Non-overlapping Sequence-Dependent Setup Scheduling with Dedicated Tasks*

Marek Vlk[1,2], Antonin Novak[2,3], Zdenek Hanzalek[2], and Arnaud Malapert[4]

[1] Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics, Charles University, CZ
[2] Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, CZ
[3] Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, CZ
[4] Université Côte d'Azur, I3S, CNRS, FR
{marek.vlk, antonin.novak, zdenek.hanzalek}@cvut.cz, arnaud.malapert@unice.fr

Abstract. The paper deals with a parallel machines scheduling problem with dedicated tasks with sequence-dependent setup times that are subject to the non-overlapping constraint. This problem emerges in the productions where only one machine setter is available on the shop floor. We consider that setups are performed by a single person who cannot serve more than one machine at the same moment, i.e., the setups must not overlap in time. We show that the problem remains $\mathcal{NP}$-hard under the fixed sequence of tasks on each machine. To solve the problem, we propose an Integer Linear Programming formulation, five Constraint Programming models, and a hybrid heuristic algorithm LOFAS that leverages the strength of Integer Linear Programming for the Traveling Salesperson Problem (TSP) and the efficiency of Constraint Programming at sequencing problems minimizing makespan. Furthermore, we investigate the impact of the TSP solution quality on the overall objective value. The results show that LOFAS with a heuristic TSP solver achieves on average 10.5 % worse objective values but it scales up to 5000 tasks with 5 machines.

Keywords: Constrained Setup Times; Constraint Programming; Hybrid Heuristic

1 Introduction

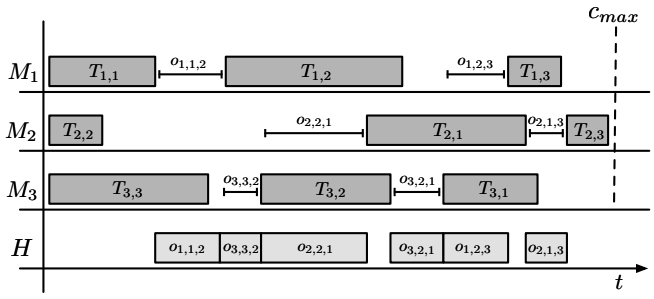
The trend of flexible manufacturing brings many challenges typically arising from low-volume batches of a larger number of product variants. One of such challenges is the minimization of setups that must be performed when a machine

* This work was funded by Ministry of Education, Youth and Sport of the Czech Republic within the project Cluster 4.0 number CZ.02.1.01/0.0/0.0/16_026/0008432.

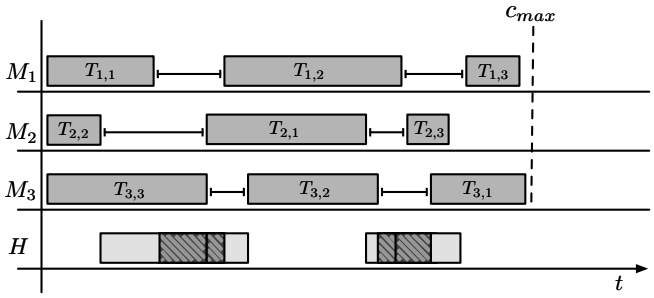
switches from one product variant to another. Switching involves, e.g., tool adjustment, which requires a machine setter to reconfigure the particular machine and make the desired adjustment.

In order to cut labor costs, the companies often try to limit the number of machine setters. As a basic case, only a single machine setter is present on the shop floor to perform the reconfigurations. Consequently, any schedule that does not account for the limited capacity of a machine setter is deemed to be infeasible, as a single machine setter cannot set two machines at the same time.

In this paper, we study a scheduling problem where the tasks are dedicated to the machines and have sequence-dependent setup times. Each setup occupies an extra unary resource, i.e., the machine setter, hence, setups must not overlap in time. The goal is to minimize the makespan of the overall schedule. To solve the problem, we design an Integer Linear Programming (ILP) model, five Constraint Programming (CP) models, and heuristic algorithm LOFAS.



(a) Feasible schedule.



(b) Infeasible schedule, setups are overlapping.

Fig. 1: The illustration of a schedule with three machines and three tasks to be processed on each machine [17].

The same problem was addressed in [17]. This is the revised and improved version of that paper. The main contributions of this revised and improved paper with respect to [17] are:

- a complexity result for the restricted problem with fixed sequences of tasks
- two new CP models utilizing the concept of cumulative function
- LOFAS algorithm enhanced by a heuristic for the subproblem allowing to solve larger instances
- experimental results showing the impact of the subproblem solution method on the solution quality

The rest of the paper is organized as follows. We first survey the existing work in the related area. Next, Sect. 3 gives the formal definition of the problem at hand. In Sect. 4, we describe an ILP model, while in Sect. 5 we introduce five CP models, and in Sect. 6 we propose the LOFAS algorithm. Finally, we present computational experiments in Sect. 7 and draw conclusions in Sect. 8.

2 Related Work

There is a myriad of papers on scheduling with sequence-dependent setup times or costs [2], proposing exact approaches [11] as well as various heuristics [15]. But the research on the problems where the setups require extra resource is scarce.

An unrelated parallel machine problem with machine and job sequence-dependent setup times, studied by [13], considers also the non-renewable resources that are assigned to each setup, which affects the amount of time the setup needs and which is also included in the objective function. On the other hand, how many setups may be performed at the same time is disregarded. The authors propose a Mixed Integer Programming formulation along with some static and dynamic dispatching heuristics.

A lotsizing and scheduling problem with a common setup operator is tackled in [14]. The authors give ILP formulations for what they refer to as a dynamic capacitated multi-item multi-machine one-setup-operator lotsizing problem. Indeed, the setups to be performed by the setup operator are considered to be scheduled such that they do not overlap. However, these setups are not sequence-dependent in the usual sense. The setups are associated to a product whose production is to be commenced right after the setup and thus the setup time, i.e., the processing time of the setup, does not depend on a pair of tasks but only on the succeeding task.

A complex problem that involves machines requiring setups that are to be performed by operators of different capabilities has been addressed in [5]. The authors modeled the whole problem in the time-indexed formulation and solved it by decomposing the problem into smaller subproblems using Lagrangian Relaxation and solving the subproblems using dynamic programming. A feasible solution is then composed of the solutions to the subproblems by heuristics, and, if impossible, the Lagrangian multipliers are updated using surrogate subgradient method as in [19]. The down side of this approach is that the time-indexed

formulation yields a model of pseudo-polynomial size. This is not suitable for our problem as it poses large processing and setup times.

In [17], the problem with sequence-dependent non-overlapping setups is introduced. The authors propose three CP models, ILP model and heuristics utilizing a decomposition of the problem. The resulting subproblems deal with the order of tasks on machines independently. The order of tasks is found by an ILP model with lazy subtour elimination and the order of setups by a CP model.

3 Problem Statement

The problem addressed in this paper consists of a set of machines and a set of independent non-preemptive tasks, each of which is dedicated to one particular machine where it will be processed. Also, there are sequence-dependent setup times on each machine. In addition, these setups are to be performed by a human operator who is referred to as a machine setter. Such a machine setter cannot perform two or more setups at the same time. It follows that the setups on all the machines must not overlap in time. Examples of a feasible and an infeasible schedule with 3 machines can be seen in Fig. 1. Even though the schedule in Fig. 1b on the machines contains setup times, such schedule is infeasible since it would require overlaps in the schedule for the machine setter.

The aim is to find a schedule that minimizes the completion time of the latest task. It is clear that the latest task is on some machine and not in the schedule of a machine setter since the completion time of the last setup is followed by at least one task on a machine.

3.1 Formal Definition

Let $M = \{M_1, ..., M_m\}$ be a set of machines and for each $M_i \in M$, let $T^{(i)} = \{T_{i,1}, ..., T_{i,n_i}\}$ be a set of tasks that are to be processed on machine M_i, and let $T = \bigcup_{M_i \in M} T^{(i)} = \{T_{1,1}, ..., T_{m,n_m}\}$ denote the set of all tasks. Each task $T_{i,j} \in T$ is specified by its processing time $p_{i,j} \in \mathbb{N}$. Let $s_{i,j} \in \mathbb{N}_0$ and $C_{i,j} \in \mathbb{N}$ be start time and completion time, respectively, of task $T_{i,j} \in T$, which are to be found. All tasks are non-preemptive, hence, $s_{i,j} + p_{i,j} = C_{i,j}$ must hold.

Each machine $M_i \in M$ performs one task at a time. Moreover, the setup times between two consecutive tasks processed on machine $M_i \in M$ are given in matrix $O^{(i)} \in \mathbb{N}^{n_i \times n_i}$, $O = \bigcup_{M_i \in M} O^{(i)}$. That is, $o_{i,j,j'} = (O^{(i)})_{j,j'}$ determines the minimal time distance between the start time of task $T_{i,j'}$ and the completion time of task $T_{i,j}$ if task $T_{i,j'}$ is to be processed on machine M_i right after task $T_{i,j}$, i.e., $s_{i,j'} - C_{i,j} \geq o_{i,j,j'}$ must hold.

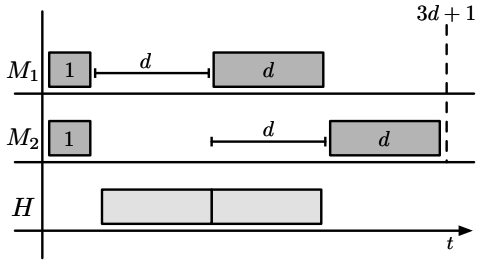
Let $H = \{h_1, \ldots, h_\ell\}$, where $\ell = \sum_{M_i \in M} n_i - 1$, be a set of setups that are to be performed by the machine setter. Each $h_k \in H$ corresponds to the setup of a pair of tasks that are scheduled to be processed in a row on some machine. Thus, function $st : H \to M \times T \times T$ is to be found. Also, $s_k \in \mathbb{N}_0$ and $C_k \in \mathbb{N}$ are start time and completion time of setup $h_k \in H$, which are to be found. Assuming $h_k \in H$ corresponds to the setup between tasks $T_{i,j} \in T$ and $T_{i,j'} \in T$, i.e.,

$st(h_k) = (M_i, T_{i,j}, T_{i,j'})$, it must hold that $s_k + o_{i,j,j'} = C_k$, also $C_{i,j} \leq s_k$, and $C_k \leq s_{i,j'}$. Finally, since the machine setter may perform at most one task at any time, it must hold that, for each $h_k, h_{k'} \in H, k \neq k'$, either $C_k \leq s_{k'}$ or $C_{k'} \leq s_k$.

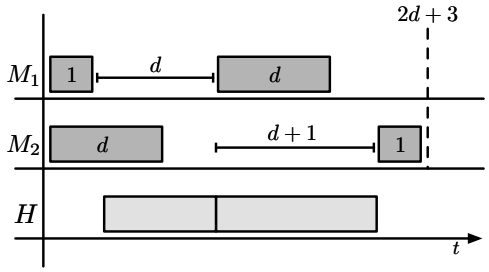
The objective is to find such a schedule that minimizes the makespan, i.e., the latest completion time of any task:

$$\min \max_{T_{i,j} \in T} C_{i,j} \tag{1}$$

We note that minimizing the makespan for each machine separately does not guarantee globally optimal solution. In fact, such a solution can be arbitrarily bad. Consider a problem depicted in Fig. 2. It consists of two machines, M_1



(a) An instance where optimal sequences on machines separately lead to a suboptimal solution.



(b) Suboptimal sequence on one machine yields a globally optimal solution.

Fig. 2: Solving the problem greedily for each machine separately can lead to arbitrarily bad solutions. The numbers depict the processing times of the tasks and setups [17].

and M_2, and two tasks on each machine, with processing times $p_{1,1} = p_{2,1} = 1, p_{1,2} = p_{2,2} = d$, where d is any constant greater than 2, and with setup times $o_{1,1,2} = o_{2,1,2} = d, o_{1,2,1} = o_{2,2,1} = d + 1$. Then, an optimal sequence on each

machine yields a solution of makespan $3d + 1$, whereas choosing suboptimal sequence on either of the machines gives optimal objective value $2d + 3$.

In the next subsection, we study the complexity of the problem where the order of the tasks is predefined.

3.2 Complexity of the Problem with Fixed Sequences

It is easy to see that the problem with sequence-dependent non-overlapping setups is strongly $\mathcal{NP}$-hard even for the case of one machine, i.e., $m = 1$, which can be shown by the reduction from the shortest Hamiltonian path problem. However, we show that even the restricted problem where the task sequences on each machine are fixed is $\mathcal{NP}$-hard as well, suggesting another source of hardness.

Definition 1 (Problem with fixed sequences). Let us denote the position of task $T_{i,j}$ on machine M_i by $\pi_i(j)$. Then the problem with fixed sequences of tasks is defined as follows.
INPUT: M, T, O, sequences of tasks for each machine $\pi_1, \pi_2, \ldots, \pi_m$.
OUTPUT: a feasible schedule with non-overlapping setups such that $\forall M_i \in M$: $\pi_i(j) < \pi_i(j') \implies s_{i,j} < s_{i,j'}$ minimizing $\max_{T_{i,j} \in T} C_{i,j}$.

Essentially, the problem is, given a fixed sequence of tasks on each machine, find a feasible schedule for setups H such that the latest completion time of any task is minimized.

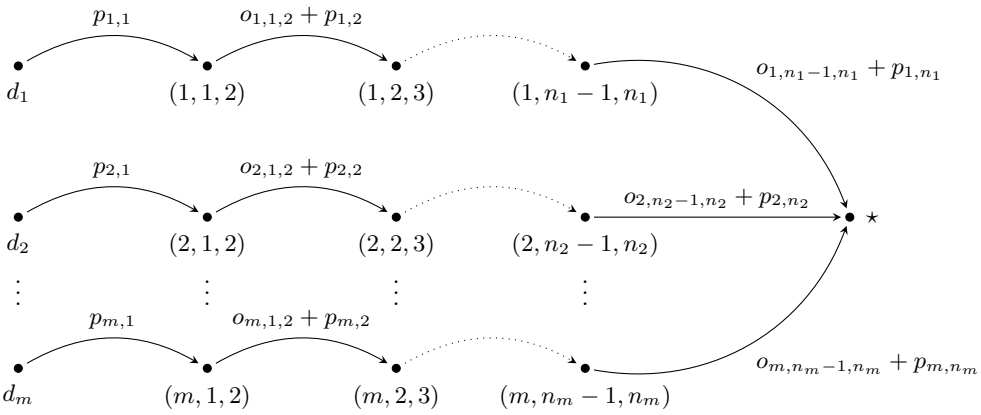


Fig. 3: Precedence graph of the problem with fixed sequences on machines.

It can be seen that the problem with fixed sequences is equivalent to $1|l_{ij} > 0, m\, n_1, \ldots, n_m\text{-chains}|C_{\max}$, i.e., a single machine scheduling problem with minimum time lags, where the precedence graph has a form of m chains of lengths $n_1, \ldots, n_m$, followed by a single common task (see Fig. 3). Indeed, consider the

following reduction. Let us remind that by a *minimum time lag* $l_{j,j'} > 0$ between tasks T_j and $T_{j'}$ (i.e., $T_j \xrightarrow{l_{j,j'}} T_{j'}$) we mean that $T_{j'}$ cannot start earlier than $s_{j'} \geq s_j + l_{j,j'}$, hence $l_{j,j'}$ defines a minimum distance between the corresponding start times.

Without loss of generality, let us assume in the reduction below that the sequence of tasks on each machine $M_i \in M$ is $\pi_i = (1, 2, \ldots, n_i)$. Then, the reduction goes as follows. The sequence of setups on each machine $M_i \in M$ defines a chain of $n_i - 1$ tasks plus one dummy task d_i. The processing time of a dummy task d_i is set to $p_i = 0$. The processing times of each task representing setups are equal to the corresponding setup time $o_{i,j,j'}$ given by the tasks sequences. A minimum time lag of length $p_{i,1}$ connects each dummy task d_i to the first setup on the corresponding machine. Then, in each chain corresponding to a machine M_i, the length of time lag between j-th setup and $(j+1)$-th setup has length $o_{i,j,j+1} + p_{i,j+1}$. Finally, the last setup in each chain is connected to the dummy task $\star$ with $p_\star = 0$ by the time lag of length $o_{i,n_i-1,n_i} + p_{i,n_i}$. See an example of the graph in Fig. 3.

It is easy to see that the reduction the other way around is possible directly as well, hence establishing the problem equivalence. Finally, we note that the problem $1|l_{ij} > 0, m\,2\text{-chains}|C_{\max}$ is known to be strongly $\mathcal{NP}$-hard by the reduction from 3-PARTITION problem due to [18]. This suggests that the constraint requiring non-overlapping setups introduces yet another source of hardness to the classical problem with sequence-dependent setup times. We will utilize the solution for the problem with fixed sequences further in Sect. 6 as a part of the proposed heuristic algorithm.

4 Integer Linear Programming Model

The proposed model is split into two parts. The first part handles scheduling of tasks on the machines using efficient *rank-based model* [10]. This modeling uses binary variables $x_{i,j,q}$ to encode whether task $T_{i,j} \in T^{(i)}$ is scheduled at q-th position in the permutation on machine $M_i \in M$. Another variable is $\tau_{i,q}$ denoting the start time of a task that is scheduled at q-th position in the permutation on machine $M_i \in M$.

The second part of the model resolves the question, in which order and when the setups are performed by a machine setter. There, we need to schedule all setups H, where the setup time π_k of the setup $h_k \in H$ is given by the corresponding pair of tasks on the machine. The ordering of setups is determined by $z_{k,l}$ binary variables, that take value 1 if setup h_l is scheduled after the setup h_k.

Let us denote the set of all natural numbers up to n as $[n] = \{1, \ldots, n\}$. We define the following function $\phi : H \to M \times [\max_{M_i \in M} n_i]$ (e.g., $\phi(h_k) = (M_i, q)$), that maps $h_k \in H$ to setups between the tasks scheduled at positions q and $q+1$ on machine $M_i \in M$. Since the time of such setup is a variable (i.e., it depends on the pair of consecutive tasks on M_i), the modeling with *rank-based model* would contain non-linear expressions. Therefore, we use the a disjunctive

model [3,4] that admits processing times given as variables. Its disadvantage over the rank-based model is that it introduces a *big M* constant in the constraints, whereas the rank-based model does not. See Fig. 4 for the meaning of variables.

The full model is stated as:

$$\min \quad c_{max} \tag{2}$$

s.t.

$$c_{max} \geq \tau_{i,n_i} + \sum_{T_{i,j} \in T^{(i)}} p_{i,j} \cdot x_{i,j,n_i} \quad \forall M_i \in M \tag{3}$$

$$\sum_{q \in [n_i]} x_{i,j,q} = 1 \quad \forall M_i \in M, \forall T_{i,j} \in T^{(i)} \tag{4}$$

$$\sum_{T_{i,j} \in T^{(i)}} x_{i,j,q} = 1 \quad \forall M_i \in M, \forall q \in [n_i] \tag{5}$$

$$s_k + \pi_k \leq s_l + \mathcal{M} \cdot (1 - z_{k,l}) \quad \forall h_l, h_k \in H : l < k \tag{6}$$

$$s_l + \pi_l \leq s_k + \mathcal{M} \cdot z_{k,l} \quad \forall h_l, h_k \in H : l < k \tag{7}$$

$$\pi_k \geq o_{i,j,j'} \cdot (x_{i,j,q} + x_{i,j',q+1} - 1) \quad \forall h_k \in H : \phi(h_k) = (M_i, q), \forall T_{i,j}, T_{i,j'} \in T^{(i)} \tag{8}$$

$$s_k + \pi_k \leq \tau_{i,q+1} \quad \forall h_k \in H : \phi(h_k) = (M_i, q) \tag{9}$$

$$s_k \geq \tau_{i,q} + \sum_{T_{i,j} \in T^{(i)}} p_{i,j} \cdot x_{i,j,q} \quad \forall h_k \in H : \phi(h_k) = (M_i, q) \tag{10}$$

where

$$c_{max} \in \mathbb{R}_0^+ \tag{11}$$

$$\tau_{i,q} \in \mathbb{R}_0^+ \quad \forall M_i \in M, \forall q \in [n_i] \tag{12}$$

$$s_k, \pi_k \in \mathbb{R}_0^+ \quad \forall h_k \in H \tag{13}$$

$$x_{i,j,q} \in \{0,1\} \quad \forall M_i \in M, \forall T_{i,j} \in T^{(i)}, \forall q \in [n_i] \tag{14}$$

$$z_{k,l} \in \{0,1\} \quad \forall h_k, h_l \in H : l < k \tag{15}$$

The constraint (3) computes makespan of the schedule while constraints (4)–(5) states that each task occupies exactly one position in the permutation and that each position is occupied by exactly one task. Constraints (6) and (7) guarantee that setups do not overlap. $\mathcal{M}$ is a constant that can be set as $|H| \cdot \max_i O^{(i)}$. Constraint (8) sets processing time π_k of the setup $h_k \in H$ to $o_{i,j,j'}$ if task $T_{i,j'}$ is scheduled on machine M_i right after task $T_{i,j}$. Constraints (9) and (10) are used to avoid conflicts on machines. The constraint (9) states that a task cannot start before its preceding setup finishes. Similarly, the constraint (10) states that a setup is scheduled after the corresponding task on the machine finishes.

4.1 Additional Improvements

We use the following improvements of the model that have a positive effect on the solver performance.

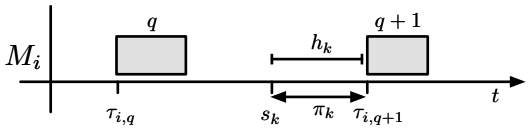


Fig. 4: Meaning of the variables in the model [17].

1. **Warm Starts.** The solver is supplied with an initial solution. It solves a relaxed problem, where it relaxes on the condition that setups do not overlap. Such solution is obtained by solving the shortest Hamiltonian path problem given by setup time matrix $O^{(i)}$ independently for each machine $M_i \in M$. Since such solution might be infeasible for the original problem, we transform it in a polynomial time into a feasible one. It is done in the following way. Since now the permutations on machines are fixed, the problem is reduced to the problem with minimum time lags on a single machine, as described in Sect. 3.2. The solution is constructed by scheduling first m setups in the machine order, then followed by next m setups etc. Hence, for the case of fixed permutations shown in Fig. 3, the order is $(1,1,2) \prec (2,1,2) \prec \ldots \prec (m,1,2) \prec (1,2,3) \prec \ldots \prec (m, n_m - 1, n_m)$.
2. **Lower Bounds.** We supply a lower bound on c_{max} variable given as the maximum of all best proven lower bounds for corresponding shortest Hamiltonian path problem on each machine.
3. **Pruning of Variables.** We can reduce the number of variables in the model due to the structure of the problem. We fix values of some of the $z_{k,l}$ variables according to the following rule. Let $h_k, h_l \in H$ such that $\phi(h_k) = (M_i, q)$ and $\phi(h_l) = (M_i, v)$ for any $M_i \in M$. Then, $q < v \Rightarrow z_{k,l} = 1$ holds in some optimal solution. Note that the rule holds only for setups following from the same machine.

   The rule states that the relative order of setups on the same machine is determined by the natural ordering of task positions on that machine. See for example setups $o_{1,1,2}$ and $o_{1,2,3}$ in Fig. 1. Since these setups follow from the same machine, their relative order is already predetermined by positions of the respective tasks. Essentially, the rule fixes the precedences according to the underlying precedence graph, as e.g., shown in Fig. 3.

5 Constraint Programming Models

A next way how the problem at hand can be tackled is by the Constraint Programming (CP) formalism, where special global constraints modeling resources and efficient filtering algorithms are used [16]. CP works with so-called *interval variables* whose start time and completion time are denoted by predicates $StartOf$ and $EndOf$, and the difference between the completion time and the start time of the interval variable can be set using predicate $LengthOf$.

We construct the CP models as follows. We introduce interval variables $I_{i,j}$ for each $T_{i,j} \in T$, and the lengths of these interval variables are set to the corresponding processing times:

$$LengthOf(I_{i,j}) = p_{i,j} \tag{16}$$

The sequence is resolved using the $NoOverlap$ constraint. The $NoOverlap(I)$ constraint on a set I of interval variables states that it constitutes a chain of non-overlapping interval variables, any interval variable in the chain being constrained to be completed before the start of the next interval variable in the chain. In addition, the $NoOverlap(I,\ O^{(i)})$ constraint is given a so-called *transition distance* matrix $O^{(i)}$, which expresses a minimal delay that must elapse between two successive interval variables. More precisely, if $I_{i,j}, I_{i,j'} \in I$, then $(O^{(i)})_{j,j'}$ gives a minimal allowed time difference between $StartOf(I_{j'})$ and $EndOf(I_j)$. Hence, the following constraint is imposed, $\forall M_i \in M$:

$$NoOverlap\Big(\bigcup_{T_{i,j} \in T^{(i)}} \{I_{i,j}\}\ ,\ O^{(i)}\Big) \tag{17}$$

The objective function is to minimize the makespan:

$$\min \max_{T_{i,j} \in T} EndOf(I_{i,j}) \tag{18}$$

This model would already solve the problem if the setups were not required to be non-overlapping. In what follows we describe three ways how the non-overlapping setups are resolved. Constraints (16)–(18) are part of each of the following models.

5.1 CP1: with Implications

Let us introduce $I^{st}_{i,j}$ for each $T_{i,j} \in T$ representing the setup after task $T_{i,j}$. There is $\sum_{M_i \in M} n_i$ such variables. As the interval variable $I^{st}_{i,j}$ represents the setup after task $T_{i,j}$, we use the constraint $EndBeforeStart(I_1, I_2)$, which ensures that interval variable I_1 is completed before interval variable I_2 can start. Thus, the following constraint needs to be added, $\forall M_i \in M, \forall T_{i,j} \in T^{(i)}$:

$$EndBeforeStart(I_{i,j}, I^{st}_{i,j}) \tag{19}$$

To ensure that the setups do not overlap in time is enforced through the following constraint:

$$NoOverlap\Big(\bigcup_{T_{i,j} \in T} \{I^{st}_{i,j}\}\Big) \tag{20}$$

Notice that this constraint is unique and it is over all the interval variables representing setups on all machines. This $NoOverlap$ constraint does not need any transition distance matrix as the default values 0 are desired.

Since it is not known a priori which task will follow task $T_{i,j}$, the quadratic number of implications determining the precedences and lengths of the setups must be imposed. For this purpose, the predicate $Next$[5] is used. $Next(I)$ equals the interval variable that is to be processed in the chain right after interval variable I. Thus, the following constraints are added, $\forall M_i \in M, \forall T_{i,j}, T_{i,j'} \in T^{(i)}, j \neq j'$:

$$Next(I_{i,j}) = I_{i,j'} \Rightarrow EndOf(I^{st}_{i,j}) \leq StartOf(I_{i,j'}) \tag{21}$$

$$Next(I_{i,j}) = I_{i,j'} \Rightarrow LengthOf(I^{st}_{i,j}) = o_{i,j,j'} \tag{22}$$

Note that the special value when an interval variable is the last one in the chain is used to turn the last setup on a machine into a dummy one.

5.2 CP2: with Element Constraints

Setting the lengths of the setups can be substituted by the element constraint, which might be beneficial as global constraints are usually more efficient. More precisely, this model contains also constraints (19), (20), and (21), but constraint (22) is substituted as follows.

Assume the construct $Element(Array, k)$ returns the k-th element of $Array$, $(O^{(i)})_j$ is the j-th row of matrix $O^{(i)}$, and $IndexOfNext(I_{i,j})$[6] returns the index of the interval variable that is to be processed right after $I_{i,j}$. Then the following constraint is added, for each $I^{st}_{i,j}$:

$$LengthOf(I^{st}_{i,j}) \;=\; Element\big((O^{(i)})_j,\; IndexOfNext(I_{i,j})\big) \tag{23}$$

5.3 CP3: with Optional Interval Variables

In this model, we use the concept of *optional interval variables* [8]. An optional interval variable can be set to be *present* or *absent*. The predicate $PresenceOf$ is used to determine whether or not the interval variable is present in the resulting schedule. Whenever an optional interval variable is absent, all the constraints that are associated with that optional interval variable are implicitly satisfied and predicates $StartOf$, $EndOf$, and $LengthOf$ are set to 0.

Hence, we introduce optional interval variable $I^{opt}_{i,j,j'}$ for each pair of distinct tasks on the same machine, i.e., $\forall M_i \in M, \forall T_{i,j}, T_{i,j'} \in T^{(i)}, j \neq j'$. There are $\sum_{M_i \in M} n_i(n_i - 1)$ such variables. The lengths of these interval variables are set to corresponding setup times:

$$LengthOf(I^{opt}_{i,j,j'}) = o_{i,j,j'} \tag{24}$$

To ensure that the machine setter does not perform more than one task at the same time, the following constraint is added:

[5] Note that in the IBM CP Optimizer, the function TypeOfNext is used
[6] Again, in the IBM CP Optimizer, the function TypeOfNext is used

$$NoOverlap(\bigcup_{\substack{T_{i,j},T_{i,j'}\in T \\ j\neq j'}} \{I^{opt}_{i,j,j'}\}) \tag{25}$$

In this case, to ensure that the setups are indeed processed in between two consecutive tasks, we use the constraint $EndBeforeStart(I_1, I_2)$, which ensures that interval variable I_1 is completed before interval variable I_2 can start, but if either of the interval variables is absent, the constraint is implicitly satisfied. Thus, the following constraints are added, $\forall I^{opt}_{i,j,j'}$:

$$EndBeforeStart(I_{i,j}, I^{opt}_{i,j,j'}) \tag{26}$$

$$EndBeforeStart(I^{opt}_{i,j,j'}, I_{i,j'}) \tag{27}$$

Finally, in order to ensure the correct presence of optional interval variables, the predicate $PresenceOf$ is used. Thus, the following constraint is imposed, $\forall I^{opt}_{i,j,j'}$:

$$PresenceOf(I^{opt}_{i,j,j'}) \Leftrightarrow Next(I_{i,j}) = I_{i,j'} \tag{28}$$

Notice that each $I_{i,j}$ (except for the last one on a machine) is followed by exactly one setup. Thus, we tried using a special constraint called *Alternative*, which ensures that exactly one interval variable from a set of variables is present. However, preliminary experiments showed that adding this constraint is counterproductive.

5.4 CP4: with Cumulative Function

In this model, the machine setter is represented as a cumulative function to avoid the quadratic number of constraints of CP1 and CP2 or the quadratic number of optional task variables of CP3. The definition of the non-negative cumulative function has a linear number of terms.

Again, we use interval variables $I_{i,j}$ for each $T_{i,j} \in T$ to model the execution of each task $T_{i,j} \in T$ and $I^{st}_{i,j}$ for each $T_{i,j} \in T$ representing the setup after task $T_{i,j}$. See Fig. 5 for the illustration of main concepts.

The idea now is that the lengths of the interval variables are not fixed. The length of interval variables $I_{i,j}$ is increased to wait for the machine setter if he is busy on another machine, i.e., the length of interval variable $I_{i,j}$ at least the processing time task $p_{i,j}$ but may be prolonged until the machines setter is available. Thus, the constraints (16) on the lengths of the interval variables $I_{i,j}$ are relaxed because they must only be greater than the corresponding processing times:

$$LengthOf(I_{i,j}) \geq p_{i,j} \tag{29}$$

The length of the setup executed by the machine setter will be determined by the corresponding tasks. In particular, as the last setup on a machine becomes

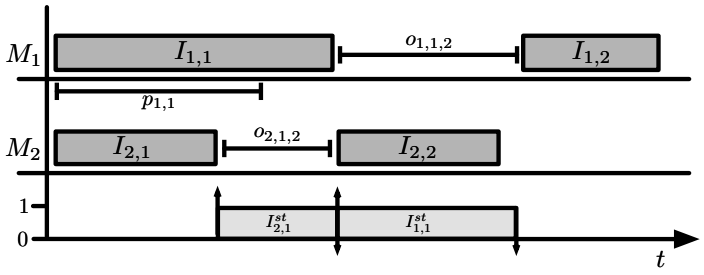


Fig. 5: Illustration of variables and constraints for CP4. The length of $I_{1,1}$ is greater than $p_{1,1}$ as the completion of $I_{1,1}$ must wait for the machine setter to complete the setup $I^{st}_{2,1}$.

a dummy setup (of zero length), we merely set the length of the setup variables to be at least zero:

$$LengthOf(I^{st}_{i,j}) \geq 0 \tag{30}$$

The cumulative function is built thanks to primitive $Pulse(a, h)$ that specifies that h unit of resource is used during interval a. The cumulative function is composed of $Pulse$ terms for each $I^{st}_{i,j}$ representing the usage of a machine setter, and the cumulative function must remain lower than 1 because there is a single machine setter:

$$\sum_{T_{i,j} \in T} Pulse(I^{st}_{i,j}, 1) \leq 1 \tag{31}$$

What remains to be done is to synchronize start and completion times between the tasks and setups. This is done using the constraint $EndAtStart(I_1, I_2)$, which ensures that interval variable I_1 is completed exactly when interval variable I_2 starts, and by $StartOfNext(I_{i,j})$, which gives the start time of the interval variable that is to be processed right after $I_{i,j}$. Thus, the following constraints are added, for each $I^{st}_{i,j}$:

$$EndAtStart(I_{i,j}, I^{st}_{i,j}) \tag{32}$$

$$EndOf(I^{st}_{i,j}) \geq StartOfNext(I_{i,j}) \tag{33}$$

Note that the inequality in constraint (33) is necessary because $StartOfNext$ gives 0 for the last task on a machine and thus the completion time of the last setup is equal to its start time (hence the length of the setup is 0). Also, note that the setups cannot be shorter than required due to constraint (17).

5.5 CP5: without Setup Variables

This model exploits the same idea as CP4, but we can go even further and completely omit the interval variables representing the setups. In this model,

the interval variables $I_{i,j}$ are, again, relaxed because they must only be greater than the corresponding processing times, i.e., constraint (29) is kept.

The main difference in this model is that the cumulative function only requires the introduction of interval variables S_i, one for each machine M_i. See Fig. 6 for the illustration of main concepts. Variable S_i starts with the first task of the machine M_i and ends with the last task, which is enforced by the *Span* constraint:

$$Span\big(S_i, \bigcup_{T_{i,j} \in T^{(i)}} \{I_{i,j}\}\big) \tag{34}$$

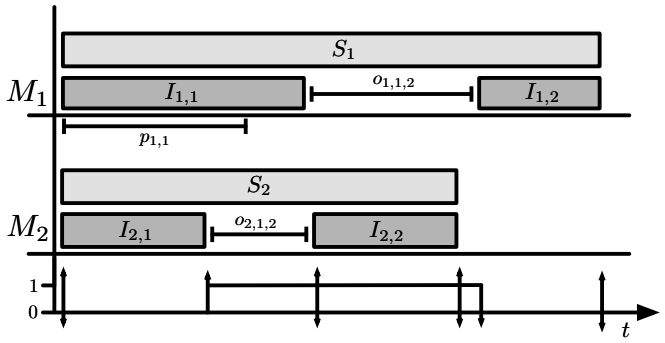


Fig. 6: Illustration of variables and constraints for CP5.

The cumulative function is now realized as follows:

$$\sum_{M_i \in M} Pulse(S_i, 1) - \sum_{T_{i,j} \in T} Pulse(I_{i,j}, 1) \leq 1 \tag{35}$$

The first term enforces that the cumulative function remains non-negative when the first task $T_{i,j}$ of the machine M_i starts and that the machine setter is not required at the end of the last task $T_{i,j}$ of the machine M_i. The second term enforces that the machine setter is available when a task $T_{i,j}$ ends, possibly after a waiting time as allowed by constraints (29), and that the machine setter becomes available again when a task $T_{i,j}$ starts.

Despite the lowest number of variables, the main drawback of this model is that the schedule of the machine setter becomes implicit which leads to less filtering.

5.6 Additional Improvements

We use the following improvements:

1. **Search Phases.** Automatic search in the solver is well tuned-up for most types of problems, leveraging the newest knowledge pertaining to variable selection and value ordering heuristics. In our case, however, preliminary results showed that the solver struggles to find any feasible solution already for small instances. It is clear that it is easy to find some feasible solution, e.g., by setting an arbitrary order of tasks on machines and then shifting the tasks to the right such that the setups do not overlap. To make the solver find some feasible solution more quickly, we set the search phases such that the sequences on machines are resolved first, and then the sequences of setups for the machine setter are resolved. This is included in all the CP models described.
2. **Warm Starts.** Similarly to improvement (1) in Sect. 4.1, we boost the performance by providing the solver with a starting point. We do this only for CP3 as the preliminary numerical experiments showed a slight superiority of CP3.
   More precisely, we first find an optimal sequence of tasks minimizing makespan on each machine separately and then we set those interval variables $I^{opt}_{i,j,j'}$ to be present if $T_{i,j'}$ is sequenced directly after $T_{i,j}$ on machine M_i. This is all that we set as the starting point. Notice that unlike in Sect. 4.1, we do not calculate the complete solution but we let the solver do it. The solver then quickly completes the assignment of all the variables such that it gets a solution of reasonably good objective value.
   Note that the optimal sequences on machines are solved using ILP so it can be seen as a hybrid approach. This model with warm starts is in what follows referred to as *CP3ws*.

6 LOFAS Heuristics

We propose an approach that guides the solver quickly towards solutions of very good quality but cannot guarantee optimality of what is found. There are two main phases of this approach. In the first phase, the model is decomposed such that its subproblems are solved optimally or near-optimally and then the solutions of the subproblems are put together so as to make a feasible solution of the whole problem. In the second phase, the solution found is locally improved by repeatedly adjusting the solution in promising areas. More details follow.

6.1 Decomposition Phase

The idea of the model decomposition is as follows. First, we find a sequence of tasks minimizing makespan on each machine separately. Second, given these sequences on each machine, the setups to be performed are known, hence, the lengths of the setups are fixed as well as the precedence constraints with respect to the tasks on machines. This is the problem with fixed permutations described in Sect. 3.2.

The pseudocode for obtaining the initial solution is given in Algorithm 1. It takes one machine at a time and finds a sequence for it while minimizing

makespan. The time limit for the computation of one sequence on a machine is given in such a way that there is a proportional remaining time limit for the rest of the algorithm. SEQ(i, $TimeLimit$) returns the best sequence it finds on machine $M_i \in M$ in the given $TimeLimit$. The $TimeLimit$ is computed using $RemainingTime()$, which is the time limit for the entire run of the algorithm minus the time that already elapsed from the beginning of the run of the algorithm. Just in case that no sequence is found in the given $TimeLimit$ yet, the search is allowed to continue until a first sequence is found (or up to $RemainingTime()$).

In the end, the solution is found using the knowledge of the sequences on each machine $M_i \in M$.

Algorithm 1 Solving the decomposed model.

```
function SolveDecomposed
    for each M_i ∈ M do
        TimeLimit ← RemainingTime()/(m − i + 2)
        Seq_i ← Seq(i, TimeLimit)
    end for
    Return Solve(Seq, RemainingTime())
end function
```

Clearly, this decomposition may lead to a schedule arbitrarily far from the optimum, as shown in Sect. 3.1. Hence, we apply the improving phase, that explores other sequences on the machines.

6.2 Improving Phase

Once we have some solution to the problem, the idea of the heuristic is to improve it applying the techniques known as local search [7] and large neighborhood search [12].

It is clear that in order to improve the solution, something needs to be changed on the *critical path*, which is such a sequence of setups and tasks on machines that the completion time of the last task is equal to the makespan and that none of these tasks and setups can be shifted to the left without violating resource constraints (see an example in Fig. 7). Hence, we find the critical path first.

The most promising place to be changed on the critical path could be the longest setup. Hence, we find the longest setup on the critical path, then we prohibit the two consecutive tasks corresponding to the setup from being processed in a row again and re-optimize the sequence on the machine in question. Two tasks are precluded from following one another by setting the corresponding setup time to infinite value. Also, we add extra constraint restricting the makespan to be less than the incumbent best objective value found. The makespan on one machine being equal to or greater than the incumbent best objective value found cannot lead to a better solution.

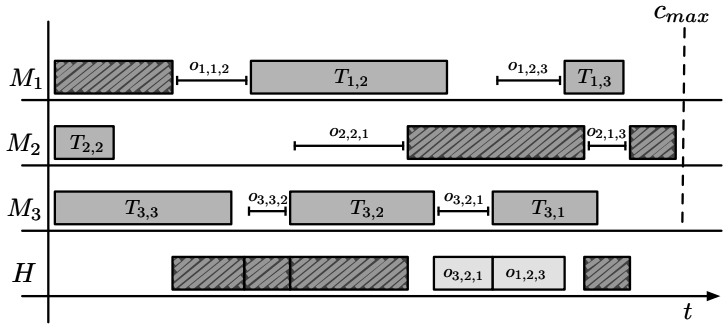


Fig. 7: Illustration of a critical path depicted by dashed rectangles [17].

After a new sequence is found, the solution to the whole problem is again re-optimized subject to the new sequence. The algorithm continues this way until the sequence re-optimization returns infeasible, which happens due to the extra constraint restricting the makespan. It means that the solution quality deteriorated too much and it is unlikely to find a better solution locally at this state. Thus, the algorithm reverts to the initial solution obtained from the decomposed model, restores the original setup times matrices, and tries to prohibit another setup time on the critical path. For this purpose, the list of *nogoods* to be tried is computed once from the first critical path, which is just a list of setups on the critical path sorted in non-increasing order of their lengths. The whole iterative process is repeated until the total time limit is exceeded or all the nogoods are tried.

The entire heuristic algorithm is hereafter referred to as LOFAS (Local Optimization for Avoided Setup). The pseudocode is given in Algorithm 2.

Preliminary experiments confirmed the well-known facts that ILP using lazy approach is very efficient for searching an optimal sequence on one resource, and CP is more efficient for minimizing makespan when the lengths of interval variables and the precedences are fixed. Nevertheless, for instances with many tasks, the solution involving ILP might be computationally infeasible. Hence, we also propose to find a suboptimal sequence for each machine by a heuristic method. Thus, in what follows, we distinguish the following two variants of the algorithm

1. **Exact subproblem.** The sequence is found by ILP with lazy subtour elimination, as described in [17]. In the experiments below, we denote this variant as *LOFAS*.
2. **Heuristic subproblem.** The suboptimal sequence is found by *Guided Local Search* algorithm implemented in Google's *OR-Tools* [1]. Note that this algorithm is a metaheuristic, hence, it consumes all the time limit assigned even if the objective value is not improving for several iterations (i.e., cannot

Algorithm 2 Local Optimization for Avoided Setup [17].

function LOFAS
  $S^{init} \leftarrow$ SolveDecomposed
  $S^{best} \leftarrow S^{init}$
  $P_{crit} \leftarrow$ critical path in S^{init}
  $nogoods \leftarrow \{h_k \in H \cap P_{crit}\}$
  sort $nogoods$ in non-increasing order of lengths
  for each $h_k \in nogoods$ **do**
    $h_{k'} \leftarrow h_k$
    while true **do**
      $(M_i, T_{i,j}, T_{i,j'}) \leftarrow st(h_{k'})$
      $o_{i,j,j'} \leftarrow \infty$
      impose constraint: $\max_{T_{i,j} \in T^{(i)}} C_{i,j} < ObjVal(S^{best})$
      $Seq_i \leftarrow$ Seq(i, $RemainingTime()/2$)
      if Seq_i **is** *infeasible* **then**
        Revert to S^{init}
        Restore original $O^{(i)}, \forall M_i \in M$
        break
      end if
      $S^{new} \leftarrow$ Solve(Seq, $RemainingTime()$)
      if $ObjVal(S^{best}) > ObjVal(S^{new})$ **then**
        $S^{best} \leftarrow S^{new}$
      end if
      if $RemainingTime() \leq 0$ **then**
        return S^{best}
      end if
      $P_{crit} \leftarrow$ critical path in S^{new}
      $h_{k'} \leftarrow$ longest setup $\in \{h_k \in H \cap P_{crit}\}$
    end while
  end for
  return S^{best}
end function

prove optimality). In the experiments, we denote this variant as *LOFAS /w heur*.

7 Experimental Results

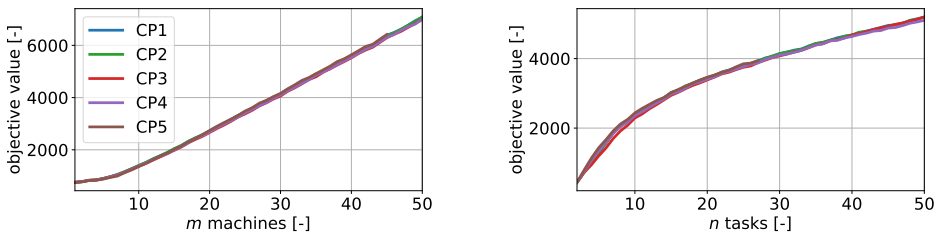
For the implementation of the constraint programming approaches, we used the IBM CP Optimizer version 12.9 [9]. The only parameter that we adjusted is `Workers`, which is the number of threads the solver can use and which we set to 1. In Google's *OR-Tools*, we only set `LocalSearchMetaheuristic` to `GuidedLocalSearch`.

For the integer programming approach, we used Gurobi solver version 8.1 [6]. The parameters that we adjust are `Threads`, which we set to 1, and `MIPFocus`, which we set to 1 in order to make the solver focus more on finding solutions of better quality rather than proving optimality. We note that parameters tuning with Gurobi Tuning Tool did not produce better values over the baseline ones.

The experiments were run on a Dell PC with an Intel® Core™ i7-4610M processor running at 3.00 GHz with 16 GB of RAM. We used a time limit of 60 seconds per problem instance.

7.1 Problem Instances

We evaluated the approaches on randomly generated instances of various sizes with the number of machines m ranging from 1 to 50 and the number of tasks on each machine $n_i = n$, $\forall M_i \in M$, ranging from 2 to 50. Thus, we generated $50 \times 49 = 2450$ instances in total. Processing times of all the tasks and setup times are chosen uniformly at random from the interval $[1, 50]$. Instances are publicly available at https://github.com/CTU-IIG/NonOverlappingSetupsScheduling.



(a) Mean objective value for different number of machines m.

(b) Mean objective value for different number of tasks n.

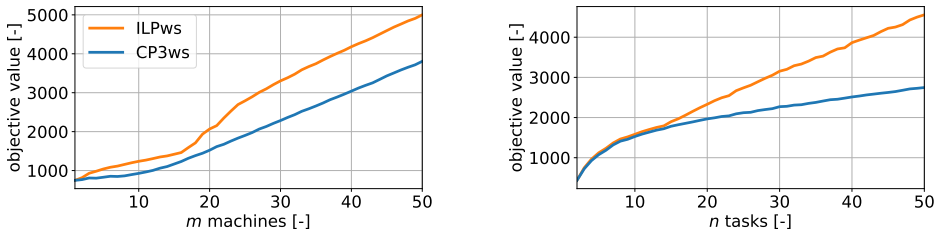
Fig. 8: Comparison of CP models.

7.2 Scalability with Respect to Machines and Tasks

Fig. 8a shows the dependence of the best objective value found by CP models within the 60s time limit on the number of machines, averaged over the various number of tasks. Analogically, Fig. 8b shows the dependence of the best objective value on the number of tasks, averaged over the varying number of machines.

The results show that the performances of CP models are almost equal (the graphs almost amalgamate). However, it can be seen that the curve of CP5 is not complete. It is because CP5 fails to find any solution for 88 of the largest instances, despite having the lowest number of variables. We note that CP3 is the best on average but the advantage is negligible. On the other hand, CP4 has better worst-case performance. We will no longer distinguish between the CP models and we will use the minimum of all five CP models that will be referred to as *CPmin*.

The comparison of CP3ws to ILPws is shown in Fig. 9. Recall that both the approaches get a warm start in a certain sense. The results confirm lower performance of the ILP approach. When the ILP approach model did not get the initial solution as a warm start, it was not able to find any solution even for very small instances (i.e., 2 machines and 8 tasks). In fact, the objective value found by the ILPws is often the objective value of the greedy initial solution given as the warm start (i.e., Sect. 4.1).



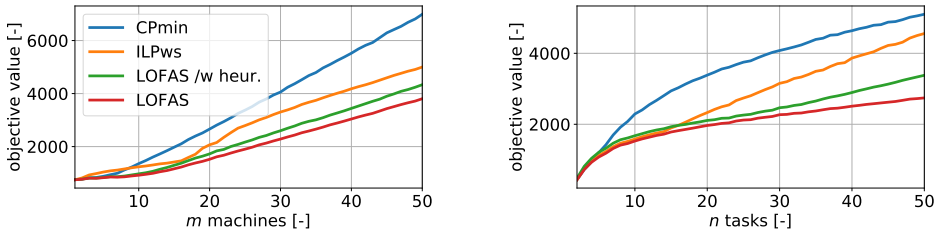
(a) Mean objective value for different number of machines m.

(b) Mean objective value for different number of tasks n.

Fig. 9: Comparison of exact models with warm starts.

Further, we compare the best objective value found by the heuristic algorithm LOFAS and LOFAS */w heur.* from Sect. 6 against CPmin and ILPws. The results are shown in Fig. 10. Note that we omit the results of CP3ws (CP3 model with warm starts) in Fig. 10 as the results were almost the same as those of LOFAS and the curves amalgamated.

To obtain better insight into the performance of the proposed methods, we compared the resulting distributions of achieved objectives from each method. We took results of each method for all instances and ordered them in a non-decreasing way with respect to achieved objective value and plotted them. The



(a) Mean objective value for different number of machines m.

(b) Mean objective value for different number of tasks n.

Fig. 10: Comparison of exact models and the heuristic algorithms.

results are displayed in Fig. 11. It can be seen that the proposed heuristics are able to find the same or better solutions in nearly all cases. Furthermore, we note that LOFAS */w heur.* outperformed both CPmin and ILPws as well. For the ILPws, one can notice a spike at around 65 % of instances. This is caused by the fact that for some instances, the ILP solver was not able to improve upon the initial warm start solution in the given time limit and these instances thus contribute to the distribution with higher objective values.

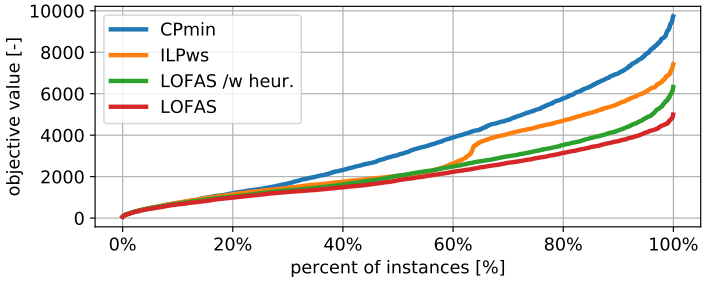


Fig. 11: Objective distributions of different methods.

A comparison of LOFAS to CP3ws on larger instances [17] showed the superiority of LOFAS. However, LOFAS still did not find any solution to the biggest instances (as reported in [17]) because the time limit was exceeded during the decomposition phase, i.e., during seeking an optimal sequence for a machine. This was the motivation for developing LOFAS */w heur.*

7.3 Comparison of Exact and Heuristic Subproblem in LOFAS

In this section, we compare solution quality produced by LOFAS heuristics with different methods for solving the machine subproblem, i.e., the method Seq(i, $TimeLimit$). The experiments were designed to asses if and how much different objective values are achieved when using the heuristic solution of the subproblem and if the heuristic variant scales better.

We have generated instances with $m \in \{5, 10, 15, 20\}$ machines and the number of tasks on each machine n ranging from 50 up to 1000. For each combination of m and n, we have generated 10 instances. The results are reported in Tab. 1. The column *objective* denotes the mean objective value if all instances were solved in the given time limit. Otherwise, we report the number of instances that were solved within the time limit. The results confirm the hypothesis that LOFAS gives better solutions regarding the objective, whereas LOFAS */w heur.* is able to find some solutions for larger instances when LOFAS does not manage to find any solution. More precisely, LOFAS scales only up to 300 tasks on 15 machines or 350 tasks on 5 machines. A heuristic solution of the subproblem in LOFAS */w heur.* allows obtaining a solution for instances of up to 1000 tasks on 5 machines. However, with the increasing number of machines, the CP solver struggles to produce any feasible solution to the whole problem, given the sequences on machines. Therefore, LOFAS */w heur.* did not find a solution for any instance with 20 machines, starting from 400 tasks.

To provide further details into the behavior of the algorithm, we report two other statistics. *Dead ends* shows how many times the algorithm hit an infeasible subproblem (due to the constraint on the objective) and restarted to the original solution, and *improv.* reports the number of iterations, where one iteration means avoiding the largest setup on the critical path and solving the subproblem. It can be clearly seen that these two numbers are significantly lower for LOFAS */w heur.* because it almost always wastes all the time allocated to a subproblem, whereas LOFAS is able to save some time on smaller instances, which is efficiently used for exploring more potential improvements. Note that for the instances that were not solved, these numbers are always zero.

Table 1: Comparison of exact and heuristic subproblem solvers in LOFAS heuristics.

		LOFAS with exact subproblem			LOFAS with heuristic subproblem		
m	n	objective [–]	dead ends [–]	improv. [–]	objective [–]	dead ends [–]	improv. [–]
5	50	1504.1 (±57.8)	64.6 (±2.4)	134.3 (±51.4)	1517.9 (±55.2)	0.0 (±0.0)	4.9 (±1.8)
10	50	1514.9 (±49.4)	91.4 (±43.4)	349.1 (±145.9)	1545.9 (±32.2)	0.0 (±0.0)	0.8 (±1.2)
15	50	1623.3 (±48.3)	0.0 (±0.0)	15.0 (±31.6)	1871.9 (±24.3)	0.0 (±0.0)	0.0 (±0.0)
20	50	2012.1 (±29.4)	0.0 (±0.0)	3.9 (±6.0)	2485.8 (±45.2)	0.0 (±0.0)	1.6 (±0.7)
5	100	2874.9 (±88.7)	72.2 (±46.2)	247.3 (±127.6)	2939.6 (±83.1)	0.0 (±0.0)	4.8 (±2.5)
10	100	2982.7 (±92.9)	67.9 (±52.0)	189.3 (±108.1)	3063.3 (±70.1)	0.0 (±0.0)	2.3 (±1.8)
15	100	2879.6 (±59.5)	17.9 (±17.1)	77.0 (±28.4)	3279.2 (±34.7)	0.0 (±0.0)	0.0 (±0.0)
20	100	2985.8 (±32.0)	0.0 (±0.0)	15.6 (±19.7)	4035.0 (±50.9)	0.0 (±0.0)	0.0 (±0.0)
5	150	4162.4 (±156.2)	66.5 (±36.1)	125.6 (±62.5)	4258.9 (±150.6)	0.0 (±0.0)	2.4 (±3.1)
10	150	4309.0 (±128.0)	25.5 (±16.8)	79.3 (±40.0)	4415.2 (±120.9)	0.0 (±0.0)	0.7 (±1.3)
15	150	4315.1 (±95.0)	6.4 (±7.2)	43.6 (±13.8)	4673.6 (±72.0)	0.0 (±0.0)	0.0 (±0.0)
20	150	4348.3 (±44.1)	5.6 (±7.3)	22.3 (±7.3)	5456.4 (±60.1)	0.0 (±0.0)	0.0 (±0.0)

5	200	5612.8 (±130.1)	39.6 (±22.8)	67.8 (±26.3)	5707.2 (±126.7)	0.0 (±0.0)	5.1 (±1.9)
10	200	5629.6 (±64.2)	5.6 (±10.2)	30.9 (±18.2)	5734.3 (±56.6)	0.0 (±0.0)	1.4 (±1.5)
15	200	5677.4 (±112.7)	7.6 (±5.4)	25.4 (±6.9)	6032.4 (±64.7)	0.0 (±0.0)	0.0 (±0.0)
20	200	5674.0 (±92.6)	3.4 (±3.6)	10.8 (±3.1)	7023.8 (±61.8)	0.0 (±0.0)	0.0 (±0.0)
5	250	6803.2 (±115.6)	27.6 (±12.4)	40.7 (±9.4)	6903.8 (±115.6)	0.0 (±0.0)	4.8 (±0.4)
10	250	6958.9 (±131.7)	5.8 (±7.2)	19.0 (±5.5)	7100.1 (±125.5)	0.0 (±0.0)	0.6 (±1.3)
15	250	7011.9 (±177.1)	1.3 (±1.5)	7.2 (±3.9)	7417.1 (±45.5)	0.0 (±0.0)	0.2 (±0.4)
20	250	6999.8 (±58.8)	0.6 (±0.8)	2.1 (±1.6)	8339.4 (±107.6)	0.0 (±0.0)	0.0 (±0.0)
5	300	8129.7 (±99.2)	15.2 (±9.1)	20.9 (±11.6)	8246.7 (±91.4)	0.0 (±0.0)	2.3 (±2.5)
10	300	8237.6 (±92.3)	4.7 (±4.1)	8.6 (±4.2)	8384.7 (±54.9)	0.0 (±0.0)	0.6 (±1.0)
15	300	8361.0 (±61.4)	2.4 (±2.4)	5.1 (±2.8)	8707.8 (±61.1)	0.0 (±0.0)	0.0 (±0.0)
20	300	5/10	0.3 (±0.5)	0.4 (±0.5)	9883.1 (±131.9)	0.0 (±0.0)	0.0 (±0.0)
5	350	9719.8 (±177.5)	14.6 (±5.5)	16.3 (±4.3)	9822.0 (±177.5)	0.0 (±0.0)	4.6 (±0.5)
10	350	6/10	4.2 (±4.1)	4.6 (±4.3)	9785.4 (±51.6)	0.0 (±0.0)	1.1 (±1.2)
15	350	2/10	0.1 (±0.3)	0.1 (±0.3)	10417.9 (±96.9)	0.0 (±0.0)	0.0 (±0.0)
20	350	0/10	0.0 (±0.0)	0.0 (±0.0)	2/10	0.0 (±0.0)	0.0 (±0.0)
5	400	8/10	5.0 (±3.4)	6.0 (±4.3)	11234.6 (±96.3)	0.0 (±0.0)	3.4 (±1.3)
10	400	0/10	0.0 (±0.0)	0.0 (±0.0)	11113.3 (±113.8)	0.0 (±0.0)	0.4 (±0.5)
15	400	0/10	0.0 (±0.0)	0.0 (±0.0)	11770.8 (±85.4)	0.0 (±0.0)	0.0 (±0.0)
20	400	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	450	8/10	3.2 (±4.7)	3.2 (±4.7)	12276.4 (±157.3)	0.0 (±0.0)	3.2 (±1.7)
10	450	0/10	0.0 (±0.0)	0.0 (±0.0)	12498.1 (±131.5)	0.0 (±0.0)	1.0 (±0.7)
15	450	0/10	0.0 (±0.0)	0.0 (±0.0)	13095.3 (±77.9)	0.0 (±0.0)	0.0 (±0.0)
20	450	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	500	3/10	0.8 (±1.3)	1.0 (±1.7)	13910.3 (±226.5)	0.0 (±0.0)	3.3 (±1.3)
10	500	0/10	0.0 (±0.0)	0.0 (±0.0)	13888.2 (±188.1)	0.0 (±0.0)	0.7 (±0.8)
15	500	0/10	0.0 (±0.0)	0.0 (±0.0)	7/10	0.0 (±0.0)	0.0 (±0.0)
20	500	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	550	0/10	0.0 (±0.0)	0.0 (±0.0)	15077.0 (±150.9)	0.2 (±0.4)	2.9 (±0.7)
10	550	0/10	0.0 (±0.0)	0.0 (±0.0)	15196.8 (±197.6)	0.0 (±0.0)	0.2 (±0.4)
15	550	0/10	0.0 (±0.0)	0.0 (±0.0)	1/10	0.0 (±0.0)	0.0 (±0.0)
20	550	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	600	0/10	0.0 (±0.0)	0.0 (±0.0)	16528.5 (±260.3)	0.2 (±0.4)	3.0 (±1.2)
10	600	0/10	0.0 (±0.0)	0.0 (±0.0)	16706.7 (±327.2)	0.0 (±0.0)	0.2 (±0.4)
15	600	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	600	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	650	0/10	0.0 (±0.0)	0.0 (±0.0)	17676.3 (±172.8)	0.2 (±0.4)	1.9 (±1.7)
10	650	0/10	0.0 (±0.0)	0.0 (±0.0)	18235.4 (±173.0)	0.0 (±0.0)	0.2 (±0.4)
15	650	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	650	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	700	0/10	0.0 (±0.0)	0.0 (±0.0)	19114.6 (±317.8)	0.4 (±0.8)	3.2 (±0.4)
10	700	0/10	0.0 (±0.0)	0.0 (±0.0)	19532.0 (±198.8)	0.0 (±0.0)	0.0 (±0.0)
15	700	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	700	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	750	0/10	0.0 (±0.0)	0.0 (±0.0)	20477.8 (±296.6)	0.1 (±0.3)	2.9 (±0.6)
10	750	0/10	0.0 (±0.0)	0.0 (±0.0)	20774.5 (±123.7)	0.0 (±0.0)	0.2 (±0.4)
15	750	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	750	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	800	0/10	0.0 (±0.0)	0.0 (±0.0)	21762.7 (±179.1)	1.2 (±1.2)	2.8 (±2.1)
10	800	0/10	0.0 (±0.0)	0.0 (±0.0)	8/10	0.1 (±0.3)	0.5 (±0.7)
15	800	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	800	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	850	0/10	0.0 (±0.0)	0.0 (±0.0)	23144.0 (±210.5)	1.5 (±1.8)	2.8 (±2.6)
10	850	0/10	0.0 (±0.0)	0.0 (±0.0)	9/10	0.0 (±0.0)	0.0 (±0.0)
15	850	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	850	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	900	0/10	0.0 (±0.0)	0.0 (±0.0)	24555.4 (±316.8)	1.1 (±1.5)	2.7 (±1.8)
10	900	0/10	0.0 (±0.0)	0.0 (±0.0)	1/10	0.0 (±0.0)	0.0 (±0.0)
15	900	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	900	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)

5	950	0/10	0.0 (±0.0)	0.0 (±0.0)	25701.5 (±188.7)	0.5 (±0.7)	2.2 (±1.1)
10	950	0/10	0.0 (±0.0)	0.0 (±0.0)	3/10	0.0 (±0.0)	0.0 (±0.0)
15	950	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	950	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	1000	0/10	0.0 (±0.0)	0.0 (±0.0)	27054.0 (±275.9)	1.7 (±1.6)	3.3 (±1.9)
10	1000	0/10	0.0 (±0.0)	0.0 (±0.0)	1/10	0.0 (±0.0)	0.0 (±0.0)
15	1000	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	1000	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)

7.4 Discussion

We have seen that performances of CP models are almost equal with CP3 being the best but its advantage is almost negligible. Further, the experiments have shown that ILP without a warm start cannot find a feasible solution for instances with $n \geq 8$ tasks reliably, whereas with warm starts it was significantly better than the best CP model without a warm start. The quality of the solutions from CP with warm starts is much better than ILP with warm starts (even though the warm start for CP is not a complete solution), as can be seen in Fig. 9. As expected, the heuristic algorithm LOFAS produced the best solutions among all compared methods, although only slightly better than CP3 model with warm starts. Smaller instances evidenced that LOFAS achieves objective values quite close to optimal ones. The advantage of LOFAS */w heur.* can be seen in its scalability capabilities as it can solve instances with up to 1000 tasks on 5 machines. On the 50×49 instance set from Sect. 7.1, LOFAS */w heur.* rendered solutions of objective value worse on average by 10.5 % than LOFAS.

8 Conclusions

This paper tackled the problem of scheduling sequence-dependent non-overlapping setups on dedicated machines. An ILP model, five CP models, and a heuristic approach were proposed. The results showed that all exact methods are finding solutions far from optima within the given time limit, whereas the proposed heuristic algorithm finds high-quality solutions in very short computation time.

The main contributions of this paper with respect to [17] are new CP models using the cumulative function and a new complexity result for the restricted version of the problem. Furthermore, we have proposed an enhancement to the LOFAS algorithm proposing a heuristic for the subproblem, which allows solving larger instances, and extensive experimental evaluation that showed the effect of the subproblem solution method on the scalability and solution quality.

For future work, we will consider a more complex problem, which will avoid the limitation that the tasks are already assigned to machines. Also, instead of non-overlapping setups for one machine setter, we will consider more machine setters that will be treated as a resource with limited capacity.

Acknowledgements

We would like to thank Philippe Laborie for his help with the design of CP4 model.

References

1. Google's or-tools, https://developers.google.com/optimization/, accessed May 22, 2019
2. Allahverdi, A., Ng, C., Cheng, T.E., Kovalyov, M.Y.: A survey of scheduling problems with setup times or costs. European journal of operational research **187**(3), 985–1032 (2008)
3. Applegate, D., Cook, W.: A computational study of the job-shop scheduling problem. ORSA Journal on computing **3**(2), 149–156 (1991)
4. Balas, E.: Project scheduling with resource constraints. Tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group (1968)
5. Chen, D., Luh, P.B., Thakur, L.S., Moreno Jr, J.: Optimization-based manufacturing scheduling with multiple resources, setup requirements, and transfer lots. IIE Transactions **35**(10), 973–985 (2003)
6. Gurobi: Constraints. http://www.gurobi.com/documentation/8.1/refman/constraints.html (2019), accessed June 12, 2019
7. Hentenryck, P.V., Michel, L.: Constraint-based local search. The MIT press (2009)
8. Laborie, P., Rogerie, J., Shaw, P., Vilim, P.: Reasoning with conditional time-intervals. part ii: An algebraical model for resources. In: FLAIRS conference. pp. 201–206 (2009)
9. Laborie, P., Rogerie, J., Shaw, P., Vilim, P.: IBM ILOG CP optimizer for scheduling. Constraints **23**(2), 210–250 (2018)
10. Lasserre, J.B., Queyranne, M.: Generic scheduling polyhedra and a new mixed-integer formulation for single-machine scheduling. Proceedings of the 2nd IPCO (Integer Programming and Combinatorial Optimization) conference pp. 136–149 (1992)
11. Lee, Y.H., Pinedo, M.: Scheduling jobs on parallel machines with sequence-dependent setup times. European Journal of Operational Research **100**(3), 464–474 (1997)
12. Pisinger, D., Ropke, S.: Large neighborhood search. In: Handbook of metaheuristics, pp. 399–419. Springer (2010)
13. Ruiz, R., Andres-Romano, C.: Scheduling unrelated parallel machines with resource-assignable sequence-dependent setup times. The International Journal of Advanced Manufacturing Technology **57**(5-8), 777–794 (2011)
14. Tempelmeier, H., Buschkuhl, L.: Dynamic multi-machine lotsizing and sequencing with simultaneous scheduling of a common setup resource. International Journal of Production Economics **113**(1), 401–412 (2008)
15. Vallada, E., Ruiz, R.: A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. European Journal of Operational Research **211**(3), 612–622 (2011)
16. Vilim, P., Bartak, R., Cepek, O.: Extension of $O(n \log n)$ filtering algorithms for the unary resource constraint to optional activities. Constraints **10**(4), 403–425 (2005). https://doi.org/10.1007/s10601-005-2814-0, https://doi.org/10.1007/s10601-005-2814-0
17. Vlk., M., Novak., A., Hanzalek., Z.: Makespan minimization with sequence-dependent non-overlapping setups. In: Proceedings of the 8th International Conference on Operations Research and Enterprise Systems - Volume 1: ICORES,. pp. 91–101. INSTICC, SciTePress (2019). https://doi.org/10.5220/0007362700910101
18. Wikum, E.D., Llewellyn, D.C., Nemhauser, G.L.: One-machine generalized precedence constrained scheduling problems. Operations Research Letters

16(2), 87 – 99 (1994), http://www.sciencedirect.com/science/article/pii/0167637794900647
19. Zhao, X., Luh, P.B., Wang, J.: Surrogate gradient algorithm for lagrangian relaxation. Journal of optimization Theory and Applications **100**(3), 699–712 (1999)