Highlights

Constraint Programming and Constructive Heuristics for Parallel Machine Scheduling with Sequence-Dependent Setups and Common Servers

Vilém Heinz, Antonín Novák, Marek Vlk, Zdeněk Hanzálek

- Effective Constraint Programming model is proposed.
- Constructive heuristics with multiple improvement methods are presented.
- An effective approach composed of Constraint Programming and heuristics is proposed.
- Improvement over the state-of-the-art exact and heuristic approaches is achieved.

Constraint Programming and Constructive Heuristics for Parallel Machine Scheduling with Sequence-Dependent Setups and Common Servers

Vilém Heinz[a,b,*], Antonín Novák[a,b], Marek Vlk[b], Zdeněk Hanzálek[b]

[a] *Faculty of Electrical Engineering, Czech Technical University in Prague, CZ*
[b] *Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, CZ*

Abstract

This paper examines scheduling problem denoted as $P|seq, ser|C_{max}$ in Graham's notation; in other words, scheduling of tasks on parallel identical machines (P) with sequence-dependent setups (seq) each performed by one of the available servers (ser). The goal is to minimize the makespan (C_{max}). We propose a Constraint Programming (CP) model for finding the optimal solution and constructive heuristics suitable for large problem instances. These heuristics are also used to provide a feasible starting solution to the proposed CP model, significantly improving its efficiency. This combined approach constructs solutions for benchmark instances of up to 20 machines and 500 tasks in 10 seconds, with makespans 3 % to 11.5 % greater than the calculated lower bounds with a 5% average. The extensive experimental comparison also shows that our proposed approaches outperform the existing ones.

Keywords: Scheduling, Parallel machines, Sequence-Dependent setups, Servers, Constraint Programming, Heuristic

2010 MSC: 68M20, 90-08, 90B35, 90C59, 90C99

1. Introduction

In recent years, manufacturing has become more complex than ever. Factories must handle a large variety of products on multiple production lines, customers demand highly

*Corresponding author
Email address: vilem.heinz@cvut.cz (Vilém Heinz)

proprietary products of small-batch volumes, and companies are constantly forced to adapt to market shifts. We are in an environment where the organization of work and effective utilization of production capacities is a crucial yet very complex problem to tackle [1].

In this paper, we focus on a scenario often encountered in real-world production. Imagine a production environment with multiple identical machines. Tasks are to be freely assigned to these machines. Depending on their assignment and order, adjustments on said machines (sequence-dependent setups) are performed between them by servers. The number of servers can be arbitrary, and any server can perform any setup.

Example. To illustrate the importance of the considered problem, we provide an example of a production planning challenge tackled by an existing company. This company produces plastic tubes of different shapes and sizes but from the same material and with a very similar manufacturing process. Thus, all machines they use are the same and depending only on their settings, they produce plastic tubes of different shapes and/or sizes. These machines' settings are adjusted by workers present in the factory. Since every tube shape and size requires different machine settings to be produced, adjustment between each tube pair in production has its own unique sequence and can require variable time to adjust. Clearly, there is no need to adjust the machine between the production of tubes of the same shape and size.

In general, this scheduling problem containing servers, parallel identical machines and sequence-dependent setups is encountered at paint shops, printing houses, custom component manufacturing and 3D printing, foods, chemical, and other production industries where the machines are used for multiple types of different products with the necessity of adjustments. We refer the reader to *Huang et al.* [2], *Kim et al.* [3] and *Hamzadayi et al.* [4] for similar problem examples.

Needless to say, human resource capacity in such productions is sometimes disregarded during the scheduling phase because of its complex real-world restrictions and uncertainties, leaving decisions to expert knowledge. We argue that even in such cases, a solution with human resource capacity taken into account can help to guide this decision process and improve the overall solution.

1.1. Approach

We propose an efficient Constraint Programming (CP) [5] model as an exact approach to the considered problem. As demonstrated in *Laborie et al.* [6], CP is more effective than conventionally used Integer Linear Programming (ILP) for many scheduling problems. However, since the problem is $\mathcal{NP}$-hard, the CP model's execution is still exponential in the worst case. Thus, we propose constructive heuristic algorithms that provide a solution in polynomial time. We also discuss several applicable improvements, yielding better solutions in exchange for more prolonged but still polynomial executions. Ultimately, we combine both approaches by using a heuristic solution as a starting point of the CP model, leveraging their distinct strengths. This combination results in significant performance improvement of the CP model, making much larger instances solvable. In general, this method of providing a starting solution to the main algorithm is called warm starting, and it was successfully applied to various scheduling problems, such as in *de Abreu et al.* [7] or *Pour et al.* [8].

1.2. Contribution and Paper Outline

We summarize the key contributions of the paper as:

1. Addressing a new scheduling problem with servers, arising as a natural generalization of particular cases *Huang et al.* [2], *Kim et al.* [3] and *Hamzadayi et al.* [4] which lay important groundwork studying aspects of our considered problem. This results in a problem combining sequence-dependent setup times and machine-task independence with extension to multiple servers availability.
2. Proposing CP model utilizing cumulative resource function that can solve instances of up to ten machines and tens of tasks to the optimality usually under an hour. With a warm start, the CP model can be very effectively used on instances of tens of machines and hundreds of tasks in just several-minute runtimes.
3. Proposing domain-specific heuristics that provide feasible solutions to instances of hundreds of machines and tens of thousands of tasks between several seconds and several minutes of runtime. With the optional improvements applied, solutions have a lower bound gap of 3.5 % to 20 % with an 8% average, measured on the benchmark instance set.

4. Providing more efficient approaches than the state-of-the-art for related problems proposed in papers *Huang et al.* [2], *Kim et al.* [3] and *Hamzadayi et al.* [4]. Both CP model as an alternative to ILP models and warm started CP model, as an alternative to state-of-the-art heuristics, show improvements in efficiency over their respective counterparts as they are able to solve instances of said related problems. This finding is very important as it proves that our approaches are not just addressing a new problem but also doing it efficiently compared to related state-of-the-art approaches. For our considered problem benchmark instances of sizes up to 20 machines and 500 tasks, in 10 seconds of computation, the warm started CP model provides solutions with makespans 3 % to 11.5 % greater than the calculated lower bounds with a 5% average.

The rest of the paper is organized as follows: Section 2 formally describes the problem and its complexity. Section 3 lists relevant literature on similar problems and discusses the current state-of-the-art approaches. Section 4 describes the proposed CP model. Section 5 describes developed constructive heuristics, proposes ways of improving their solutions in a trade-off with computational time and describes constructive heuristics warm start application to the CP model. Section 6 contains the experimental results, discusses differences between proposed approaches and provides a comparison to the current state-of-the-art. Section 7 sums up the findings discovered in this paper.

2. Problem Statement

The considered problem is denoted $P|seq, ser|C_{max}$ in Graham's notation. P denotes parallel identical machines task scheduling, seq denotes sequence-dependent setups, ser denotes presence of servers performing said setups and C_{max} denotes the overall goal of minimizing the makespan. We define problem input parameters in the following way:

- Let $M = \{M_1, \ldots, M_m\}$ be a set of identical machines where every machine can execute at most one task or one setup at a time. The number of machines m is given by the problem instance.
- Let $R = \{R_1, \ldots, R_r\}$ be a set of available servers. Servers are considered to be identical. The number of servers r is given by the problem instance and we generally

expect it to be strictly lower than m.

- Let $T = \{T_1, \ldots, T_t\}$ be a set of independent non preemptive tasks where each task can be executed on any machine. Every task is described by its processing time given by the problem instance, denoted $p_i \in \mathbb{N}$.
- Let $\boldsymbol{O} = [o_{i,j}] \in (\mathbb{N} \cup \{\infty\})^{t \times t}$ be the setup times matrix given by the problem instance, where $o_{i,j}$ denotes the setup length between tasks T_i and T_j.

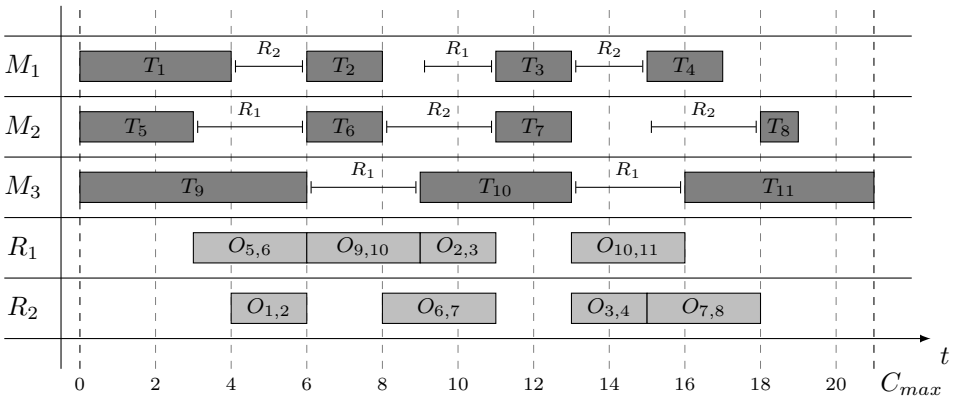

Figure 1: A Gantt chart of identical parallel machine production with sequence-dependent setups performed by servers.

The problem solution is defined by task assignments to machines and the order of tasks on each machine. For every task T_i in the problem solution, let $s_i \in \mathbb{N}_0$ be start time and $c_i \in \mathbb{N}$ completion time of that task in the resulting schedule. Then, let us define set of present setups in the schedule as $U = \{U_1, \ldots, U_u\}$. Let $\overline{s}_y \in \mathbb{N}$ be the start, $\overline{p}_y \in \mathbb{N}$ the processing time and $\overline{c}_y \in \mathbb{N}$ the completion time of the setup U_y in the resulting schedule.

We consider the solution feasible if the following conditions hold:

(A1) Every task must be executed without preemption, i.e., it cannot be temporarily suspended during the execution, so $s_i + p_i = c_i$ must hold.

(A2) If task T_j is next after T_i in the machine sequence, setup $o_{i,j}$ must be performed by a server and s_j - $c_i \geq o_{i,j}$ must hold. This also means that no setup must be executed before the first task on each machine.

(A3) Setup must be performed by exactly one server.

(A4) Setups are non-preemptive, $\overline{s}_y + \overline{p}_y = \overline{c}_y$ must hold for any $U_y \in U$.

(A5) If U_y is the setup between tasks T_i and T_j then $c_i \leq \overline{s}_y$ and $\overline{c}_y \leq s_j$ must hold.

(A6) For any two setups U_y, U_z performed by the same R_x, $\overline{s}_z \geq \overline{c}_y$ or $\overline{s}_y \geq \overline{c}_z$ must hold. This also ensures that server can perform at most one setup at a time.

Finally, the makespan minimization can be thought of as the minimization of the latest task completion time.

Fig. 1 shows an example of feasible schedule of the considered problem. In this case, we have 3 machines (M_1 to M_3), 2 servers (R_1 and R_2) and 11 tasks (T_1 to T_{11}). Each row in Fig. 1 represents the schedule of one problem's resource, either a machine or a server. Setups on machines are marked by server performing them. The zero represents the start and C_{max} the end of the schedule, i.e. makespan. Task processing times and their respective setups in the schedule are in Table 1. The makespan is 21.

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
4	2	2	2	3	2	2	1	6	4	5

$O_{5,6}$	$O_{9,10}$	$O_{2,3}$	$O_{10,11}$	$O_{1,2}$	$O_{6,7}$	$O_{3,4}$	$O_{7,8}$
3	3	2	3	2	3	2	3

Table 1: Processing and setup times of the tasks in Fig. 1.

2.1. Complexity of the Problem

The problem addressed in this paper is a more general variant of the $\mathcal{NP}$-hard problem $PD|seq, ser = 1|C_{max}$ tackled in *Vlk et al.* [9], where only one server is considered and every task is dedicated to only one specific machine.

We can reduce instance of $PD|seq, ser = 1|C_{max}$ to instance of $P|seq, ser|C_{max}$ by setting the setup times between all pairs of tasks dedicated to different machines in $PD|seq, ser = 1|C_{max}$ instance to be infinity in $P|seq, ser|C_{max}$ instance. This makes the solution, where two tasks originally assigned to different machines are placed on the same machine infeasible. Since the availability of only one server is implicitly supported by our problem definition, the reduction is complete, showing that $P|seq, ser|C_{max}$ is $\mathcal{NP}$-hard.

3. Related Work

The existing related problems can be classified into three categories; first containing papers focused on sequence-dependent problems, second containing papers dedicated to problems with servers and third containing papers combining both properties. We discuss each in its section, while in the last one we also consider relations of existing problems to ours.

3.1. Problems with Sequence-Dependent Setup Times

Allahverdi et al. [10] survey showed that many papers tackling sequence-dependent setup scheduling exist. Papers like *Vallada et al.* [11] and *Lee et al.* [12] focused on heuristics, handling instances of up to lower hundreds of tasks in a few minutes of computational time. More recent papers usually focus on combination with other interesting properties.

Lunardi et al. [13] focused on sequence-dependent scheduling with job decomposition, precedence constraints, preemption, unavailability periods and partial overlaps. Their metaheuristic was compared to CP proposed in *Lunardi et al.* [14]. Comparison indicated that CP alone could not handle some large instances and improved slower, but it guarantees improvement until an optimal solution is found. This suggests that using a faster method to warm start the CP model, which further improves the solution, might be a promising way to tackle similar problems.

Rauchecker et al. [15] tackled the problem of unrelated parallel machines with sequence-dependent setup times focusing on high-performance computing and parallelization. It is shown that by applying their branch-and-price algorithm in a parallel manner, the computations can be reduced from hours to just minutes of computation.

3.2. Problems with Servers

Amir et al. [16] proposed an ILP formulation for a single server, two parallel identical machines scheduling problem. They found that their model could not compute instances larger than 12 tasks. However, they show that when specific conditions are met, there exist fast approaches applicable in such constrained scenarios. A more effective block model ILP formulation for the two parallel identical machines problem was formulated

by *Hasani et al.* [17]. This model provides an optimal solution to certain instances of up to 250 tasks in 3600 seconds. Alternatively the problem can be modeled by timed automata [18], but their performance is much lower.

A scheduling and lot sizing problem with a common setup operator (server) is studied in *Tempelmeier et al.* [19]. The setups performed by the operator are considered to be scheduled without overlaps. The setups are associated only with the following task, making them sequence-independent. The proposed ILP formulations can solve instances of tens of tasks, usually under a 1-minute time limit.

A problem involving setups performed by operators of different capabilities has been studied in *Chen et al.* [20]. The problem is modelled using time-indexed formulation and solved by decomposition into smaller subproblems using Lagrangian relaxation. Subproblems are solved using Dynamic Programming (DP), and feasible solution is obtained by the composition of the subproblems. If that is impossible, the Lagrangian multipliers are updated using the surrogate subgradient method as in *Zhao et al.* [21]. The downside is that the time-indexed formulation yields a model with pseudo-polynomial size, which is unsuitable if large processing and setup times are present. The proposed approach can handle instances of tens of machines in a matter of minutes.

3.3. Problems with Sequence-Dependency and Servers

Papers *Vlk et al.* [22] and *Vlk et al.* [9] study problem with sequence-dependent setups. However, tasks are dedicated to machines, and a single server is assumed. CP models, an ILP model, and heuristics utilizing the problem's decomposition are proposed in both papers. The resulting subproblems deal with task ordering on machines independently. The results show that proposed CP models can find a solution for instances of tens of machines and tens of tasks for each machine in 1 minute. The proposed LOFAS algorithm with a heuristic can solve instances of up to 1000 tasks on 5 machines. The same problem as in *Vlk et al.* [9] is studied in *Huang et al.* [2]. An ILP model and a Genetic Algorithm (GA) are proposed. It is stated that ILP formulation is unusable in a real-world scenario. The GA approach can solve instances of 10 machines and 100 tasks in under 50 seconds.

The problem in *Kim et al.* [3] allows assignment of tasks to any machine. However, since setups are sequence-independent, there is no direct comparison in terms of generality to the aforementioned papers. Again, only one server is allowed. ILP formulations

and hybrid heuristics are proposed. The results show the ILP model can solve instances of 6 machines and up to 40 tasks, providing optimal or close to the optimal solution in 3600s. The hybrid heuristic solves the same instances under 1 minute, with resulting schedules 2 % to 5 % longer than schedules obtained by ILP in 3600s.

Paper *Hamzadayi et al.* [4] generalizes *Vlk et al.* [9], *Huang et al.* [2] and *Kim et al.* [3], tackling machine-independent with sequence-dependent setups. However, it still considers only one server, limiting its real-world applicability. ILP model is proposed, while the best performing proposed heuristic is GA. The results show that the ILP model can find the optimal solution in 18000 seconds for instances of 3 machines and 9 tasks. The GA can solve instances of 10 machines and 100 tasks in one minute.

Papers tackling more general problems than our considered $P|seq, ser|C_{max}$ also exist. In *Costa et al.* [23], a hybrid GA to a problem extended by server's skill (server's setup execution speed) and parallel unrelated machines is proposed. While the proposed GA can be applied to our problem, their experimental results show that its scalability is lower than the scalability of our proposed approach. Their proposed ILP model has no detailed results provided.

In more recent work by *Yeppes-Borrero et al.* [24], unrelated machines and setups requiring multiple servers are considered. Provided experimental results show that their proposed ILP model does not scale too well. While their proposed heuristic scales well, their results indicate it does not scale as well as our proposed approach.

Luis Fanjul-Peyro [25] considers unrelated machines and multiple types of resources, i.e., servers. ILP model and exact approaches are proposed, but no heuristic approach is provided, so the scalability of the provided approaches is poor.

Other papers tackling more general problems like *Lee et al.* [26], *Caricato et al.* [27], *Yeppes-Borrero et al.* [28] or papers considering servers and sequence-dependency in different settings like *Gnatowski et al.* [29] exist, but their respective problem definitions are even more distant to ours than the problem definitions of the aforementioned papers.

In conclusion, research shows that while papers addressing less or more general problems exist, to the best of our knowledge, no paper tackles exactly our considered problem, which represents many real-world production scenarios. It is apparent from this section that as problem complexity increases, the ability to solve larger instances de-

creases rapidly, especially for exact approaches. While more general problem approaches do not provide the same efficiency and scalability as ours, our proposed approaches offer improved efficiency over existing approaches for less general problems. Furthermore, as seen in this section, the majority of works also employ ILP modelling as an exact approach. However, results from *Vlk et al.* [9] show that CP is far more suitable than ILP for this kind of scheduling problem. This contrasts with the current state of literature and is a good reason to investigate CP capability further.

4. Constraint Programming Approach

The first approach we use to solve the considered $P|seq, ser|C_{max}$ problem is Constraint Programming (CP). Specifically, we use *IBM ILOG CP Optimizer* CP solver. The following few paragraphs describe basic CP concepts used. A reader familiar with CP formalism can skip to Section 4.1, where model specifics are explained.

The main modelling expression used is *interval variable*. As the name suggests, it represents some activity with its required reserved time in the schedule. Its length is denoted by LengthOf, start time by StartOf and completion time by EndOf. While LengthOf is given by the problem instance, StartOf and EndOf are determined by the CP solver. We use the interval variables to represent tasks and setups in our problem.

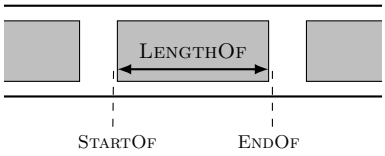

Figure 2: Interval variable in the schedule.

Every interval variable can be set *optional*. Depending on the model's conditions and optimization criteria, the optional interval variable is *present* or *absent* in the solution. Unlike the present variable, the absent variable does not have to conform to the model's constraints. We can also specifically enforce the presence/absence of variables by introducing constraints implying either. One example is global constraint Alternative which makes subset (usually of size one) of variables of given set present and rest absent while respecting other conditions in the model.

To model the relations we use *constraints*. In addition to the linear algebraic constraints known from the ILP modelling, CP offers more complex ones such as logic constraints, scheduling constraints, global constraints and others. Global constraints are especially useful, enabling us to model specific predefined relations over a set of variables in a concise and computationally optimized way. One such constraint is the aforementioned ALTERNATIVE. This is a big reason why CP is so effective for some types of problems.

4.1. Model Formulation

Since $PD|seq, ser = 1|C_{max}$ problem considered in *Vlk et al.* [9] is a more constrained variant of our problem, we utilize some of the ideas proposed in it. There are two key ideas behind the proposed model. First, since servers are identical, we do not have to represent which specific server is performing which setup. So rather than considering particular servers, we consider only how many servers we need at a particular time to perform the setups currently scheduled. Second, we do not represent every possible setup between a pair of tasks by a variable. Thus, we avoid the quadratic number of setup variables and related constraints; the number will equal the number of tasks. This is achieved by representing every possible setup following some particular task by one interval variable only.

When constructing the model, we must first ensure that every task is executed in the produced schedule. We denote the set of interval variables representing tasks as I^T and the interval variable of specific task T_i as I_i^T. We use inequalities instead of equalities when assigning the processing times to variables in I^T so the interval variable representing task can prolong itself until a server is available to perform the following setup. This will be useful later. Still, at least time equal to the task's processing time must be reserved in the schedule so that it can be executed. We assign the relaxed processing length to the interval variables as follows:

$$\forall I_i^T \in I^T : \textsc{LengthOf}(I_i^T) \geq p_i. \tag{CP-1}$$

Since we do not know which task will be assigned to which machine, we use the ALTERNATIVE constraint. For every $I_i^T \in I^T$ and for every machine $M_k \in M$ we add

a new optional interval variable $I_{i,k}^{T^{Opt}}$. Then for each task, all its optional variables are passed to the ALTERNATIVE constraint, ensuring that only one will be present. This way, every task will be present on exactly one machine. We also associate selected present $I_{i,k}^{T^{Opt}}$ with the interval variable I_i^T representing our task in other model's conditions by passing I_i^T as an argument to ALTERNATIVE constraint. The resulting constraints are following:

$$\forall I_i^T \in I^T : \textsc{Alternative}\left(I_i^T, \bigcup_{M_k \in M} \left\{I_{i,k}^{T^{Opt}}\right\}\right). \tag{CP-2}$$

To ensure that tasks do not overlap on any particular machine, for every machine, we add NOOVERLAP constraint containing optional interval variables representing tasks scheduled on that machine. NOOVERLAP constraint ensures that no two interval variables from a given set overlap in the produced schedule. Note that formally, interval variables passed to NOOVERLAP constraint must be wrapped in sequence variable whose explanation we omit as it has no semantic significance in this case. NOOVERLAP constraint can also be given *transition matrix*, which defines empty interval sizes between the end of one and the start of the following interval variable in the given set. We use our setup times matrix $\boldsymbol{O}$ as the *transition matrix* of the NOOVERLAP constraint to insert setup times between tasks in the final machine schedules. The resulting formulation is following:

$$\forall M_k \in M : \textsc{NoOverlap}\left(\bigcup_{I_i^T \in I^T} \{I_{i,k}^{T^{Opt}}\}, \boldsymbol{O}\right). \tag{CP-3}$$

We define a set I^S of interval variables, where I_i^S represents setup following its respective I_i^T from I^T. Notice that while setup times are already present between tasks thanks to Eq. (CP-3), we need these variables for server constraints. Since we do not need a setup after the last task on each machine, it will be a dummy setup of zero length. Due to that, we merely set the length of the setup variables to be at least zero, and the actual length of the setup executed will be determined later:

$$\forall I_i^S \in I^S : \textsc{LengthOf}(I_i^S) \geq 0. \tag{CP-4}$$

Next, we synchronize the start and completion times between tasks and setups. We

use constraint $\textsc{EndAtStart}(I_A, I_B)$, ensuring that the interval variable I_A is completed exactly when the interval variable I_B starts. Using it, we can set the end of every task equal to the start of its following setup since the task's length is assigned by inequality and can be prolonged. The resulting constraints are following:

$$\forall I_i^T \in I^T : \textsc{EndAtStart}(I_i^T, I_i^S). \tag{CP-5}$$

Now, we constrain ending times of all setups. We achieve this by using $\textsc{StartOfNext}(I')$ constraint, which returns the start time of the next interval variable following I' in the sequence. Thus, we can get the start time of the optional interval variable representing the next task after the given one in the machine sequence. Then, we set the end of the setup bigger or equal to the following task's start time. Using equality would not suffice because StartOfNext evaluates to 0 for absent $I_{i,k}^{T^{Opt}}$ and the last task on every machine. Note that even though the setup could theoretically be prolonged and end after the following task's start because of the inequality, it does not affect the feasibility of task placement and can be fixed by shortening such setups to the correct lengths after the solution is found. Thus, the final constraints are as follows:

$$\forall I_i^S \in I^S, \forall M_k \in M : \textsc{EndOf}(I_i^S) \geq \textsc{StartOfNext}(I_{i,k}^{T^{Opt}}). \tag{CP-6}$$

Finally, we set the maximum number of concurrently executed setups less or equal to the servers available. We achieve this by using the expression $\textsc{Pulse}(I', a)$. This expression specifies that a units of resource, in our case servers, is used during interval I'. The cumulative function is composed of Pulse terms for each I_i^S representing the use of precisely one server. At any point in the schedule, the function is upper-bounded by the number of servers r provided:

$$\sum_{I_i^S \in I^S} \textsc{Pulse}(I_i^S, 1) \leq r. \tag{CP-7}$$

In the end, we minimize the makespan by minimizing the end of the last task:

$$\textsc{Max}\left(\bigcup_{I_i^T \in I^T} \{\textsc{EndOf}(I_i^T)\}\right). \qquad \text{(CP-C}_{max}\text{)}$$

Thus, the complete CP model further denoted as $\mathtt{CP_0}$ is defined as follows:

$$\begin{aligned} &\textsc{Minimize} \quad \text{(CP-C}_{max}\text{)} \\ &\quad \text{s.t.} \\ &\qquad \text{(CP-1)} - \text{(CP-7)} \end{aligned}$$

An alternative CP model formulation where every setup was represented by its own interval variable was considered, but it was omitted from the paper due to having noticeably worse performance.

5. Constructive Heuristic Algorithms

Since the considered $P|seq, ser|C_{max}$ problem is $\mathcal{NP}$-hard, using (only) an exact approach is computationally intractable for many larger problem instances. Therefore, it is essential to develop heuristic methods that produce a feasible solution of reasonable quality in an affordable time. In this section, two constructive heuristic algorithms are proposed:

- *Locally Optimal Selection of Setups* (LOSOS),
- *Resolution of Setup Overlaps Lazily* (ROSOL).

The main difference between the two is that LOSOS handles server allocation during task scheduling while ROSOL handles it in a separate phase after task sequences on machines are set. Three external functions called GenerateStartingTasks, SelectNextTask and OptimizeScheduleEnds are used by both algorithms:

- GenerateStartingTasks selects a starting (first) task for every machine. The baseline version of this method selects tasks randomly.
- SelectNextTask provides a suitable task to follow after the current last one scheduled on a machine. The baseline version of this method selects the following

task greedily by picking yet unscheduled task with the shortest setup time from the currently ending task to itself.

- OptimizeScheduleEnds re-optimizes the ending parts of machines' schedules after all tasks are assigned. This is important since the end of the machine schedule is the least compact and optimized part. The baseline version of this method only considers shortening the longest machine by moving the task to another machine.

Further improvements to these functions are proposed in Section 5.3. For now, we consider them as black-box functions providing us with the described output.

To allow easy reproduction of the algorithms, described pseudocodes for both algorithms are provided in Appendix A.

5.1. Locally Optimal Selection of Setups (LOSOS)

The idea of LOSOS is to iteratively construct chains of tasks on machines, always selecting the next task greedily according to some given evaluation function while at the same time considering server availability.

The algorithm steps are following. First, GenerateStartingTasks is called to set starting tasks to machines. Then, a queue of servers is created, keeping track of times when servers are ready to work. Now, while there are tasks to schedule, the following is repeated. Machine with the currently shortest schedule is found. The task to follow the currently ending task is selected by SelectNextTask. Then, the soonest available server is assigned to perform the setup. After all, tasks are assigned to machines, OptimizeScheduleEnds is called to re-optimize the schedule.

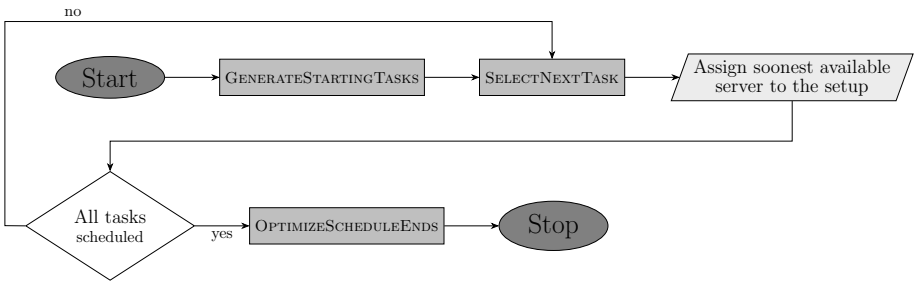

Figure 3: Flowchart of LOSOS algorithm.

The algorithm is visualized in Fig. 3. It has polynomial time complexity $\mathcal{O}((\log(r) + \log(m) + t) \cdot t + m)$ if priority queues are used for servers. Baseline time complexities of GenerateStartingTasks and OptimizeScheduleEnds are considered to be $\mathcal{O}(m)$ and SelectNextTask baseline time complexity to be $\mathcal{O}(t)$.

5.2. Resolution of Setup Overlaps Lazily (ROSOL)

The problem of LOSOS is that it does not consider upcoming server requirements when scheduling tasks and setups. To solve this, ROSOL first ignores server constraints when scheduling. Then, in the additional phase, further logic is applied to resolve these constraints violations in a more informed way.

The algorithm steps are following. First, GenerateStartingTasks is called to set starting tasks to machines. Then, while there are unscheduled tasks, we repeatedly find the machine with the currently shortest schedule and assign task selected by SelectNextTask to it. When all tasks are scheduled, the problem is solved without servers' constraints, and OptimizeScheduleEnds is called to balance the schedule and reduce possible unnecessary conflicts arising when assigning the servers.

We ensure that servers' constraints hold in the following way. We move from schedule start to its end, checking if the number of concurrently performed setups k does not exceed the number of servers r. If it does, we need to postpone (move to the later time in the schedule) $k - r$ setups until at least one server performing non-postponed setup is free. The non-postponed setups are executed fully without preemption, meaning that they cannot be postponed once their execution started. We repeat this until we reach the end of the schedule. Finally, we call OptimizeScheduleEnds again.

The key remaining question is how to choose which setups to postpone. This is decided based on the *tolerance coefficient* of the machine. Machine's *tolerance coefficient* is equal to the difference between its schedule length and the current makespan plus the setup length currently to be executed on that machine. The *tolerance coefficient* is calculated for every machine involved in the conflict, and setups on machines with $k - r$ lowest tolerance coefficients are postponed.

The algorithm is visualized in Fig. 4. It has polynomial time complexity; $\mathcal{O}((m + r \cdot \log(m) + t) \cdot t)$ if priority queues are used where possible and if only relevant places are checked for setup conflicts. Same as for LOSOS, we consider time complexities of external

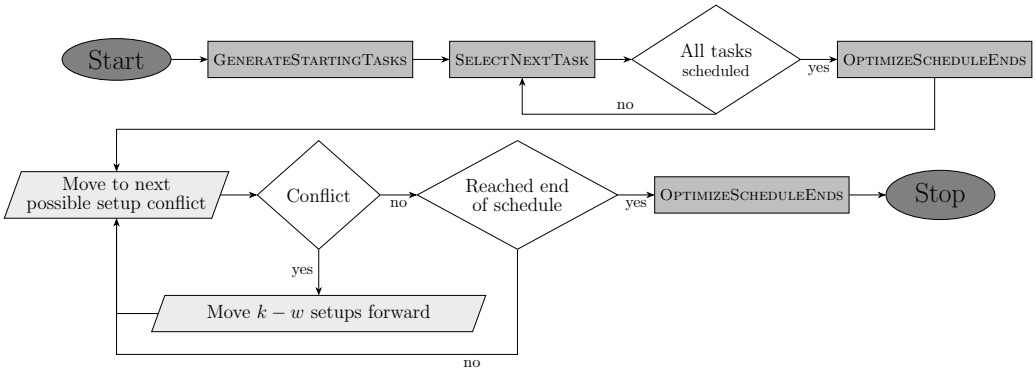

Figure 4: Flowchart of ROSOL algorithm.

methods to be $\mathcal{O}(m)$ for GenerateStartingTasks and OptimizeScheduleEnds and $\mathcal{O}(t)$ for SelectNextTask.

5.3. Constructive Heuristics Improvements

Algorithms LOSOS and ROSOL are efficient in most cases, but they are greedy heuristics, exploring only a very limited part of solution space. For certain instances, this can lead to a solution very far from the optimum. Therefore, this section focuses on making both algorithms take less greedy and more informed decisions by extending methods GenerateStartingTasks, OptimizeScheduleEnds and SelectNextTask. The proposed improvements are:

- Selecting starting tasks in an informed way. (GenerateStartingTasks)
- Optimizing the endings of machine schedules with additional task swapping. (OptimizeScheduleEnds)
- Improving setup/task selection using task priority coefficient. (SelectNextTask)
- Improving setup/task selection using idleness reduction. (SelectNextTask)

5.3.1. Starting Tasks Selection

There is no way of choosing the best possible starting tasks without solving the entire problem, but we can devise methods that will, on average, improve the results. The description of the best performing method out of the tested ones follows.

Consider $T^{StartingTasks}$ denoting m selected starting tasks and z_j denoting $\min_{T_i \in T}(o_{i,j})$, i.e., the shortest setup from any $T_i \in T$ to a given T_j. Then, $T^{StartingTasks}$ are selected such that for every $T_{Starting} \in T^{StartingTasks}$, $z_{Starting}$ is one of the m biggest z_j from all $T_j \in T$. In other words, we are looking for m tasks with their minimal preceding setup being one of m biggest ones from all tasks. This way, we avoid performing setups for tasks that would require a long time, even when placed after their best compatible preceding task. Since only setup times vary depending on the sequence, testing shows a considerable improvement over picking tasks randomly. The asymptotic time complexity of the resulting method is $\mathcal{O}(t^2)$, thus having no impact on the asymptotic time complexity of LOSOS or ROSOL when used.

Please note that this method works well because we assume all machines to execute their first task without a setup. If we would want this task selection method to work well in a case where setup is also necessary before the first task, we could add m virtual zero-length tasks with required setup lengths to the other tasks and infinite setup lengths from all other tasks to these virtual tasks.

5.3.2. End of Schedule Optimization

It has been observed that the most problematic part of the schedule is its end. The remaining tasks have only a few possible options where to be scheduled with their best compatible preceding tasks usually already unavailable, resulting in long setups and large differences between individual machine schedule lengths. Re-optimizing the end of the schedule reduces this effect. It is split into two phases:

1. First, we find the machine with the longest schedule, denoted $M_{longest}$. We take its last task and check if it can be moved to another machine while decreasing the makespan. If yes, we move it to the machine with the shortest resulting schedule after the move. We find new $M_{longest}$ and repeat this step until it decreases the makespan.

2. Second, we switch ending tasks between pairs of machines to reduce the makespan further. If possible, a pair of machines is chosen so that the makespan decreases after the switch. If not, pair of machines is chosen such that the maximum schedule length of the pair decreases while the makespan does not increase, giving us a pos-

sibility for future improvement. The best pair in regards to reducing the makespan or machine schedule length is selected. Again, we repeat this until no machine pair meets the criteria.

Potentially the optimization could be improved by looping the steps or considering more than just subsets of two machines. However, this would impact the time complexities of LOSOS and ROSOL and the analysis of built schedules showed that the application of these two steps improved and equalized the schedule well enough. Considering the properties of machine schedules, the asymptotic time complexity of the resulting method is $\mathcal{O}(m^2)$, thus having no impact on the asymptotic time complexity of LOSOS or ROSOL ($m \leq t$) when used.

5.3.3. Coefficient of Task Selection Priority

The baseline version of SELECTNEXTTASK method used in LOSOS and ROSOL chooses the next task to be scheduled greedily based on the setup length minimization. This works well for some distributions of setup lengths, but it has a fundamental flaw. As we deplete tasks with short setup times, it can happen that tasks having only long setups will remain, causing huge setbacks at the end of the schedule. The problem is further amplified because as long setups pile up at the end, they cause a shortage of available servers. If long setups were dispersed throughout the solution against the shorter ones, the problem would not occur. Hence, it is often better to take the locally suboptimal solution to obtain shorter setup times at the end of the schedule.

Thus, we are looking for a way of predicting which tasks should be saved for later and which are to be used immediately. As one possible way to evaluate this, we propose a function calculating so-called *task coefficient* for the given task. We can think of this *task coefficient* as a measure of future usefulness compared to the usefulness now. So unlike in the baseline version of SELECTNEXTTASK, where only the currently scheduled task and the next one to follow were considered, other possible future placements of the next task are considered as well.

The calculation is following. Let T_i represent currently ending scheduled task, T_j represent task considered to follow T_i and $T_{unresolved}$ denote the set of tasks waiting to be scheduled or currently being executed. Then $T_{x1}, T_{x2}, T_{x3} \in T_{unresolved}$ denote tasks

with smallest, second smallest and third smallest setup time to T_j respectively, with setups denoted as $o_{x1,j}$, $o_{x2,j}$ and $o_{x3,j}$. The resulting *task coefficient* is then calculated as follows:

$$\begin{aligned} &\mathit{TaskCoefficient}_j(T_i, T_{unresolved}) \\ &= \underbrace{o_{i,j}^4}_{\text{current usefulness}} + \underbrace{|o_{i,j} - o_{x1,j}| \cdot (o_{i,j} - o_{x1,j}) + |o_{i,j} - o_{x2,j}| \cdot (o_{i,j} - o_{x2,j}) + (o_{i,j} - o_{x3,j})}_{\text{possible future usefulness}}. \end{aligned} \tag{1}$$

It is clear from Eq. (1) that the largest emphasis is still on setup between T_i and T_j, but other future scheduling possibilities are taken into account. To decide which task to follow after T_i, *task coefficients* are calculated for all possible tasks to follow T_i, and the one with the smallest *task coefficient* is chosen.

We considered several polynomial functions with different numbers and different powers of elements, but experiments showed that the one proposed in Eq. (1) performs the best. However, the efficiency of *task coefficient* is influenced by the setup times distribution. Its asymptotic time complexity is $\mathcal{O}(t^2)$ but since SelectNextTask is called t times in both LOSOS and ROSOL, it increases the overall complexity of both algorithms.

5.3.4. Idleness Reduction

We can further improve SelectNextTask method by considering future availabilities of servers when only one currently unoccupied one remains. This is only applicable when calling SelectNextTask from LOSOS as ROSOL handles servers after task assignments and orders are already decided.

Let T_i denote the currently ending task, T_j denote the next earliest ending task except for T_i and $T_{unscheduled}$ denote the set of yet unscheduled tasks. Let R_l denote the only available unoccupied server at the moment when the setup after T_i is being scheduled and R_s denote the next earliest ending server except for R_l. Now, let us consider two issues with the current way of setup scheduling:

1. If we choose $T_x \in T_{unscheduled}$ to follow after T_i with too long $o_{i,x}$ in-between, it might happen that when T_j ends, both R_l and R_s will be occupied, thus no server will be available to execute setup following T_j. Therefore, the machine with T_j scheduled on it would idle instead of executing the following setup.

2. If we assign too short $o_{i,x}$ to R_l, then after $o_{i,x}$ is finished, there might be no machine requiring the server. In consequence, R_l would idle instead of executing other longer $o_{i,x}$, which might have to be executed later, thus negatively affecting the makespan. Instead, executing a longer setup would maximize this free window's use and possibly shorten future setup times.

To address issue 1, we calculate difference between ends of T_j and T_i as $t_d = c_j - c_i$. Then, we choose task T_x with $o_{i,x} \leq t_d$, so after R_l finishes setup between T_i and T_x, it will be ready to execute setup following T_j once T_j finishes. If more tasks have $o_{i,x} \leq t_d$, we choose one according to the selection criteria, e.g. *task coefficient*. In case there is no $T_x \in T_{unresolved}$ with $o_{i,x} \leq t_d$, we pick T_x with the smallest possible $o_{i,x}$ and again use selection criteria for possible tie breaking.

We only consider issue 2 when we have some estimation of possible task usefulness elsewhere, like the aforementioned *task coefficient*. Otherwise, we could potentially execute longer setups without a good reason. We calculate t_d again but embed it into the *task coefficient* calculation, changing $o_{i,j}^4$ in *task coefficient* equation to $max(0, o_{i,j} - t_d)^4$. This way, we only emphasize the portion of the setup time, where the server could already work on a different machine, weighing more other scheduling possibilities.

Addressing issue 1 is more important than addressing issue 2 since we rather let server wait while machines execute as we minimize the makespan. Thus, if issue 1 arises, we disregard issue 2 and also any task utility metric like *task coefficient*. It is clear that both issues and *task coefficient* are partially in a trade-off, and some weighted decision between them could be derived, but this was not further examined. Since the *task coefficient* must be executed because of the issue 2 and the time complexity of additional operations is lower, the overall asymptotic time complexity is $\mathcal{O}(t^2)$. Since SELECTNEXTTASK is called t times in both LOSOS and ROSOL, it increases the overall complexity of both algorithms.

5.4. Model Warm Starting: Synergy of the Approaches

To improve the CP model's computational performance, we consider warm starting, a commonly used technique of obtaining a feasible or partial starting solution by using existing efficient algorithms for a similar problem. We developed and tested multiple warm starting methods but using the constructive heuristics proposed in this section

yielded the best results as they provide quality solutions in a very short time.

Because the execution time of constructive heuristics is negligible compared to the CP model, we execute both and select the better solution as a warm start. Based on preliminary testing, the most efficient configuration is to execute LOSOS with all improvements and ROSOL with starting task selection and end of schedule optimization. As we showed before, improvements to SelectNextTask method noticeably increase computational complexity, so executing ROSOL without them saves computational time for the CP model. Also, since task coefficient and idleness reduction do not guarantee improvement, having diversity between LOSOS and ROSOL solution is good.

The initial solution is passed to the CP model by assigning tasks to machines given the warm start solution assignment and ordering. We also add setup time spaces between tasks and set their start and end times accordingly.

6. Experimental Evaluation

In this section, we compare our proposed approaches to each other and then to the state-of-the-art approaches to similar problems. All experimental results were executed at a single core of the Intel Xeon 4110 processor running at 2.1GHz. A graphic card was not used in any of the calculations. Algorithms were implemented using C++. For CP, *IBM ILOG CP Optimizer 20.1.0* was used, while for ILP, *Gurobi 9.5* was used. All generated instances and solutions obtained by different approaches can be found enclosed on the GitLab repository [30].

In all following tables, m denotes the number of machines, t the number of tasks and r the number of servers. By symbol ∞, we denote that no feasible solution was found. The approaches are compared by the Relative Difference (RD) calculated as $(compared - baseline)/baseline = RD$, where *compared* denotes the objective value of the evaluated approach and *baseline* denotes the objective value of the approach (or lower bound) to which the comparison is made.

6.1. Comparison of Our Approaches

We decided to compare all our approaches on the same instance set to show where their efficiencies meet. To make this possible, m is set between 12 and 20, t between $15m$

and $20m$ and r is either 2 or 5. The processing and setup times of instances used in this subsection were randomly sampled from a uniform distribution between 1 and 50. The range of the distribution is based on *Hall et al.* [31] which states that instances with both big and small variances of their attributes should be involved in the evaluation process. In subsection 6.2, we evaluate using methodologies given by the compared papers. Based on these comparisons, we show that our approaches are performant on various different ranges of attributes. However, no distribution of attributes in the compared papers has a range starting from 1. Thus, we include instances with such high variance in the comparison of our own. The uniform distribution was chosen as it is a common practice.

Since the constructive heuristics took between 1ms and 650ms to execute depending on the particular instance and improvements used, to obtain relevant comparison, we set CP model executions time limits considerably small. However, the results of supplementary experiments provided in Appendix D show that warm started CP models can be effectively applied to instances of up to 50 machines and 1000 tasks. With an hour time limit, the relative gap between the heuristics and CP model solutions is higher than on the tests performed in Table 2.

In further text, LOSOS and ROSOL denote algorithms without any improvements applied, LOSOS_{SE} denotes LOSOS execution with GenerateStartingTasks and OptimizeScheduleEnds improvements applied and $\text{LOSOS}_{\text{SEIC}}$ execution with all improvements applied. There is no $\text{ROSOL}_{\text{SEIC}}$ as idleness reduction cannot be applied to ROSOL and the results for ROSOL_{SE} are not reported since the improvement achieved is very similar to LOSOS_{SE}. To distinguish CP model variants, CP_0 denotes model without warm start, $\text{CP}_{\text{WS-SE}}$ denotes model with better solution from LOSOS_{SE} and ROSOL_{SE} used as warm start and $\text{CP}_{\text{WS-SEIC}}$ denotes model with better solution from $\text{LOSOS}_{\text{SEIC}}$ and ROSOL_{SE} used as a warm start. All results referenced in this subsection are in the Table 2. The last column of the table contains lower bounds. The percentage in the round brackets in columns $\text{LOSOS}_{\text{SEIC}}$ and $\text{CP}_{\text{WS-SEIC}}$(10s) denotes RD to lower bound for each instance. The lower bound calculation with additional information can be found in Appendix C.

Constructive heuristics. The difference between LOSOS and ROSOL is not very pronounced, with ROSOL being slightly better. A more noticeable difference between the two can be observed on instances with a smaller number of servers since ROSOL assigns

servers to setups in a more informed way. Regarding the effect of improvements proposed in Section 5.3, based on the results from Table 2, application of all improvements together yields more than 4 % average decrease in the objective. We consider this enhancement notable given the sizes of relative gaps between the heuristics and the respective lower bounds provided in Table 2. However, for a few instances like number 1 or 14 in Table 2, $\text{LOSOS}_{\text{SEIC}}$ does not yield the best solution out of all heuristics as it does not guarantee improvement. Thus, if constructive heuristics are used without the CP model, it would make sense to execute LOSOS_{SE}/ROSOL_{SE} and $\text{LOSOS}_{\text{SEIC}}$ and pick the better solution as LOSOS_{SE} has fairly negligible execution time. Regarding heuristics scalability, the graph provided in Appendix B shows that without the use of task coefficient and idleness reduction, even instances of hundreds of machines and tens of thousands of tasks can be solved in several seconds to several minutes of runtime.

CP models. Considering CP models, testing shows that CP_0 is not very usable for larger instances. The warm started variants yield incomparably better results than CP_0 in a fraction of its time limit. They also noticeably improve the warm started solution, showing that the model scales well and is effective once a reasonable starting solution is provided. In cases where the warm started model produced a solution with the same objective as the warm start, it was still able to construct the problem and search part of the solution space, meaning the problem was still feasible for the model in such a short time limit given. Using all improvements in warm starting showed beneficial as $\text{CP}_{\text{WS-SEIC}}$ provided better results than $\text{CP}_{\text{WS-SE}}$. Also, note that the warm starting is included within the time limit.

	parameters			objective value [-]							
#	m	t	r	LOSOS	ROSOL	LOSOS_{SE}	$\text{LOSOS}_{\text{SEIC}}$	CP_0(120s)	$\text{CP}_{\text{WS-SE}}$(10s)	$\text{CP}_{\text{WS-SEIC}}$(10s)	LB
1	12	180	2	436	418	427	419 (9.7%)	406	**398**	399 (4.5%)	382
2	12	180	5	418	418	418	409 (7.1%)	**399**	403	402 (5.2%)	382
3	12	240	2	580	563	553	541 (8.4%)	1798	**515**	518 (3.8%)	499
4	12	240	5	545	545	543	542 (8.6%)	564	519	**517 (3.6%)**	499
5	12	300	2	738	763	719	704 (5.9%)	7741	688	**685 (3.0%)**	665
6	12	300	5	743	743	712	710 (6.8%)	5492	686	**685 (3.0%)**	665
7	14	210	2	460	458	443	430 (9.4%)	1464	413	**412 (4.8%)**	393
8	14	210	5	440	440	434	426 (8.4%)	**407**	415	410 (4.3%)	393
9	14	280	2	612	615	595	587 (8.7%)	6406	**563**	**563 (4.3%)**	540
10	14	280	5	615	615	583	569 (5.4%)	6501	561	**557 (3.1%)**	540
11	14	350	2	759	741	752	748 (8.7%)	2950	**727**	**727 (5.7%)**	688
12	14	350	5	730	730	730	720 (4.7%)	3623	713	**710 (3.2%)**	688
13	16	240	2	456	414	433	434 (16.0%)	4823	**395**	**395 (5.6%)**	374
14	16	240	5	407	406	407	408 (9.1%)	1068	**394**	**394 (5.3%)**	374
15	16	320	2	603	609	591	569 (6.8%)	7775	573	**560 (5.1%)**	533
16	16	320	5	580	580	580	557 (4.5%)	8261	553	**551 (3.4%)**	533
17	16	400	2	732	728	708	**687 (5.7%)**	9769	700	**687 (5.7%)**	650
18	16	400	5	707	707	698	**679 (4.5%)**	6824	697	**679 (4.5%)**	650
19	18	270	2	436	433	411	437 (19.7%)	1717	**400**	403 (10.4%)	365
20	18	270	5	411	411	402	405 (11.0%)	5758	**382**	**382 (4.7%)**	365
21	18	360	2	579	577	573	552 (10.4%)	7260	**549**	**549 (9.8%)**	500
22	18	360	5	557	550	539	534 (6.8%)	8522	533	**525 (5.0%)**	500
23	18	450	2	722	720	698	703 (9.8%)	7681	**686**	**686 (7.2%)**	640
24	18	450	5	717	717	686	**680 (6.2%)**	8514	686	**680 (6.2%)**	640
25	20	300	2	493	472	473	457 (14.5%)	5630	**445**	**445 (11.5%)**	399
26	20	300	5	454	447	445	434 (8.8%)	7550	430	**419 (5.0%)**	399
27	20	400	2	588	577	568	568 (9.4%)	9822	**547**	**547 (5.4%)**	519
28	20	400	5	558	558	**546**	551 (6.2%)	8943	**546**	**546 (5.2%)**	519
29	20	500	2	765	752	743	724 (10.9%)	12811	**705**	**705 (8.0%)**	653
30	20	500	5	715	716	697	**676 (3.5%)**	11874	695	**676 (3.5%)**	653
$\sum$	-	-	-	17556	17423	17107	16860	172353	16517	**16414**	15600
RD	-	-	-	12.54 %	11.69 %	9.66 %	8.08 %	1004.83 %	5.88 %	**5.22 %**	0.0 %

Table 2: The comparison of all proposed approaches.

6.2. Comparison to Existing Approaches

To evaluate whether our approaches have state-of-the-art quality, we compare them with existing ones. This is important as a comparison only between our approaches provides no external point of reference in regards to their overall quality. However, as the literature review revealed, there is no paper tackling our considered $P|seq, ser|C_{max}$ problem. Thus, we use problems $PD|seq, ser = 1|C_{max}$, $P|ser = 1|C_{max}$ and $P|seq, ser = 1|C_{max}$ for the comparison as they are just restricted variants of $P|seq, ser|C_{max}$. The instances of said problems are transformed to the instances of $P|seq, ser|C_{max}$ so our approaches can be used to solve them. However, this makes the instances more complex than the original ones, even if their set of feasible solutions is the same, so our approaches might have to do more computations when solving them.

To compare the existing exact approaches realized by ILP models to our CP_0, we

formalized them using Gurobi 9.5. This way, we could make a direct comparison on the same hardware and with the same time limits, thus obtaining an entirely fair comparison. The problem instances used in the comparison were generated according to the descriptions provided in each respective paper.

As for the heuristic comparison, we compare our proposed heuristic alternative $\mathsf{CP}_{\mathsf{WS\text{-}SEIC}}$ to existing heuristics. Since the comparison methodology differs for every paper, it is described in each paper's respective subsection. Again, all instances were generated according to their respective original paper descriptions.

All comparisons are made on instance sizes chosen by the original paper. Table layouts are not entirely consistent because they partially adopt layouts from their respective papers to make the comparison easier to evaluate.

6.2.1. Dedicated Machines with Single Server

In *Huang et al.* [2], problem denoted $PD|seq, ser = 1|C_{max}$, where only 1 server is present, and tasks are dedicated to machines is considered. The $PD|seq, ser = 1|C_{max}$ instances were transformed to $P|seq, ser|C_{max}$ instances by setting infinite length setup times between tasks that are not supposed to be on the same machine. This way, if the CP solver would schedule tasks originally dedicated to different machines on the same machine, the resulting schedule would be infinitely long, thus infeasible.

There were no computational results provided for the MIP model in *Huang et al.* [2], only a statement that it performs poorly. In comparison, both ILP and CP_0 were given a 3600-second time limit. Even though CP_0 had to tackle larger problem instances, it performed much better, solving all instances and in most cases proving the optimality as well. The results can be seen in Table 3. Note that t/m in Table 3 denotes the number of tasks per machine.

	parameters				objective value [-]		CPU time [s]	
#	m	t/m	Setup LB	Setup UB	MIP[2]	CP_0	MIP[2]	CP_0
1	2	4	5	25	**329**	**329**	0.154	**0.015**
2	2	4	5	50	**351**	**351**	0.137	**0.006**
3	2	4	25	50	**394**	**394**	0.109	**0.014**
4	2	6	5	25	**403**	**403**	3.676	**0.698**
5	2	6	5	50	**423**	**423**	22.444	**0.082**
6	2	6	25	50	**508**	**508**	134.017	**0.711**
7	2	8	5	25	**469**	**469**	682.419	**1.194**
8	2	8	5	50	**489**	**489**	654.805	**0.184**
9	2	8	25	50	**627**	**627**	3600.0	**13.423**
10	3	6	5	25	**419**	**419**	54.212	**1.443**
11	3	6	5	50	**432**	**432**	24.088	**1.576**
12	3	6	25	50	543	**539**	3600.0	**28.06**
13	3	9	5	25	682	**673**	3600.0	**4.838**
14	3	9	5	50	∞	**693**	3600.0	**5.317**
15	3	9	25	50	941	**837**	3600.0	**14.113**
16	3	12	5	25	∞	**743**	3600.0	**88.882**
17	3	12	5	50	∞	**767**	3600.0	**69.836**
18	3	12	25	50	∞	**1042**	**3600.0**	**3600.0**
19	4	8	5	25	577	**552**	3600.0	**11.106**
20	4	8	5	50	754	**582**	3600.0	**10.479**
21	4	8	25	50	1063	**831**	**3600.0**	**3600.0**
22	4	12	5	25	∞	**768**	**3600.0**	**3600.0**
23	4	12	5	50	∞	**813**	**3600.0**	**3600.0**
24	4	12	25	50	∞	**1278**	**3600.0**	**3600.0**
25	4	16	5	25	∞	**1078**	3600.0	**2065.339**
26	4	16	5	50	∞	**1097**	**3600.0**	**3600.0**
27	4	16	25	50	∞	**1678**	**3600.0**	**3600.0**

Table 3: MIP model proposed in *Huang et al.* [2] compared to CP_0.

The heuristic comparison in *Huang et al.* [2] was realized by comparing their Genetic Algorithm (GA) to a lower bound denoted as LB_2. We calculated the LB_2 on the generated instances and compared RD between our CP model and LB_2 on said instances to their RD between GA and LB_2 on their instances (in paper denoted as GA-LB_2 gap), obtaining a reasonably fair comparison. Because $CP_{WS\text{-}SEIC}$ uses constructive heuristics for warm starting and those do not respect machine dedication, we had to use much less effective CP_0. The time limits were set according to the runtimes of the GA in the original paper. Note that t/m in Table 4 denotes the number of tasks per machine.

The results in Table 4 show that out of 30 cases, CP_0 provided better RD to LB_2 than GA in 16 instances and worse in 13 instances. CP_0 excelled at solving smaller instances, but its effectivity decreased with increasing size. This is because we had to use CP_0 instead of $CP_{WS\text{-}SEIC}$, solved a larger problem instance than the GA and also because of the very short time limit, which would generally favour heuristics. The largest tested instance had only a 30-second time limit. In conclusion, the comparison shows that even when tackling a much more restricted case of our problem with our less tailored approach

without its main improvements applied, it still produces competitive results.

	parameters				objective value [-]		RD [%]	
#	m	t/m	Setup LB	Setup UB	LB2	CP_0	GA[2]	CP_0
1	2	5	5	25	380	380	**0.0**	**0.0**
2	2	5	5	50	390	390	0.47	**0.0**
3	2	5	25	50	463	463	0.43	**0.0**
4	2	10	5	25	671	671	0.46	**0.0**
5	2	10	5	50	691	691	1.1	**0.0**
6	2	10	25	50	855	855	1.08	**0.0**
7	3	5	5	25	380	380	0.04	**0.0**
8	3	5	5	50	390	390	0.42	**0.0**
9	3	5	25	50	463	463	1.56	**0.0**
10	3	10	5	25	689	689	0.57	**0.0**
11	3	10	5	50	717	717	1.51	**0.0**
12	3	10	25	50	874	926	**1.82**	5.95
13	5	5	5	25	380	380	0.09	**0.0**
14	5	5	5	50	390	390	1.94	**0.0**
15	5	5	25	50	552	605	**2.03**	9.6
16	5	10	5	25	689	689	0.69	**0.0**
17	5	10	5	50	717	740	**2.14**	3.21
18	5	10	25	50	1225	1353	**5.48**	10.45
19	7	5	5	25	380	380	0.48	**0.0**
20	7	5	5	50	390	413	**4.38**	5.9
21	7	5	25	50	769	827	**1.89**	7.54
22	7	10	5	25	689	692	1.18	**0.44**
23	7	10	5	50	717	846	**6.37**	17.99
24	7	10	25	50	1683	1823	**5.94**	8.32
25	10	5	5	25	402	402	2.36	**0.0**
26	10	5	5	50	418	541	**6.99**	29.43
27	10	5	25	50	1112	1180	**2.31**	6.12
28	10	10	5	25	689	800	**3.81**	16.11
29	10	10	5	50	717	1186	**22.98**	65.41
30	10	10	25	50	2391	2568	**6.87**	7.4

Table 4: GA proposed in *Huang et al.* [2] compared to CP_0.

6.2.2. Sequence-Independent Setups with Single Server

In *Kim et al.* [3], problem denoted $P|ser = 1|C_{max}$, where only 1 server is present, and setups are sequence-independent and executed before every task is considered. The $P|ser = 1|C_{max}$ instances were transformed to $P|seq, ser|C_{max}$ instances in two steps. First, the instance's setup times matrix was extended to sequence-dependent format by simply duplicating the setup times. Second, m zero-length tasks were added to the instance, each one ending up as the first task on one of the machines. Then, the setups between these zero-length tasks and their following real tasks emulate the setup before the first task on each machine in $P|ser = 1|C_{max}$ problem.

We implemented a better performing MIP model from *Kim et al.* [3] denoted as MIP-2 and compared it to our CP_0. Table 5 shows the comparison results for instances of 6 machines with either 20, 30, or 40 tasks, with two additional settings, α and p. Parameter

α represents the amount of variance in task and setup times, where 0.1 means a maximum of 10 % deviation from average on either side. Parameter p describes a multiplier of the setup length; the bigger it is, the longer the setups can be. The results show that CP_0 always returned the same or better solution than the MIP model. The models were given a 3600-second time limit.

parameters		**(20, 6)**		**(30, 6)**		**(40, 6)**	
α	p	MIP-2[3]	CP_0	MIP-2[3]	CP_0	MIP-2[3]	CP_0
0.1	0.5	**205**	**205**	284	**278**	382	**370**
0.1	0.7	213	**210**	291	**288**	390	**380**
0.1	1	**225**	**225**	322	**316**	420	**413**
0.3	0.5	190	**186**	278	**274**	375	**366**
0.3	0.7	193	**192**	291	**282**	383	**375**
0.3	1	**212**	**212**	314	**307**	415	**403**
0.5	0.5	190	**186**	285	**279**	381	**373**
0.5	0.7	193	**189**	293	**282**	387	**377**
0.5	1	213	**211**	324	**309**	421	**408**

Table 5: MIP-2 model proposed in *Kim et al.* [3] compared to CP_0.

To evaluate the efficiency of the hybrid heuristic denoted as HA in *Kim et al.* [3], authors calculated RD (in paper denoted as gap percentage) between HA results and results of MIP-2 running for 3600 seconds. Because we obtained MIP-2 results in the previous comparison, we can compare HA against $\mathsf{CP}_{\text{WS-SEIC}}$ by comparing their calculated RD with RD between $\mathsf{CP}_{\text{WS-SEIC}}$ and MIP-2. This comparison should be fair even if there is any difference between their and our hardware because the MIP-2 results and the $\mathsf{CP}_{\text{WS-SEIC}}$ results were obtained on the same computer. Thus, if our machine would be faster, the MIP-2 results to which $\mathsf{CP}_{\text{WS-SEIC}}$ is compared would also profit from the computational power increase. $\mathsf{CP}_{\text{WS-SEIC}}$'s time limit was set according to the HA's runtime limit stated in *Kim et al.* [3].

The results in Table 6 show that for every single instance, $\mathsf{CP}_{\text{WS-SEIC}}$ provided a better solution than HA. In 25 out of 30 instances, $\mathsf{CP}_{\text{WS-SEIC}}$ with limited time also provided better results than MIP-2 running for 3600 seconds, with only 2 instances being worse. We believe that the consistency and size of the difference ensure that our approach improves over the compared one, even if the comparison method would slightly influence the results.

	parameters				objective value [-]		RD [%]	
#	m	t	p	α	MIP-2[3]	$\text{CP}_{\text{WS-SEIC}}$	HA[3]/MIP-2[3]	$\text{CP}_{\text{WS-SEIC}}$/MIP-2[3]
1	6	20	0.5	0.1	205	205	1.87	**0.0**
2	6	20	0.7	0.1	213	210	2.34	**-1.41**
3	6	20	1	0.1	225	225	2.72	**0.0**
4	6	20	0.5	0.3	190	186	1.06	**-2.11**
5	6	20	0.7	0.3	193	193	2.15	**0.0**
6	6	20	1	0.3	212	216	2.72	**1.89**
7	6	20	0.5	0.5	190	187	3.15	**-1.58**
8	6	20	0.7	0.5	193	190	2.02	**-1.55**
9	6	20	1	0.5	213	214	4.03	**0.47**
10	6	30	0.5	0.1	284	280	1.24	**-1.41**
11	6	30	0.7	0.1	291	289	2.21	**-0.69**
12	6	30	1	0.1	322	320	1.74	**-0.62**
13	6	30	0.5	0.3	278	275	1.29	**-1.08**
14	6	30	0.7	0.3	291	283	2.65	**-2.75**
15	6	30	1	0.3	314	313	2.42	**-0.32**
16	6	30	0.5	0.5	285	279	2.68	**-2.11**
17	6	30	0.7	0.5	293	284	2.9	**-3.07**
18	6	30	1	0.5	324	312	2.7	**-3.7**
19	6	40	0.5	0.1	382	372	2.22	**-2.62**
20	6	40	0.7	0.1	390	382	3.55	**-2.05**
21	6	40	1	0.1	420	415	2.05	**-1.19**
22	6	40	0.5	0.3	375	367	1.4	**-2.13**
23	6	40	0.7	0.3	383	376	2.7	**-1.83**
24	6	40	1	0.3	415	407	4.66	**-1.93**
25	6	40	0.5	0.5	381	374	0.91	**-1.84**
26	6	40	0.7	0.5	387	378	2.36	**-2.33**
27	6	40	1	0.5	421	414	3.86	**-1.66**

Table 6: HA proposed in *Kim et al.* [3] compared to $\text{CP}_{\text{WS-SEIC}}$.

6.2.3. Single Server

In *Hamzadayi et al.* [4], $P|seq, ser = 1|C_{max}$ problem is considered, meaning the only difference compared to our problem is the presence of just 1 server. For the exact approach comparison, we changed the original testing time limit from 18000 to 3600 seconds to obtain the MIP model and CP_0 results in a more reasonable time. It can be seen from the Table 7 that CP_0 was always on par or better than the MIP model. In fact, in cases where results were the same, we know that CP_0 reached the optimal solution. Model CP_0 was also able to prove optimality in some instances where the MIP model did not even reach the optimal solution.

	parameters		objective value [-]		CPU time [s]	
#	m	t	MIP[4]	CP_0	MIP[4]	CP_0
1	2	6	**200**	**200**	2.135	**0.043**
2	2	8	**237**	**237**	144.502	**0.305**
3	2	10	**334**	**334**	3600.0	**0.957**
4	2	14	441	**415**	3600.0	**3600.0**
5	2	20	687	**611**	3600.0	**3600.0**
6	3	9	**188**	**188**	577.058	**0.696**
7	3	12	279	**253**	3600.0	**7.166**
8	3	15	376	**346**	3600.0	**3600.0**
9	3	21	∞	**373**	**3600.0**	**3600.0**
10	3	30	∞	**667**	**3600.0**	**3600.0**
11	4	12	205	**196**	3600.0	**15.741**
12	4	16	231	**199**	3600.0	**3600.0**
13	4	20	381	**309**	**3600.0**	**3600.0**
14	4	28	∞	**448**	**3600.0**	**3600.0**
15	4	40	∞	**656**	**3600.0**	**3600.0**
16	5	15	230	**209**	3600.0	**1972.226**
17	5	20	369	**247**	**3600.0**	**3600.0**
18	5	25	668	**299**	**3600.0**	**3600.0**
19	5	35	∞	**458**	**3600.0**	**3600.0**
20	5	50	∞	**669**	**3600.0**	**3600.0**
21	7	21	265	**167**	3600.0	**3600.0**
22	7	28	∞	**264**	**3600.0**	**3600.0**
23	7	35	∞	**334**	**3600.0**	**3600.0**
24	7	49	∞	**425**	**3600.0**	**3600.0**
25	7	70	∞	**640**	**3600.0**	**3600.0**
26	10	30	∞	**210**	**3600.0**	**3600.0**
27	10	40	∞	**275**	**3600.0**	**3600.0**
28	10	50	∞	**355**	**3600.0**	**3600.0**
29	10	70	∞	**457**	**3600.0**	**3600.0**
30	10	100	∞	**678**	**3600.0**	**3600.0**

Table 7: MIP model proposed in *Hamzadayi et al.* [4] compared to CP_0.

The authors in *Hamzadayi et al.* [4] evaluated their GA effectivity by comparing it to their other proposed approaches. However, most instances had no results for the MIP model, so we had no way of fairly comparing $\mathsf{CP}_{\text{WS-SEIC}}$ to the GA without executing the GA ourselves. Thus, we reimplemented the GA according to the paper's description and to the best of our knowledge, denoting it in the following text as GA_R.

The results in Table 8 show $\mathsf{CP}_{\text{WS-SEIC}}$ outperforming GA_R in every case except two, where they yielded the same result. The runtime of $\mathsf{CP}_{\text{WS-SEIC}}$ for each instance was set equal to the runtime obtained by GA_R on our hardware, which was lower than runtimes reported in *Hamzadayi et al.* [4]. Thus, $\mathsf{CP}_{\text{WS-SEIC}}$ had no unfair advantage considering the time limit. Every machine-task combination was run 20 times to even out the GA's random nature, and the rounded average was reported. The last column shows the RD of objective values of GA_R to $\mathsf{CP}_{\text{WS-SEIC}}$. It is noticeable that the difference gets more significant with the growing instance's size. This is probably the result of GA's inability

to effectively find improvements in such a vast solution space.

	parameters		objective value [-]		RD [%]
#	m	t	$\text{CP}_{\text{WS-SEIC}}$	GA_{R}[4]	GA_{R}[4] to $\text{CP}_{\text{WS-SEIC}}$
1	2	6	**201**	**201**	0.0 %
2	2	8	**255**	256	0.39 %
3	2	10	**333**	337	1.2 %
4	2	14	**443**	464	4.74 %
5	2	20	**650**	693	6.62 %
6	3	9	**200**	**200**	0.0 %
7	3	12	**267**	271	1.5 %
8	3	15	**331**	340	2.72 %
9	3	21	**451**	479	6.21 %
10	3	30	**643**	696	8.24 %
11	4	12	**204**	211	3.43 %
12	4	16	**253**	262	3.56 %
13	4	20	**332**	351	5.72 %
14	4	28	**458**	495	8.08 %
15	4	40	**649**	714	10.02 %
16	5	15	**206**	213	3.4 %
17	5	20	**272**	287	5.51 %
18	5	25	**332**	355	6.93 %
19	5	35	**447**	492	10.07 %
20	5	50	**661**	741	12.1 %
21	7	21	**212**	223	5.19 %
22	7	28	**280**	304	8.57 %
23	7	35	**334**	373	11.68 %
24	7	49	**460**	528	14.78 %
25	7	70	**646**	755	16.87 %
26	10	30	**230**	251	9.13 %
27	10	40	**294**	327	11.22 %
28	10	50	**369**	421	14.09 %
29	10	70	**495**	580	17.17 %
30	10	100	**713**	844	18.37 %

Table 8: GA proposed in *Hamzadayi et al.* [4] compared to $\text{CP}_{\text{WS-SEIC}}$.

7. Conclusion and Future Work

This paper addressed the scheduling of tasks on non-dedicated machines with sequence-dependent setups that are performed by servers. Initially, the Constraint Programming model was proposed to formalize and optimally solve the problem. However, the CP model is suitable only for specific instance sizes. Thus, two polynomial constructive heuristics LOSOS and ROSOL were proposed. Since the execution of constructive heuristics is very fast, improvements to these methods were proposed, enhancing the solution's quality in a trade-off with the execution time. To reach the best solution possible, the CP model and constructive heuristics were combined into a warm started CP model called $\text{CP}_{\text{WS-SEIC}}$ which provides high-quality solutions even for short time limits and large instances. As demonstrated, $\text{CP}_{\text{WS-SEIC}}$ can effectively solve instances of tens of

machines and hundreds of tasks.

The efficiency of the proposed approaches was evaluated and compared against the existing state-of-the-art ones. Since, to the best of our knowledge, there is no paper tackling our considered $P|seq, ser|C_{max}$ problem, we compared our approaches to approaches for problems $PD|seq, ser = 1|C_{max}$, $P|ser = 1|C_{max}$ and $P|seq, ser = 1|C_{max}$. The comparison was achieved by transforming the instances from original problems to our problem, however, this meant that our approaches had to tackle more complex instances. Despite that, our CP model proved to be better than all compared MIP models, providing better or equal solutions to every single instance tested and also being much better at proving the optimality. As for the comparison of $\mathsf{CP}_{\mathsf{WS\text{-}SEIC}}$ to the heuristics for the compared problems, $\mathsf{CP}_{\mathsf{WS\text{-}SEIC}}$ provided on par results against genetic algorithm for the most restricted problem $PD|seq, ser = 1|C_{max}$ and better results than heuristics for $P|ser = 1|C_{max}$ and for $P|seq, ser = 1|C_{max}$. It also has the additional advantage of being able to reach optimum if enough time is provided.

In conclusion, we showed that our approaches are effective, relevant, and can be used for more specific problems with better efficiency than the existing approaches. We consider this the most significant contribution of this paper and see a great promise in applying proposed approaches in real-world production scenarios.

To accommodate even more real-world productions, we see great potential in introducing setups over setups to simulate operations like the movement of server from one machine to another, as we did not see this addressed anywhere in the literature.

Acknowledgements

This work was funded by EU Structural funds and the 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. This work was supported by the EU and the Ministry of Industry and Trade of the Czech Republic under the Project OP PIK CZ.01.1.02/0.0/0.0/20_321/0024399.

Appendix A. Constructive Heuristic Pseudocodes

Appendix A.1. Locally Optimal Selection of Setups (LOSOS)

Algorithm 1 Pseudocode of LOSOS algorithm.

```
1: function LOSOS SOLVE
2:     T^StartingTasks ← GENERATESTARTINGTASKS                    ▷ called function
3:     for each M_m ∈ M do
4:         Schedule M_m ← T_m^StartingTasks
5:     end for
6:     T^Remaining ← T \ {T^StartingTasks}
7:     WorkersFreeFromTime ← {0} → |R|                  ▷ List of zeroes of size |R|.
8:     while T^Remaining ≠ ∅ do
9:         ClosestEndingMachine ← argmin(End(M))
10:        ClosestEndingTask ← T_{−1,ClosestEndingMachine}      ▷ Last task is indexed by "-1".
11:        NextTask, SetupLength ← SELECTNEXTTASK(ClosestEndingTask, T^Remaining)
12:                                                             ▷ called function
13:        Start(NextSetup) ← max(End(ClosestEndingTask), min(WorkersFreeFromTime))
14:        Start(NextTask) ← Start(NextSetup) + SetupLength
15:        Schedule ClosestEndingMachine ← NextSetup
16:        Schedule ClosestEndingMachine ← NextTask
17:        argmin(WorkersFreeFromTime) ← Start(NextTask)
18:        T^Remaining ← T^Remaining \ {ClosestEndingTask}
19:    end while
20:    OPTIMIZESCHEDULEENDS                                     ▷ called function
21: end function
```

Starting tasks $T^{StartingTasks}$ are selected, and one is assigned to every machine. There are multiple possible criteria for choosing $T^{StartingTasks}$; the best one is discussed in detail in subsection 5.3.1. After these tasks are assigned, they are removed from the set of remaining tasks, which is denoted by $T^{Remaining}$. See lines 2 to 6.

Data structure representing all servers' availability is created, denoted by $WorkersFreeFromTime$. In algorithm implementation, the priority queue with the earliest available server on top was used. At the start, all servers are considered ready to work; thus, their availability is set to time 0. See line 8.

The earliest ending task, denoted by $ClosestEndingTask$, and its respective machine, denoted by $ClosestEndingMachine$, are found. In algorithm implementation, a priority queue was used for storing the machines. Then, a suitable task, denoted by $NextTask$, and its respective setup length from $ClosestEndingTask$ are found. The methods of $NextTask$ selection are discussed in detail in Section 5.3. See lines 11 to 14.

The earliest available server is selected to perform the resulting $NextSetup$ between $ClosestEndingTask$ and $NextTask$. The setup's starting time is calculated according to server's and $ClosestEndingMachine$'s availability and then is scheduled. After that, $NextTask$ is also scheduled to $ClosestEndingMachine$ and is removed from $T^{Remaining}$. See lines 16 to 22.

Finally, the end of the schedule is optimized by calling the OptimizeScheduleEnds. This is especially important if the default greedy $NextTask$ selection strategy was used, as it tends to leave long setups to the end. See line 25.

Appendix A.2. Resolution of Setup Overlaps Lazily (ROSOL)

The selection and assignment of $T^{StartingTasks}$ and the selection of $ClosestEndingTask$, $ClosestEndingMachine$ and $NextTask$ is the same as in LOSOS. The only difference here is that there is no data structure for servers. Thus, the $NextSetup$ and $NextTask$ are assigned to machines without considering server availability. See lines 2 to 17.

The problem is now solved without considering servers' constraints. Before checking if it adheres to said constraints, OptimizeScheduleEnds is called to reduce makespan if possible. Calling OptimizeScheduleEnds is beneficial, even if the solution does not adhere to server constraints, because machines' balancing will reduce future collisions when OptimizeScheduleEnds is called again at the end of the algorithm. See line 18.

All machines that are not in $SetMachines$ and currently have a setup scheduled are found and added to set $FreeMachines$. If needed, these machines' current setups can be moved to a later point in the schedule. On the other hand, $SetMachines$ contains machines where the currently executed setup will not move under any circumstances as it has been already confirmed for the execution. $TimeStep$ is updated either by the end of the present task or setup on the machine. Also, if some fixed machine previously executing a setup is finished at current $Time$, it is removed from $SetMachines$. See lines 24 to 36.

If the current number of setups executed is higher than the number of servers, some setups from $FreeMachines$ will have to be moved to a later point in the schedule. The coefficient of move priority is computed based on the machine length compared to the makespan and the length of the currently present setup on the machine. This helps us determine how would the move affect the makespan and waiting times of other machines for free server. See lines 37 to 42.

While free servers are available, the machine with the lowest coefficient is selected and moved to $SetMachines$, fixing its setup execution, thus allocating a server to it. See lines 43 to 47.

When all servers are busy, and there are still setups currently being executed, they are moved to a later point in the solution together with their following tasks and setups on the same machine. The makespan of the solution is updated, and $Time$ is moved by $TimeStep$. See lines 48 to 53.

After resolution of the servers' collisions, the end of the schedule is again optimized. See line 57.

Algorithm 2 Pseudocode of ROSOL algorithm.

function ROSOL SOLVE
  $T^{StartingTasks} \leftarrow$ GENERATESTARTINGTASKS ▷ called function
  for each $M_m \in M$ **do**
    Schedule $M_m \leftarrow T_m^{StartingTasks}$
  end for
  $T^{Remaining} \leftarrow T \setminus \{T^{StartingTasks}\}$
  while $T^{Remaining} \neq \emptyset$ **do**
    $ClosestEndingMachine \leftarrow argmin(End(M))$
    $ClosestEndingTask \leftarrow T_{-1,ClosestEndingMachine}$ ▷ Last task is indexed by "-1".
    $NextTask,\ SetupLength \leftarrow$ SELECTNEXTTASK($ClosestEndingTask$, $T^{Remaining}$)
    ▷ called function
    $Start(NextSetup) \leftarrow End(ClosestEndingTask)$
    $Start(NextTask) \leftarrow Start(NextSetup) + SetupLength$
    Schedule $ClosestEndingMachine \leftarrow NextSetup$
    Schedule $ClosestEndingMachine \leftarrow NextTask$
    $T^{Remaining} \leftarrow T^{Remaining} \setminus \{ClosestEndingTask\}$
  end while
  OPTIMIZESCHEDULEENDS ▷ called function
  $SetMachines \leftarrow \emptyset$ ▷ Machines fixed to execute current setup.
  $Time \leftarrow 0$ ▷ $Time$ represents current time point in solution.
  while $Time < Makespan$ **do** ▷ $Makespan$ represents solution's makespan.
    $TimeStep \leftarrow \infty$ ▷ The amount of time to move.
    $FreeMachines \leftarrow \emptyset$ ▷ Machines with movable current setup.
    for each $M_m \in M$ **do**
      if M_m in $Time$ executes a setup **then**
        $FreeMachines \leftarrow FreeMachines \bigcup \{M_m\}$
        $TimeStep \leftarrow min(TimeStep, End(CurrentSetup_m) - Time)$
        ▷ Currently executed setup given the current Time.
      else
        if $M_m \in SetMachines$ **then**
          $SetMachines \leftarrow SetMachines \setminus \{M_m\}$
        end if
        $TimeStep \leftarrow min(TimeStep, End(CurrentTask_m) - Time)$
        ▷ Currently executed task given the current Time.
      end if
    end for
    if $|FreeMachines| + |SetMachines| > |R|$ **then**
      for each $M_m \in FreeMachines$ **do**
        $SetupLength \leftarrow End(CurrentSetup_m) - Start(CurrentSetup_m)$
        $MachineReserve \leftarrow Makespan - End(M_m)$
        $Coefficient(M_m) \leftarrow MachineReserve + SetupLength$
      end for
      while $|SetMachines| < |R|$ & $|FreeMachines| > 0$ **do**
        $M_k \leftarrow argmin(Coefficient(FreeMachines))$
        $FreeMachines \leftarrow FreeMachines \setminus \{M_k\}$
        $SetMachines \leftarrow SetMachines \bigcup \{M_k\}$
      end while
      for each $M_m \in FreeMachines$ **do**
        Move tasks/setups starts/ends after current Time on M_m by $TimeStep$
      end for
      $Makespan \leftarrow max(End(M))$
    end if
    $Time \leftarrow Time + TimeStep$
  end while
  OPTIMIZESCHEDULEENDS ▷ called function
end function

Appendix B. Large instance runtimes

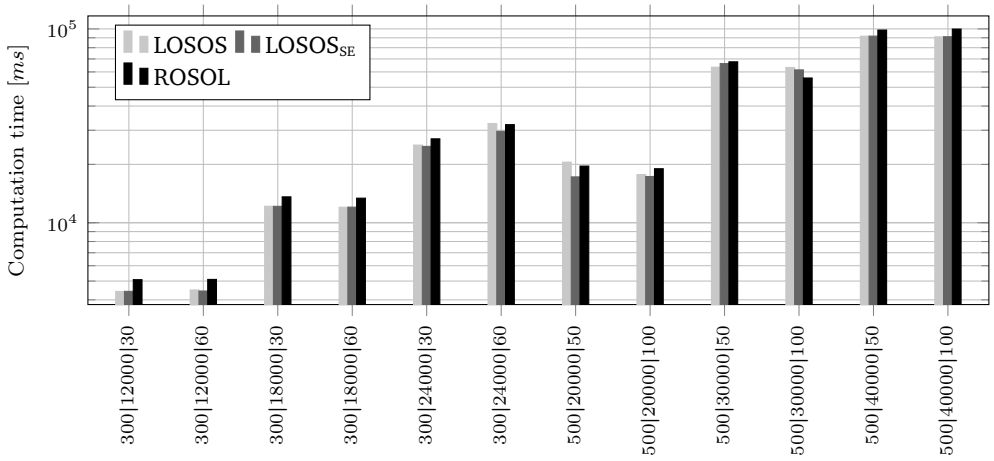

Figure B.5: Runtimes of heuristics without expensive (task coefficient and idleness reduction) improvements. The x-axis labels are in format $m \mid t \mid r$.

Appendix C. Lower bound calculation

The lower bound calculation to assess overall quality of our proposed approaches is following:

- First, $\overline{M}_p = \sum_{i=1}^{t} \frac{p_i}{m}$ is calculated.
- For every task T_j, $z_j = \min_{T_i \in T}(o_{i,j})$, i.e., the shortest setup from any $T_i \in T$ to a given T_j, are calculated and added to multiset Z. Next, m largest z_j are removed from Z, yielding Z'. Then, $\overline{M}_o = \sum_{z \in Z'} \frac{z}{m}$ and $\overline{R}_o = \sum_{z \in Z'} \frac{z}{r}$ are calculated.
- Finally, $lowerbound = max(\overline{M}_p + \overline{M}_o, \overline{R}_o)$. Please note that $\overline{M}_p + \overline{M}_o$ represent smallest possible time that the longest machine will take to execute the schedule, while $\overline{R}_o$ smallest possible time that the longest working server will take to perform the setups in the schedule.

Appendix D. CP scalability

#	m	t	r	$\text{LOSOS}_{\text{SEIC}}$	ROSOL_{SE}	$\text{CP}_{\text{WS-SEIC}}(3600\text{s})$
1	30	600	3	568	548	**521**
2	30	600	6	546	548	**522**
3	30	750	3	676	672	**647**
4	30	750	6	668	672	**647**
5	40	800	4	555	557	**534**
6	40	800	8	560	554	**533**
7	40	1000	4	703	689	**668**
8	40	1000	8	689	689	**668**
9	50	1000	5	565	559	**537**
10	50	1000	10	556	559	**535**
11	50	1250	5	707	710	**678**
12	50	1250	10	697	710	**681**
$\sum$	-	-	-	7490	7467	**7171**
RD	-	-	-	4.45 %	4.13 %	**0.0 %**

Table D.9: The comparison between the warm starts ($\text{LOSOS}_{\text{SEIC}}$ and ROSOL_{SE}) provided to $\text{CP}_{\text{WS-SEIC}}$ and solutions attained by $\text{CP}_{\text{WS-SEIC}}$ in an hour runtime.

References

[1] K.-D. Thoben, S. Wiesner, , T. Wuest, "industrie 4.0" and smart manufacturing – a review of research issues and application examples, International Journal of Automation Technology 11 (1) (2017) 4–16. doi:10.20965/ijat.2017.p0004.

[2] S. Huang, L. Cai, X. Zhang, Parallel dedicated machine scheduling problem with sequence-dependent setups and a single server, Computers & Industrial Engineering 58 (1) (2010) 165–174. doi:https://doi.org/10.1016/j.cie.2009.10.003.
URL https://www.sciencedirect.com/science/article/pii/S0360835209002642

[3] M.-Y. Kim, Y. H. Lee, Mip models and hybrid algorithm for minimizing the makespan of parallel machines scheduling problem with a single server, Computers & Operations Research 39 (11) (2012) 2457–2468. doi:https://doi.org/10.1016/j.cor.2011.12.011.
URL https://www.sciencedirect.com/science/article/pii/S0305054811003662

[4] A. Hamzadayi, G. Yildiz, Modeling and solving static m identical parallel machines scheduling problem with a common server and sequence dependent setup times, Computers & Industrial Engineering 106 (2017) 287–298. doi:https://doi.org/10.1016/j.cie.2017.02.013.
URL https://www.sciencedirect.com/science/article/pii/S0360835217300748

[5] F. Rossi, P. van Beek, T. Walsh, Chapter 1 - introduction, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Vol. 2 of Foundations of Artificial Intelligence, Elsevier, 2006, pp. 3 – 12. doi:https://doi.org/10.1016/S1574-6526(06)80005-2.
URL http://www.sciencedirect.com/science/article/pii/S1574652606800052

[6] P. Laborie, J. Rogerie, P. Shaw, P. Vilím, Ibm ilog cp optimizer for scheduling, Constraints 23 (2) (2018) 210–250. doi:10.1007/s10601-018-9281-x.
URL https://doi.org/10.1007/s10601-018-9281-x

[7] A. P. de Abreu, H. Y. Fuchigami, An efficiency and robustness analysis of warm-start mathematical models for idle and waiting times optimization in the flow shop, Computers & Industrial Engineering 166 (2022) 107976. doi:https://doi.org/10.1016/j.cie.2022.107976.
URL https://www.sciencedirect.com/science/article/pii/S0360835222000468

[8] S. M. Pour, J. H. Drake, L. S. Ejlertsen, K. M. Rasmussen, E. K. Burke, A hybrid constraint programming/mixed integer programming framework for the preventive signaling maintenance crew scheduling problem, European Journal of Operational Research 269 (1) (2018) 341 – 352. doi:https://doi.org/10.1016/j.ejor.2017.08.033.
URL http://www.sciencedirect.com/science/article/pii/S0377221717307646

[9] M. Vlk, A. Novak, Z. Hanzalek, A. Malapert, Non-overlapping sequence-dependent setup scheduling with dedicated tasks, in: G. H. Parlier, F. Liberatore, M. Demange (Eds.), Operations Research

and Enterprise Systems, Springer International Publishing, Cham, 2020, pp. 23–46. doi:https://doi.org/10.1007/978-3-030-37584-3_2.

[10] A. Allahverdi, C. Ng, T. Cheng, M. Y. Kovalyov, A survey of scheduling problems with setup times or costs, European Journal of Operational Research 187 (3) (2008) 985–1032. doi:https://doi.org/10.1016/j.ejor.2006.06.060.
URL https://www.sciencedirect.com/science/article/pii/S0377221706008174

[11] E. Vallada, R. Ruiz, A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times, European Journal of Operational Research 211 (3) (2011) 612–622. doi:https://doi.org/10.1016/j.ejor.2011.01.011.
URL https://www.sciencedirect.com/science/article/pii/S0377221711000142

[12] Y. H. Lee, M. Pinedo, Scheduling jobs on parallel machines with sequence-dependent setup times, European Journal of Operational Research 100 (3) (1997) 464–474. doi:https://doi.org/10.1016/S0377-2217(95)00376-2.
URL https://www.sciencedirect.com/science/article/pii/S0377221795003762

[13] W. T. Lunardi, E. G. Birgin, D. P. Ronconi, H. Voos, Metaheuristics for the online printing shop scheduling problem, European Journal of Operational Research 293 (2) (2021) 419–441. doi:https://doi.org/10.1016/j.ejor.2020.12.021.
URL https://www.sciencedirect.com/science/article/pii/S0377221720310493

[14] W. T. Lunardi, E. G. Birgin, P. Laborie, D. P. Ronconi, H. Voos, Mixed integer linear programming and constraint programming models for the online printing shop scheduling problem, Computers & Operations Research 123 (2020) 105020. doi:https://doi.org/10.1016/j.cor.2020.105020.
URL https://www.sciencedirect.com/science/article/pii/S0305054820301374

[15] G. Rauchecker, G. Schryen, Using high performance computing for unrelated parallel machine scheduling with sequence-dependent setup times: Development and computational evaluation of a parallel branch-and-price algorithm, Computers & Operations Research 104 (2019) 338–357. doi:https://doi.org/10.1016/j.cor.2018.12.020.
URL https://www.sciencedirect.com/science/article/pii/S0305054818303320

[16] A. H. Abdekhodaee, A. Wirth, Scheduling parallel machines with a single server: some solvable cases and heuristics, Computers & Operations Research 29 (3) (2002) 295–315. doi:https://doi.org/10.1016/S0305-0548(00)00074-5.
URL https://www.sciencedirect.com/science/article/pii/S0305054800000745

[17] K. Hasani, S. A. Kravchenko, F. Werner, Block models for scheduling jobs on two parallel machines with a single server, Computers & Operations Research 41 (2014) 94–97. doi:https://doi.org/10.1016/j.cor.2013.08.015.
URL https://www.sciencedirect.com/science/article/pii/S0305054813002219

[18] L. Waszniowski, J. Krákora, Z. Hanzálek, Case study on distributed and fault tolerant system modeling based on timed automata, Journal of Systems and Software 82 (10) (2009) 1678–1694, sI: YAU. doi:https://doi.org/10.1016/j.jss.2009.04.042.
URL https://www.sciencedirect.com/science/article/pii/S0164121209001071

[19] H. Tempelmeier, L. Buschkühl, Dynamic multi-machine lotsizing and sequencing with simultaneous scheduling of a common setup resource, International Journal of Production Economics 113 (1) (2008) 401–412, research and Applications in E-Commerce and Third-Party Logistics Management Special Section on Meta-standards in Operations Management: Cross-disciplinary perspectives. doi:https://doi.org/10.1016/j.ijpe.2007.10.001.
URL https://www.sciencedirect.com/science/article/pii/S092552730700299X

[20] D. Chen, P. Luh, L. Thakur, J. Moreno, Optimization-based manufacturing scheduling with multiple resources, setup requirements, and transfer lots, IIE Transactions 35 (2003) 973–985. doi:10.1080/07408170309342349.

[21] X. Zhao, P. B. Luh, J. Wang, Surrogate gradient algorithm for lagrangian relaxation, Journal of optimization Theory and Applications 100 (3) (1999) 699–712. doi:https://doi.org/10.1023/A:1022646725208.

[22] M. Vlk, A. Novak, Z. Hanzalek, Makespan minimization with sequence-dependent non-overlapping setups, in: Proceedings of the 8th International Conference on Operations Research and Enterprise Systems - Volume 1: ICORES,, INSTICC, SciTePress, 2019, pp. 91–101. doi:10.5220/0007362700910101.

[23] A. Costa, F. Cappadonna, S. Fichera, A hybrid genetic algorithm for job sequencing and worker allocation in parallel unrelated machines with sequence-dependent setup times, The International Journal of Advanced Manufacturing Technology 69 (12 2013). doi:10.1007/s00170-013-5221-5.

[24] J. C. Yepes-Borrero, F. Villa, F. Perea, J. P. Caballero-Villalobos, Grasp algorithm for the unrelated

parallel machine scheduling problem with setup times and additional resources, Expert Systems with Applications 141 (2020) 112959. doi:https://doi.org/10.1016/j.eswa.2019.112959.
URL https://www.sciencedirect.com/science/article/pii/S0957417419306773

[25] L. Fanjul-Peyro, Models and an exact method for the unrelated parallel machine scheduling problem with setups and resources, Expert Systems with Applications: X 5 (2020) 100022. doi:https://doi.org/10.1016/j.eswax.2020.100022.
URL https://www.sciencedirect.com/science/article/pii/S2590188520300019

[26] J.-H. Lee, H.-J. Kim, A heuristic algorithm for identical parallel machine scheduling: splitting jobs, sequence-dependent setup times, and limited setup operators, Flexible Services and Manufacturing Journal 33 (4) (2021) 992–1026. doi:10.1007/s10696-020-09400-9.
URL https://doi.org/10.1007/s10696-020-09400-9

[27] P. Caricato, A. Grieco, A. Arigliano, L. Rondone, Workforce influence on manufacturing machines schedules, The International Journal of Advanced Manufacturing Technology 115 (3) (2021) 915–925. doi:10.1007/s00170-020-06176-y.
URL https://doi.org/10.1007/s00170-020-06176-y

[28] J. C. Yepes-Borrero, F. Perea, R. Ruiz, F. Villa, Bi-objective parallel machine scheduling with additional resources during setups, European Journal of Operational Research 292 (2) (2021) 443–455. doi:https://doi.org/10.1016/j.ejor.2020.10.052.
URL https://www.sciencedirect.com/science/article/pii/S0377221720309450

[29] A. Gnatowski, J. Rudy, R. Idzikowski, On two-machine flow shop scheduling problem with disjoint setups, in: 2020 IEEE 15th International Conference of System of Systems Engineering (SoSE), 2020, pp. 277–282. doi:10.1109/SoSE50414.2020.9130513.

[30] V. Heinz, Supplemental materials, https://gitlab.com/vilem_heinz/cp_heur_paper_evalutation (2022).

[31] N. G. Hall, M. E. Posner, Generating experimental data for computational testing with machine scheduling applications, Operations Research 49 (6) (2001) 854–865. arXiv:https://doi.org/10.1287/opre.49.6.854.10014, doi:10.1287/opre.49.6.854.10014.
URL https://doi.org/10.1287/opre.49.6.854.10014