local search usually can quickly repair infeasible solutions.

6.2 Search Neighborhoods

We use four different neighborhood operators for local search:

1. **Swap jobs**: This operator selects two jobs in the schedule and simply swaps their positions, where the jobs can be on the same machine or on different machines.
2. **Shift job**: A job is moved to another position in the schedule. Potential targets are any position between consecutive jobs as well as the start and end of any machine schedule.
3. **Insert cleaning job**: Inserts a single cleaning job into any position of the schedule. Similar as with the shift job neighborhood, the target position can be between any pair of consecutive scheduled jobs, or at the start/end of a machine schedule.
4. **Remove cleaning job**: Removes a single cleaning job that is currently scheduled.

Regarding the *swap jobs* and *shift job* operators, we additionally consider *block swap* and *block shift* versions where the main idea is to swap or shift blocks of up to k consecutively scheduled jobs at once, where k is a parameter given to the algorithm. For example, if $k = 3$ the *swap job* neighborhood would not only consider swapping two single jobs, but could potentially swap two blocks of consecutively scheduled jobs with block lengths of two or three. Similarly, a block shift neighborhood move could shift blocks of jobs to a new position in the schedule. The intuition behind these block moves is that short job sequences that work well regarding job processing times and contamination levels can be moved at once to find improving neighboring solutions without the need of performing solution quality worsening intermediate steps.

When dealing with large-scale real-life instance, exploring the complete neighborhood can quickly become computationally expensive. Thus, in the proposed metaheuristic we do not explore the full neighborhood, but instead randomly select a single move out of the complete neighborhood in each iteration. Thereby, we select a single random move per iteration in 2 steps: First, we randomly select one of the neighborhoods. Then, we uniformly sample a single move from the chosen neighborhood.

6.3 Neighborhood Move Acceptance

After a single random move is selected, we evaluate the change to the current solution's quality that would be caused by the move. Based on the result we then decide whether the move should be applied to the current solution. We use a move acceptance function based on simulated annealing [8] which ensures that a cost-improving move is always accepted, whereas a non-cost-improving move is only accepted with probability p that depends on the change in solution quality as well as the current temperature value T. Equation 53 shows how probability p is calculated based on the costs of the current solution S and the candidate solution S^* which has bigger costs than S.

$$p = exp(\frac{-(cost(S^*) - cost(S))}{T}) \tag{53}$$

Regarding the temperature T, we set the initial temperature T_{init} and the final temperature T_{final} by user defined parameters. The cooling rate, which determines how fast the temperature is lowered after each search iteration, is determined dynamically after each iteration in our approach. Thus, the actual cooling rate which is applied for the next iteration is calculated by looking at the average runtime per move and the remaining time budget. Thereby, we set the cooling rate to a value that ensures that the temperature converges to the final temperature value T_{final} at the end of the runtime, assuming the current average runtime per iteration.

7 Experimental Evaluation

In this section, we present the results from an extensive evaluation of all proposed approaches on realistic problem instances from the industry. First, we describe our experimental environment and the benchmark instances. Afterwards, we present and discuss the detailed computational results.

7.1 Experimental Environment

All experiments were run on 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. To evaluate the MIQP model we used Cplex 20.1 [1] and Gurobi 9.5 [7], whereas we used CPoptimizer 20.1 [11] to evaluate the CP models. As CPoptimizer does not support floating point variables, we had to adapt the direct model from Section 3 so that it uses dynamic expressions to capture the contamination levels for each job position. We did this by using similar modeling techniques regarding the contamination constraints as in Section 5, with the only difference that the auxiliary job position variables were channeled to the predecessor variables from the direct model instead of the interval variables. For CPoptimizer we set the relative optimality gap to 10^{-7}, besides that we used default parameters for all solvers, but restricted them to single-threaded solving. Further, we used reasonable values based on the size of the real-life instances for the model parameters h, u, v and ϵ. The local search parameters were set based on manual tuning trials: $k = 4$, $T_{init} = 10^{12}$, and $T_{end} = 10^{-4}$.

All approaches were given a time limit of 1 hour per instance. As the metaheuristic approach utilizes a randomly created initial solution as well as randomly generated moves, we conducted 10 repeated experimental runs per instance with the local search approach and present the mean costs in the final results.

Table 2 The upper bounds for each instance achieved by the proposed methods.

Inst.	CP direct	CP interval	MIQP Gurobi	MIQP Cplex	LS
I 1	**403617**	**403617**	406472	445837	**403617**
I 2	**394696**	**394696**	412688	**394696**	**394696**
I 3	**25702**	**25702**	30250	27702	**25702**
I 4	**2305**	**2305**	2393	3573	**2305**
I 5	39193760	**33707**	44530	52907	**33707**
I 6	**136288**	**136288**	143421	157525	**136288**
I 7	245418700	**959455**	18291650	24524154	**959455**
I 8	6571216	**20816**	5375830	23556806	**20816**
I 9	4901583	**97586**	15987063		**97586**
I 10		**106274**	32417279		**106274**
I 11	10539330	2434331			**2384267.3**
I 12	324562700	**696805**		135383618	697368.6
I 13	440502600	**1020957**			**1020957**
I 14	910107	**407439**	3098941	1826389	**407439**
I 15	7178927	**1454166**	121659015	51971121	**1454166**
I 16	1144669	**773924**	29362144		776526
I 17		**581404**	22263532	17479750	**581404**
I 18		**33251**	19796349	188558145	**33251**
I 19	1082993		6697489		**259614**

We gathered 19 problem instances that directly represent real-life scheduling scenarios from our industry partners to evaluate the proposed methods on realistic problems. All instances together with the detailed results are available for download at `https://doi.org/10.5281/zenodo.6797397`. Detailed size parameters of the instances can further be found in Appendix A.

We further experimented with different programmed search strategies in early experiments. For the final experimental results presented in this section we used the solver's default search strategy with the direct CP model and selected a search strategy that assigns values to the π sequence variables first for the interval based model, as these strategies performed best in the early experiments. Detailed information about the search strategies and related experiments are given in Appendix B.

Based on feedback from the reviewers we additionally experimented with another variant of the MIQP model that uses smaller big Ms in constraints 22, 25, and 27. Although the best dual bound found within 1 hour of runtime could be slightly improved for most instances, the best upper bound found with MIQP heuristics was better without these changes for the majority of instances. Thus, we decided to not include these changes in our final experiments.

7.2 Computational Results

A summary of the best upper bounds produced by all evaluated approaches is given in Table 2. The table shows the best upper bounds produced by the direct- and interval variable based CP models with CPoptimizer (CP direct/CP interval), the best upper bounds reached by the MIQP model with Gurobi (MIQP Gurobi) and Cplex (MIQP Cplex), as well as the mean cost results achieved over 10 runs with local search (LS). Overall best upper bounds per instance are formatted in bold face, and empty cells denote that no solution could be found within the runtime.

We see in Table 2 that the interval variable based CP model was the overall best performing exact method as it provided best solutions for 17 instances. Only instance 19 could not be solved, whereas the direct CP model and Gurobi could find a solution within the runtime. Local search also achieved best upper bounds in 17 cases, only for instances 11 and 16 the mean cost results were not on par with the CP approach. However, regarding instances 11 and 19 the metaheuristic achieved improved results over the exact methods.

Table 3 The lower bounds for each instance achieved by exact methods.

Inst.	CP direct	CP interval	MIQP Gurobi	MIQP Cplex
I 1	369489	**403617**	389058	388824
I 2	98575	**295211**	246813	229617
I 3	9377	**25702**	17273	15845
I 4	1370	**2305**	1936	1521
I 5	14885	**24086**	21284	21284
I 6	72197	**136288**	87066	82979
I 7	306161	**707234**	487155	322028
I 8	11601	**20816**	10517	11008
I 9	72097	**97586**	91988	62341
I 10	35145	**80015**	31561	324
I 11	90555	**412877**	277498	220423
I 12	491242	**604160**	577154	534484
I 13	634342	**1020957**	842490	802717
I 14	276676	**407439**	338875	330226
I 15	764585	**1268739**	1192610	1169638
I 16	412942	**773924**	600003	434803
I 17	352486	**528173**	461746	448509
I 18	20449	**32662**	28694	28694
I 19	223730	224186	**232434**	224946

The metaheuristic actually found solutions of similar quality over all 10 runs for all instances except for instances 11/12 where the solution costs were 2383478/2391371 in the best case and 696805/702441 in the worst case.

Table 3 further displays the best lower bounds achieved with exact methods. Columns 2-6 show from left to right: Lower bounds achieved with the direct CP model (CP direct), the interval variable model (CP interval), and the MIQP model with Gurobi and Cplex (MIQP Gurobi/MIQP Cplex). We see that the interval variable model produced the best lower bounds for instances 1-18, whereas Gurobi achieved the best lower bound for instance 19.

Finally, Table 4 summarizes the overall best lower bounds (LB) and upper bounds achieved by any of the exact methods (Exact) and compares it to the best upper bound achieved over all 10 runs by local search (LS). Further, the table includes the duality gap between the lower bound and the best upper bound in percentage (Gap), as well as the time in seconds required until the best upper bound was found by the exact and local search methods (Time (Exact) and Time (LS)). We see in the results that instances 1,3,4,6,8,9,13,14, and 16 could be solved to optimality by exact methods. Surprisingly, the metaheuristic approach produced cost equivalent or improved results compared to exact methods for most instances, as only for instance 16 a better solution was achieved with CP. This indicates that the proposed metaheuristic approach can efficiently escape local optima and thereby reach high-quality results for real-life instances. However, CP based methods also produced equal or better results for 17 instances and were necessary to guarantee optimal solutions. Additionally, we see in the results that exact methods using CP can find high-quality solutions quickly as for several instances the best bound was achieved within 60 seconds of runtime. However, for some instances the best solution was found later during the search process. The best results with local search were mostly found after about half of the given runtime budget was consumed. This is an expected result, given that the used simulated annealing scheme dynamically determines the cooling scheme so that the final temperature is reached at the end of the runtime.

We further compared the best schedules produced by the proposed methods with schedules created by human planners that currently utilize a construction heuristic to generate solutions for the same set of real-life instances in the industry. For proprietary reasons we cannot provide detailed results and additional information about the used heuristic, but our approaches could successfully improve the quality of the schedules and thereby reduce the number of

Table 4 The overall best lower bounds and best results achieved with exact and heuristic methods.

Inst.	LB	Gap	Exact	Time (Exact)	LS	Time (LS)
I 1	403617	0.00	**403617**	65	**403617**	1365.27
I 2	295211	25.21	**394696**	45.22	**394696**	1555.05
I 3	25702	0.00	**25702**	43.08	**25702**	1233.50
I 4	2305	0.00	**2305**	54.95	**2305**	1394.50
I 5	24086	28.54	**33707**	55.58	**33707**	1515.24
I 6	136288	0.00	**136288**	47.19	**136288**	1544.77
I 7	707234	26.29	**959455**	1142.8	**959455**	1445.58
I 8	20816	0.00	**20816**	70.33	**20816**	1462.24
I 9	97586	0.00	**97586**	128.54	**97586**	1577.79
I 10	80027	24.70	**106274**	859.68	**106274**	1745.08
I 11	412877	82.68	2434331	1225.75	**2383478**	1548.49
I 12	604160	13.30	**696805**	428.97	**696805**	1505.05
I 13	1020957	0.00	**1020957**	70.28	**1020957**	1484.74
I 14	407439	0.00	**407439**	617.8	**407439**	1729.00
I 15	1268739	12.75	**1454166**	2035.72	**1454166**	1615.52
I 16	773924	0.00	**773924**	284.15	776526	1615.68
I 17	528173	9.16	**581404**	228.03	**581404**	1607.26
I 18	32662	1.77	**33251**	281.08	**33251**	1366.23
I 19	232434	10.47	1082993	3600	**259614**	1446.79

delayed tours up to roughly 10%. Reducing delayed tours indicates a decent improvement as tour delays lead to large costs in practice. Furthermore, as the instances are not easy and the manual planning is done by experienced planners, a 10% reduction can be considered as a good enhancement.

8 Conclusion

In this work, we introduced a novel real-life PMSP from the agricultural industry and provided a set of challenging real-life instances that we gathered from industrial partners.

We proposed a direct- and an interval variable based CP approach as well as a MIQP approach and thereby considered alternative modeling techniques to efficiently capture unique aspects such as contamination constraints, optional cleaning jobs, and a tour tardiness objective. Furthermore, we investigated a metaheuristic based on local search and simulated annealing using problem specific neighborhood operators to approach large-scale instances.

An extensive experimental evaluation with the introduced real-life problem instances shows that the CP approach using the interval variable based model was the overall best performing exact method as it provided 9 optimal results and high-quality upper bounds for most instances. Additionally, the metaheuristic could provide solutions to all instances and thereby improved results of exact methods for two large-scale instances.

An interesting subject of future work could be to hybridize the proposed exact and metaheuristic techniques within a large-neighborhood search based approach.

References

1 IBM ILOG CPLEX Optimization Studio 20.1.0. User's manual for cplex, November 2021. URL: `https://www.ibm.com/docs/en/icos/20.1.0?topic=cplex-users-manual`.

2 Ali Allahverdi. The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345–378, October 2015.

3 Ali Allahverdi, Jatinder N. D Gupta, and Tariq Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 27(2):219–239, April 1999.

4 Ali Allahverdi, C. T. Ng, T. C. E. Cheng, and Mikhail Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, June 2008.

5 Abdoul Bitar, Stéphane Dauzère-Pérès, and Claude Yugma. Unrelated parallel machine scheduling with new criteria: Complexity and models. *Computers & Operations Research*, 132:105291, August 2021.

6 Quang-Vinh Dang, Thijs van Diessen, Tugce Martagan, and Ivo Adan. A matheuristic for parallel machine scheduling with tool replacements. *European Journal of Operational Research*, 291(2):640–660, June 2021.

7 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL: `https://www.gurobi.com`.

8 S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, May 1983.

9 Chuleeporn Kusoncum, Kanchana Sethanan, Rapeepan Pitakaso, and Richard F. Hartl. Heuristics with novel approaches for cyclical multiple parallel machine scheduling in sugarcane unloading systems. *International Journal of Production Research*, 59(8):2479–2497, April 2021.

10 Philippe Laborie. IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems. In Willem-Jan van Hoeve and John N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Lecture Notes in Computer Science, pages 148–162, Berlin, Heidelberg, 2009. Springer.

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

12 Maximilian Moser, Nysret Musliu, Andrea Schaerf, and Felix Winter. Exact and metaheuristic approaches for unrelated parallel machine scheduling. *Journal of Scheduling*, December 2021.

13 Juan C. Yepes-Borrero, Federico Perea, Rubén Ruiz, and Fulgencia Villa. Bi-objective parallel machine scheduling with additional resources during setups. *European Journal of Operational Research*, 292(2):443–455, July 2021.

14 Pinar Yunusoglu and Seyda Topaloglu Yildiz. Constraint programming approach for multi-resource-constrained unrelated parallel machine scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, 0(0):1–18, February 2021.

A Benchmark Instances

Table 5 summarizes the features of the 19 problem instances that directly represent real-life scheduling scenarios from our industry partners. The table displays in each row the instance id (I1-I19), the number of jobs ($|J|$) , the number of machines ($|M|$), the number of tours ($|T|$), and the number of contamination factors ($|K|$). Furthermore, Table 5 also includes the number of variables and constraints used in the direct CP model (Vars/Cons), the interval based model (I.Vars/I.Cons), and the MIQP model (M.Vars/M.Cons). We see that the MIQP formulation in general uses more variables but less constraints than the CP models. Further, the interval based model uses slightly more variables and constraints compared to the direct CP model.

B Programmed Search Strategies

We evaluated several search strategies for the direct- and interval variable based CP models together with CPoptimizer. These search strategies are based on variable- and value selection heuristics, which determine the order of the explored variables and the value assignments. This can play a critical role in reducing the search space that needs to be enumerated by the CP solver. As the search strategies define heuristics only on a subset of all variables, the solver's default strategy is used after all mentioned variables were fixed. Furthermore, ties are broken lexicographically. Table 6 describes the search strategies used with the direct- and interval variable based models.

Table 5 Size parameters of the real-life benchmark instances.

Inst.	$\|J\|$	$\|M\|$	$\|T\|$	$\|K\|$	**Vars**	**Cons**	**I.Vars**	**I.Cons**	**M.Vars**	**M.Cons**
I 1	26	2	41	58	874	5267470	897	5268554	183289	43798
I 2	16	2	37	58	824	5267209	840	5267828	110164	28379
I 3	15	2	37	58	819	5267184	833	5267761	76058	26854
I 4	18	2	34	58	834	5267264	846	5267965	68021	31473
I 5	28	2	42	58	884	5267522	907	5268710	199545	47099
I 6	21	2	43	58	849	5267341	869	5268178	152501	36753
I 7	76	2	76	58	1124	5268762	1174	5273691	764573	132270
I 8	67	2	78	58	1079	5268529	1128	5272594	658176	115596
I 9	77	2	82	58	1129	5268790	1181	5273814	808008	135722
I 10	94	2	82	58	1214	5269230	1274	5276173	1097447	171276
I 11	61	2	74	58	1049	5268380	1096	5271881	590600	104791
I 12	78	2	90	58	1134	5268816	1200	5273945	884457	137331
I 13	67	2	92	58	1079	5268535	1141	5272580	706558	114135
I 14	68	2	85	58	1084	5268557	1131	5272703	653867	116774
I 15	63	2	90	58	1059	5268428	1115	5272113	647114	107794
I 16	65	2	69	58	1069	5268478	1119	5272341	640025	111787
I 17	66	2	70	58	1074	5268506	1121	5272460	630976	112971
I 18	65	2	73	58	1069	5268481	1121	5272351	645552	111137
I 19	60	2	71	58	1044	5268350	1099	5271769	607280	101845

Table 7 summarizes the results on the 19 instances for all evaluated search strategies with the direct- and interval variable based CP models. Each row shows results for a single search strategy, where Columns 1-5 display from left to right: The model and search strategy (Search), the number of optimal solutions achieved (O), the number of solutions achieved (S), the number of best upper bounds achieved when compared with other strategies using the same model (B), and the number of provided optimality proofs.

The results displayed in Table 7 show that the direct model could find solutions for 16 instances with the solver's default search strategy, whereas other strategies could solve less problem instances, even though *direct-search1* could find more optimal solutions and best upper bounds. No search strategy was able to prove an optimum with the direct model (lower bounds achieved with the interval model were used to determine how many optimal solutions were found with the direct formulation). As *direct-default* could solve the most instances, we only present detailed results achieved by this strategy with the direct model in Section 7.

We further see in the results shown in Table 7 that the interval variable based model could successfully solve 18 instances with several search strategies, whereas the solver's default strategy could solve 13 instances. The search strategies *interval-search3* and *interval-search7* produced the overall best results as they could find 9 optimal solutions, achieved the best upper bounds for 18 instances, and provided 8 optimality proofs. Actually, they achieved the exact same upper bounds, and produced different lower bounds only for two instances. Both search types fix sequence variables first, which indicates that this was the most efficient strategy in our experiments. In Section 7 whenever we refer to the best results produced with the interval variable based model we mean results achieved by the *interval-search3* strategy, only for the best lower bounds presented in Table 3 we combined the best bounds achieved with *interval-search3* and *interval-search7*.

Table 6 Search strategies for the direct- and interval variable based CP models.

Search Strategy	Description
direct-default	Uses the solver's default search strategy for the direct model.
direct-search1	Starts with the assignment of all x decision variables. Variables with the smallest values in their domains are selected first and a minimum value first strategy is used for value selection.
direct-search2	As *direct-search1*, but uses a smallest domain size variable selection heuristic instead of the smallest domain value strategy.
direct-search3	As *direct-search1*, but operates on the *start* instead of the x variables.
direct-search4	As *direct-search2*, but operates on the *start* instead of the x variables.
direct-search5	As *direct-search1*, but operates on the $endTour$ instead of the x variables.
direct-search6	As *direct-search1*, but operates on the *end* instead of the x variables.
direct-search7	As *direct-search2*, but operates on the *end* instead of the x variables.
interval-default	Uses the solver's default search strategy for the interval variable model.
interval-search1	Assigns values to the xm interval variables first. Each interval chooses a presence status first and then assigns a start- and end time. Interval variables with a small start- and end date are fixed first.
interval-search2	As *interval-search1*, but operates on the x instead of the xm variables.
interval-search3	Assigns values to the π sequence variables first. Thereby, the full order of intervals associated to the sequences and the presence status of each interval are fixed before start- and end times are assigned in a later search phase.
interval-search4	As *interval-search1*, but operates on the xc instead of the xm variables.
interval-search5	Starts with *interval-search3* and then continues with *interval-search1*.
interval-search6	Starts with *interval-search3* and then continues with *interval-search2*.
interval-search7	Starts with *interval-search3* and then continues with *interval-search4*.

Table 7 Summarized results produced with different programmed search strategies.

Search	O	S	B	P
direct-default	4	16	9	0
direct-search1	6	12	11	0
direct-search2	4	14	9	0
direct-search3	0	5	0	0
direct-search4	0	5	0	0
direct-search5	0	5	0	0
direct-search6	0	6	0	0
direct-search7	0	6	0	0

Search	O	S	B	P
interval-direct	9	13	11	8
interval-search1	7	11	9	6
interval-search2	4	6	5	4
interval-search3	9	18	18	8
interval-search4	6	12	7	5
interval-search5	9	18	17	8
interval-search6	9	18	17	7
interval-search7	9	18	18	8