Bottleneck Identification in Resource-Constrained Project Scheduling via Constraint Relaxation

Lukáš Nedbálek[1,2] and Antonín Novák[2]

[1]*Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic*

[2]*Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Czech Republic*

Keywords: Scheduling, RCPSP, Bottlenecks, Constraint Relaxation.

Abstract: In realistic production scenarios, Advanced Planning and Scheduling (APS) tools often require manual intervention by production planners, as the system works with incomplete information, resulting in suboptimal schedules. Often, the preferable solution is not found just because of the too-restrictive constraints specifying the optimization problem, representing bottlenecks in the schedule. To provide computer-assisted support for decision-making, we aim to automatically identify bottlenecks in the given schedule while linking them to the particular constraints to be relaxed. In this work, we address the problem of reducing the tardiness of a particular project in an obtained schedule in the resource-constrained project scheduling problem by relaxing constraints related to identified bottlenecks. We develop two methods for this purpose. The first method adapts existing approaches from the job shop literature and utilizes them for so-called untargeted relaxations. The second method identifies potential improvements in relaxed versions of the problem and proposes targeted relaxations. Surprisingly, the untargeted relaxations result in improvements comparable to the targeted relaxations.

1 INTRODUCTION

In the modern manufacturing industry, Advanced Planning and Scheduling (APS) tools are used to schedule production automatically. However, not all parameters and information are available to the APS systems in practice. Thus, the solutions obtained with the scheduling tools may not be preferable to the users. This leads to repeated interactions of the production planners with the APS system, adjusting the problem parameters to obtain an acceptable schedule.

We address a common problem in production—reducing the delay of a selected project in an existing production schedule. We achieve this by identifying *manufacturing bottlenecks* in the schedule and related constraints. Bottlenecks represent resources or constraints with the most significant impact on production performance. The algorithm identifies these bottlenecks and suggests appropriate relaxations in the form of a modified schedule. The modified schedule is then offered to the decision maker to compare the improvements, and the changes can then be accepted, rejected, or augmented to suit the current needs.

See an example in Figure 1. In the original schedule, the target project J_8 should preferably have been
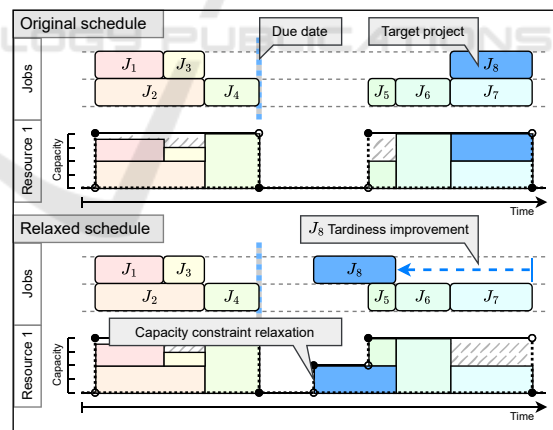

Figure 1: Example of an original and relaxed schedule with 8 jobs and a single resource. The *Jobs* segments show the (overlapping) scheduling of jobs in time. The *Resource 1* segments show the cumulative consumption of the single resource by the scheduled jobs. In the *Relaxed schedule* the *Capacity constraint relaxation* refers to the temporary increase of the resource capacity.

completed earlier, but the resource capacity is insufficient. The target project could be completed earlier by relaxing the capacity constraint, as shown in the relaxed schedule. This relaxation can then be inter-

preted as, e.g., adjusting the start of a shift for a single employee.

The above example is a particularly simple case of the problem; in reality, activities of different projects have complex interactions across multiple resources, making the identification of the constraints to be relaxed a difficult problem. In some sense, this approach is similar to the concept of duality known in continuous optimization (e.g., shadow prices in Linear Programming), which does not apply to discrete optimization problems.

We develop two different methods to address the problem. The first method adapts existing approaches from the literature, which address the problem in simpler scheduling models. The second method utilizes relaxations that focus specifically on the target project. To compare the methods, we present a set of problem instances designed to model the addressed scheduling problem, and we evaluate the two methods using the presented problem instances.

2 RELATED WORK

Following the work of Wang et al. (2016), we focus on *Execution bottleneck machines*, which are bottlenecks in a constructed schedule for a given problem instance, i.e., not for the model of the production but for the specific data. The identified execution bottlenecks may vary between problem instances and their constructed schedules. Focusing on execution bottlenecks aims to improve performance for the specific problem instance, i.e., *case-based relaxation*.

2.1 Bottlenecks in the RCPSP

To the best of our knowledge, no relevant research focuses mainly on identifying bottlenecks in the RCPSP. The closest research on bottlenecks can be found for the Job-Shop problem—scheduling on unit-capacity resources.

Luo et al. (2023) studied how identifying bottleneck machines can guide the scheduling process of a genetic algorithm. In their case study, Arkhipov et al. (2017) proposed a heuristic approach for estimating project makespan and resource load profiles. Those estimations are, in turn, used to identify bottleneck resources. However, the identified bottleneck resources were not addressed further.

2.2 Relaxing the Identified Bottlenecks

Lawrence et al. (1994) studied how identified bottlenecks shift between machines in response to introduc-

ing relaxations to the original problem. They relaxed "short-run" bottlenecks by increasing the capacity of the identified bottleneck resources and then observed whether the bottlenecks shifted to a different machine. The authors observed that while such relaxations are effective at relaxing local bottlenecks, they also increase the "bottleneck shiftiness".

In their study, R. Zhang et al. (2012) addressed the Job-Shop problem by first relaxing its capacity constraints, solving the modified relaxed problem and identifying bottlenecks in its solution. The obtained information was used to guide a proposed simulated annealing algorithm to find a solution to the original problem. Thus, the relaxation served only as an intermediate step toward obtaining a solution, rather than being the desired result.

2.3 Contribution

The specific contributions of this paper are:

- We extend two standard Job-Shop bottleneck identification indicators for application to the RCPSP. We then develop a method utilizing the extended indicators for proposing *untargeted* resource constraint relaxations.

- We develop an approach for proposing *targeted* relaxations specifically for the RCPSP extended with time-variant resource capacities.

3 PROBLEM STATEMENT

3.1 Scheduling Model

We assume the $PSm \mid intree \mid \sum_j w_j T_j$ variant of the RCPSP with several extensions to model the addressed problem.

We define a *problem instance I* as a 4-tuple $(\mathcal{J}, \mathcal{P}, \mathcal{R}, \mathcal{T})$, where $\mathcal{J} = \{1, \ldots, n\}$ is the set of *jobs*, $\mathcal{P}$ is the set of all *precedences* constraints, $\mathcal{R} = \{1, \ldots, m\}$ is the set of *resources*, $\mathcal{T} \in \mathbb{N}$ is the time horizon of the problem instance.

Each job $j \in \mathcal{J}$ has a processing *duration p_j* and a *due date d_j*. A *tardiness weight w_j* defines the penalty for each time period the job is *tardy*, i.e., not completed before its due date. *Preemption* of jobs is not allowed. The order of jobs is constrained with *precedence constraints* $i \rightarrow j$ or $(i, j) \in \mathcal{P}$. We define the *precedence graph $G = (\mathcal{J}, \mathcal{P})$*, which is assumed to be an *inforest*, consisting of a set of connected in-trees.

Jobs are assigned to *resources $\mathcal{R}$* with time-variant renewable *capacities*. The capacity of a resource $k \in \mathcal{R}$ during a time period $t \in \{1, \ldots, \mathcal{T}\}$ is denoted

as $R_k^{(t)}$. We assume the capacities of resources to represent the availability of workers operating the resource. Such capacities can (to some extent) be altered in correspondence to changing the number of operating workers. For a job j, the per-period consumption of a resource k is denoted as r_{jk}. We assume that jobs can simultaneously consume multiple resources. The resource capacity functions $R_k^{(t)}$ are assumed to be periodic with a period of 24. With this, we model *working shifts* for the operating workers (e.g., one, two, or three-shift operations).

The set of *projects* $\mathrm{P} = \{j \in \mathcal{J} \mid \nexists i : j \to i\}$ is the set of roots of the precedence in-trees. A job $j \in \mathrm{P}$ is called a *project*. For a job j, due date $d_j \in \mathbb{N}_0$ is given if $j \in \mathrm{P}$; $+\infty$ otherwise, and tardiness weight $w_j \geq 0$ is specified if $j \in \mathrm{P}$; 0 otherwise, i.e., the tardiness penalty and its weight is applied to the last job of each connected component representing a single project.

3.2 Constraint Programming Formulation

The above scheduling problem can be stated as the following constraint programming model:

$$\min \quad \sum_{j \in \mathcal{J}} w_j T_j \tag{1}$$

$$\text{s.t.} \quad C_i \leq S_j \qquad \forall i \to j \in \mathcal{P} \tag{2}$$

$$\sum_{j \in \mathcal{J}} c_{jk}^{(t)} \leq R_k^{(t)} \ \forall t \in \{1, \ldots, \mathcal{T}\} \ \forall k \in \mathcal{R} \tag{3}$$

$$\text{where} \quad S = (S_1, \ldots, S_n) \in \mathbb{N}_0^n$$
$$C = (S_1 + p_1, \ldots, S_n + p_n)$$
$$T_j = \max(0, C_j - d_j)$$
$$c_{jk}^{(t)} = r_{jk} \text{ if } S_j \leq t < C_j; 0 \text{ otherwise}$$

The expression (1) is the optimization minimization objective — the weighted tardiness of jobs. Equation (2) formulates the precedence constraints. and Equation (3) describes the resource capacity constraints — in every time period, the combined consumption of jobs scheduled during the period cannot exceed any of the resource's capacities.

We assume we have access to a solver capable of solving (1)–(3) in a reasonable time (i.e., to provide computer-assisted decision-making to production planners) through the use of constraint programming solvers, such as CP-SAT or IBM CP Optimizer.

3.3 Constraints Relaxation

Some constraints, such as job precedences, job durations, or resource consumption, are inherent to the problem (i.e., defining technological processes, physical constraints, etc.) and cannot be relaxed. The available capacities of the resources can be modified when they reflect, e.g., the number of the available workforce at the specific stage of the production process.

We consider resource *capacity additions* and *capacity migrations* as the possible relaxations of scheduling constraints (3). Capacity addition is a 4-tuple (k, s, e, c), where over the time periods $\{s, \ldots, e-1\}$ the capacity of the resource k is increased by c. Analogously, capacity migration is a 5-tuple $(k_{\mathrm{from}}, k_{\mathrm{to}}, s, e, c)$, where over the time periods $\{s, \ldots, e-1\}$ the capacity of the resource k_{from} is lowered by c and the capacity of the resource k_{to} is increased by the same amount c. For a modified instance I^*, the sets of all migrations and additions are denoted as $\mathcal{M}^{I^*}$ and $\mathcal{A}^{I^*}$, respectively.

In a real-world production system, migrating capacities can be more cost-effective than adding new capacities. For example, reassigning workers from an underutilized machine to a bottleneck machine is typically less expensive than extending workers' shifts into overtime or planning an entirely new and irregular shift. Therefore, capacity migrations are usually preferred. However, if the required capacity adjustments cannot be achieved through capacity migrations, capacity additions can be utilized.

3.4 General Procedure

The general procedure for solving the presented problem, identifying bottlenecks, relaxing corresponding constraints, and solving the modified problem instance works as follows:

1. Obtain a solution S to the problem instance I.

2. Select a target project $p \in \mathrm{P}$ for tardiness improvement. We consider an improvement to be any non-zero decrease in the project's tardiness.

3. Identify bottlenecks in the solution S to I.

4. Relax constraints corresponding to the identified bottlenecks. To do so, we utilize *capacity migrations* and *capacity additions*, as described in Section 3.3. Such relaxations are captured in a modified problem instance I^*.

5. Find solution S^* to I^*.

6. Evaluate the obtained solution S^*. Specifically, how the introduced relaxations improve the tardiness of the target project.

4 SOLUTION PROCEDURE

We present two algorithms designed for identifying and relaxing bottlenecks in the RCPSP. Both algorithms aim to improve the tardiness of a selected project by introducing relaxations to the capacity constraints. We divide the approaches into two groups — untargeted and targeted relaxations depending on whether they consider the target project when identifying bottlenecks. Thus, untargeted relaxations affect the selected project only indirectly.

4.1 Untargeted Relaxations

In this section, we propose adaptations of existing bottleneck identification indicators from the Job-Shop literature. Utilizing the adapted indicators, we propose the Identification Indicator-based Relaxing Algorithm (IIRA) for untargeted relaxations of capacity constraints in an obtained schedule.

4.1.1 Adapted Identification Indicators

We adapt two existing bottleneck identification indicators. The Machine Utilization Rate (MUR), first utilized as a bottleneck identification indicator by Lawrence et al. (1994), considers the ratio of executed work on a resource to the total time the resource was used. The Average Uninterrupted Active Duration (AUAD), initially proposed by Roser et al. (2001), computes the average length of uninterrupted execution periods, where an uninterrupted execution period is a sequence of scheduled immediately consecutive jobs.

Both identification indicators consider the relationship between the total duration of job executions on a resource and the duration for which the resource is idle. In a Job-Shop scheduling problem, this represents all the available information. In the RCPSP, however, we can utilize different resource capacities and variable resource loads for computing more complex identification indicators.

We propose Machine Resource Utilization Rate (MRUR) as the adaptation of MUR and Average Uninterrupted Active Utilization (AUAU) as the adaptation of AUAD. For resource k, the MRUR is defined as:

$$\text{MRUR}_k \overset{\text{def}}{=} \frac{\sum_{j \in \mathcal{J}} (p_j \cdot r_{jk})}{\sum_{t=1}^{C_{\max}} R_k^{(t)}},$$

where $C_{\max} \overset{\text{def}}{=} \max_{j \in \mathcal{J}} C_j$. For resource k, the AUAU is defined as:

$$\text{AUAU}_k \overset{\text{def}}{=} \frac{\sum_{i=1}^{A_k} \text{PRU}_k^{(i)}}{A_k},$$

where the Period Resource Utilization (PRU) $\text{PRU}_k^{(i)}$ of resource k during the uninterrupted active period i is defined as

$$\text{PRU}_k^{(i)} \overset{\text{def}}{=} \frac{\sum_{j \in \mathcal{J}_k^{UAP(i)}} p_j \cdot r_{jk}}{\sum_{t=a_{k,i}^S}^{a_{k,i}^E} R_k^{(t)}}.$$

For resource $k \in \mathcal{R}$, $(a_{k,1}^S, a_{k,1}^E), \ldots, (a_{k,A_k}^S, a_{k,A_k}^E)$ is the sequence of *uninterrupted active periods*, where $a_{k,i}^S \in \{1, \ldots, \mathcal{T} - 1\}$ denotes the start of the period i and $a_{k,i}^E \in \{a_{ki}^S + 1, \ldots, \mathcal{T}\}$ denotes the end of the period i. An uninterrupted active period is a (maximal) set of jobs scheduled consecutively or in parallel with no idle time on the considered resource during the period. In the formula for $\text{PRU}_k^{(i)}$,

$$\mathcal{J}_k^{UAP(i)} \overset{\text{def}}{=} \{j \in \mathcal{J}_k : a_{k,i}^S \le S_j \le a_{k,i}^E\}$$

is the set of jobs executed on resource k during the uninterrupted active period i, where

$$\mathcal{J}_k = \{j \in \mathcal{J} : r_{jk} > 0\}.$$

4.1.2 Identification Indicator-Based Relaxing Algorithm

The function of Identification Indicator-based Relaxing Algorithm (IIRA) is illustrated in Figure 2 and is formally described in Algorithm 1.

First, the bottleneck resource is identified using a specified bottleneck identification indicator $I \in \{\text{AUAU}, \text{MRUR}\}$ (lines 2 and 3). The granular load of the bottleneck resource (line 4) indicates the resource's utilization over granular periods — granular periods each represent G time periods for efficient computation. Convolution with a chosen kernel function is used to obtain the improvement potential of granular periods (line 5) and periods with the most improvement potential are then selected for capacity increase, relaxing the represented capacity constraints (lines 6 and 7). The convolution "distributes" the local information about the machine load to adjacent periods to estimate which periods to focus on. Finally, a new solution to the modified problem instance is found (line 8), capacity functions are reduced to only contain additional capacities consumed by jobs, and migrations and additions are identified.

4.2 Targeted Relaxations

As an alternative to untargeted relaxations, in this section, we present a method for detecting bottlenecks and relaxing related constraints in the RCPSP which focuses on a specified target project and its tardiness.
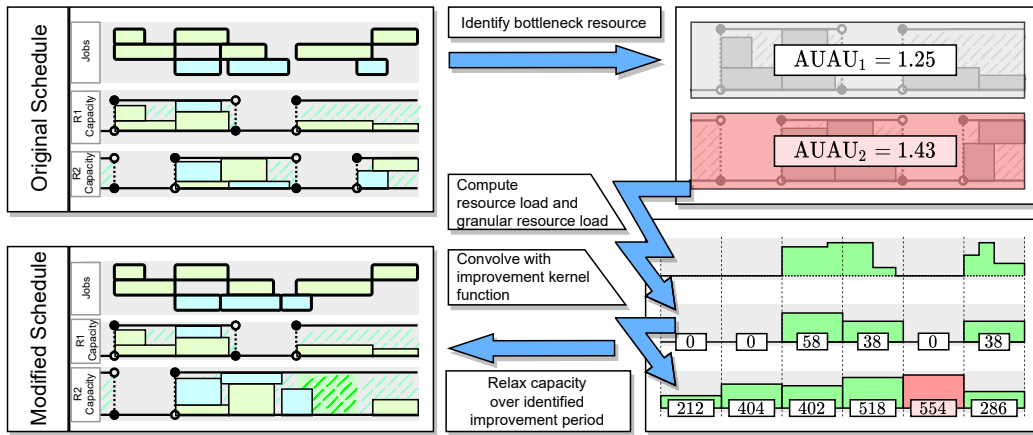
Figure 2: Illustration of the IIRA. Starting with the original schedule, the bottleneck resource is identified using the identification indicator, granular resource load is computed for the resource, utilizing convolution, a specific improvement period is chosen for capacity relaxation, and a modified schedule is obtained.

Algorithm 1: Identification Indicator-based Relaxing Algorithm (IIRA).

Parameters: Identification indicator I, convolution kernel C, granularity G, improvement periods limit P_{max}, iterations limit I_{max}, capacity improvement Δ.

Input: Solution S to problem instance I.

1: **repeat:**
2:    Evaluate S^* using I, obtaining: $I_k \ \forall k \in \mathcal{R}$
3:    Identify bottleneck resource: $k' \leftarrow \text{argmax}_k I_k$
4:    Compute granular resource load $L_{k'}(G)$
5:    Period improvement potential: $\Psi \leftarrow L_{k'}(G) * C$
6:    Select periods with highest $\Psi(i)$: $p_1, \ldots, p_{P_{max}}$
7:    Increase capacity $R'_{k'}$ over the periods by Δ
8:    Find solution S^* to the modified instance I^*
9:    Reduce capacity changes in $R'_1, \ldots, R'_m$
10:    Find migrations $\mathcal{M}^{I^*}$ and additions $\mathcal{A}^{I^*}$
11: **for I_{max} iterations**

Output: Modified instance I^* and its solution S^*, additions $\mathcal{A}^{I^*}$, migrations $\mathcal{M}^{I^*}$.

The proposed Schedule Suffix Interval Relaxing Algorithm is based on finding improvement intervals in partially relaxed versions of the given problem. A small subset of the improvement intervals is then selected, and capacity constraints corresponding to the selected improvement intervals are relaxed. The targeted relaxation aims to identify relaxations specifically for the target project requiring small changes to improve the tardiness of the project.

4.2.1 Preliminaries

To formulate Schedule Suffix Interval Relaxing Algorithm (SSIRA), we state the necessary definitions and the key ideas. First, we define the *suffix-relaxed schedule*, which is a modification of an obtained schedule where the algorithm finds improvement intervals.

Definition 1 (Suffix-relaxed schedule). *Let $S = (S_1, \ldots, S_n)$ be a schedule to a problem instance I. Given a time period $t \in \{1, \ldots, \mathcal{T}\}$, the* suffix-relaxed schedule *for the time period t is given by $\sigma^{(t)} = (\sigma_1^{(t)}, \ldots, \sigma_n^{(t)})$, where*

$$\sigma_j^{(t)} \stackrel{def}{=} \begin{cases} S_j & \text{if } S_j \leq t; \\ \max\left\{\sigma_i^{(t)} + p_i : i \rightarrow j \in \mathcal{P}\right\} & \text{otherwise.} \end{cases}$$

The precedence graph is acyclic; thus, all values of $\sigma_i^{(t)}$ are well-defined. The suffix-relaxed schedule essentially relaxes resource capacity constraints for all jobs that start after the time period t in the original schedule. The main idea is to incrementally observe how jobs that are constrained by insufficient capacities and precedence constraints could be scheduled earlier.

We define the *left-shift closure* as a tool for guiding the search for improvement intervals towards improving the tardiness of the target project. The left-shift closure of a job j defines the set of all jobs that need to be scheduled earlier for the job j to decrease its completion time.

Definition 2 (Left-shift closure). *Let $S = (S_1, \ldots, S_n)$ be a schedule to problem instance I. The* left-shift closure *of a job $j \in \mathcal{J}$ is the set $\mathcal{L}(j) \subseteq \mathcal{J}$, where:*

i) $j \in \mathcal{L}(j)$.

ii) All precedence predecessors immediately preceding in the schedule are included.

iii) All jobs sharing a common resource immediately preceding in the schedule are included.

iv) If j is scheduled exactly at any start of a resource's availability interval, all jobs scheduled at the end of the previous availability interval are included. For a given resource, an availability interval is a sequence of consecutive time periods where the resource's capacity is non-zero.

Condition i) is a trivial base case. Condition ii) states that a immediate precedence predecessor i of the job j (i.e., $i \rightarrow j \in \mathcal{P}$) is included in $\mathcal{L}(j)$ if the jobs are scheduled consecutively, i.e. $C_i = S_j$. Condition iii) involves all jobs scheduled consecutively before the job j, which share at least one required resource. In this case, the consumption of the shared resource by the preceding job can be sufficiently large to prevent the job j from being scheduled earlier. Condition iv) involves jobs at the end of the previous working shift. Assuming sufficient slack in precedence constraints, the job j starts exactly at the start of a working shift because it could not have been scheduled at the end of the previous working shift due to the lack of remaining capacities on its required resources.

4.2.2 Schedule Suffix Interval Relaxing Algorithm

The schematic highlight of SSIRA is displayed in Figure 3 and is formally given in Algorithm 2. First, improvement intervals are identified using the FIND-INTERVALSTORELAX function (line 3). Resource capacities are then relaxed based on these intervals (line 4). Finally, as in the IIRA, a solution to the modified instance is found, capacity functions are reduced, and migrations and additions are identified.

The FINDINTERVALSTORELAX function, described in Algorithm 3, finds improvement intervals in suffix-relaxed schedules and selects the best intervals based on a given ordering. Suffix-relaxed schedules are computed for each time period (line 1), representing all possible job-interval relaxations. The left-shift closure of the target project is computed (line 2). This closure represents the set of jobs considered for improvement. For jobs within the closure, potential improvement intervals are identified (lines 3–6), where the starting time of each interval is the earliest relaxed starting time across all suffix-relaxed schedules (line 5). Finally, a given number of intervals is selected based on a specified sort key (line 7).

5 EXPERIMENTS

We evaluate the performance of an untargeted bottleneck detection method called Identification Indicator-based Relaxing Algorithm (IIRA) and a targeted method called Schedule Suffix Interval Relaxing Al-

---

Algorithm 2: Schedule Suffix Interval Relaxing Algorithm (SSIRA).

Parameters: Iterations limit I_{max}, improvement intervals limit IT_{max}, interval sort key $\mathcal{K}$.
Input: Solution S to instance I, target project p.
1: $I^* \leftarrow I, S^* \leftarrow S$ ▷ Modified instance and solution
2: **repeat:**
3: $\quad \chi_1, \ldots, \chi_{IT_{max}} \leftarrow$ FINDINTERVALSTORELAX
4: $\quad$ Increase capacities $R_1^*, \ldots, R_m^*$ in the intervals
5: $\quad$ Find solution S^* to the modified instance I^*
6: $\quad$ Reduce capacity changes in $R_1^*, \ldots, R_m^*$
7: $\quad$ Find migrations $\mathcal{M}^{I^*}$ and additions $\mathcal{A}^{I^*}$
8: **for I_{max} iterations**
Output: Modified instance I^* and its solution S^*, additions $\mathcal{A}^{I^*}$, migrations $\mathcal{M}^{I^*}$.

Algorithm 3: FindIntervalsToRelax.

Input: Problem instance I, its solution S, improvement intervals limit IT_{max}, interval sort key $\mathcal{K}$, target project p.
1: Compute suffix-relaxed schedules $\sigma^{(1)}, \ldots, \sigma^{(\mathcal{T})}$
2: Compute left-shift closure $\mathcal{L}(p)$
3: $X \leftarrow \emptyset$
4: **for $j \in \mathcal{L}(p)$:**
5: $\quad s \leftarrow \min_t \left\{ \sigma_j^{(t)} : \sigma_j^{(t)} < S_j \right\}$
6: $\quad X \leftarrow X \cup \left\{ (j, s, s + p_j) \right\}$
7: Find first $\chi_1, \ldots, \chi_{IT_{max}}$ from X ordered by $\mathcal{K}$
Output: Improvement intervals $\chi_1, \ldots, \chi_{IT_{max}}$, a set of 3-tuples $(j, s, e) \in \mathcal{J} \times \{1, \ldots, \mathcal{T}\}^2$.

---

gorithm (SSIRA). We first design benchmark instances that model the addressed problem and choose ranges of parameters for each algorithm, creating evaluation parameter sets. Then, we conduct the experiments, make several observations about the outcomes, and discuss the achieved results.

5.1 Setup

We use and modify specific instances from the PSPLIB single-mode instance set. The modifications include splitting the precedence graph to create individual project components, introducing job due dates, introducing time-variable resource capacities, and scaling down job durations and resource consumptions for otherwise infeasible instances. We propose eight problem instance groups, each consisting of five individual instances of similar properties (e.g., precedence graph structure or project due dates).

We use the IBM CP Optimizer for finding optimal solutions. The solver time limit for finding a single solution was set to 10 seconds. Subsequent solving of
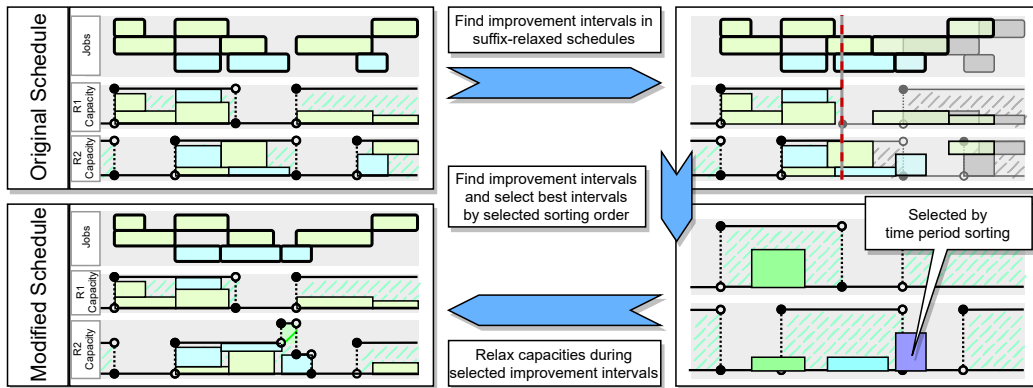
Figure 3: Illustration of the SSIRA. Starting with the original schedule, improvement intervals are found in suffix-relaxed schedules, the best improvement intervals are selected, corresponding capacities are relaxed, and a new schedule is obtained.

Table 1: Improving solutions found for the proposed problem instances. The *Improved* value states the number of instances the algorithm found an improving solution for, the *Unique* value states how many were uniquely found w.r.t. the other algorithm, and the *Best* value states how many were the best-improving solutions.

Algorithm	Criteria	Solutions	% (of 40)
IIRA	Improving	29	**72.5%**
	Unique	0	0%
	Best	22	55%
SSIRA	Improving	35	**87.5%**
	Unique	6	15%
	Best	25	62.5%

modified instances utilizes solver warm-starting. The experiments and the created instances can be found on GitHub[1].

For both algorithms IIRA and SSIRA, all combinations of parameter values are considered, forming a total of 288 combinations for the IIRA and 36 combinations for the SSIRA. Algorithms are evaluated with each combination of values on every problem instance.

The following metrics are computed:

- Tardiness improvement $\Delta T_p \stackrel{\text{def}}{=} T_p - T_p^*$. This metric is also meaningful for the IIRA (i.e., untargeted), where we measure ΔT_p for the target project p.

- Solution difference $\Delta S \stackrel{\text{def}}{=} \sum_{j \in J} \left| C_j - C_j^* \right|$.

5.2 Comparative Results

Table 1 summarizes achieved improvements. The SSIRA found improvements for more instances than

---

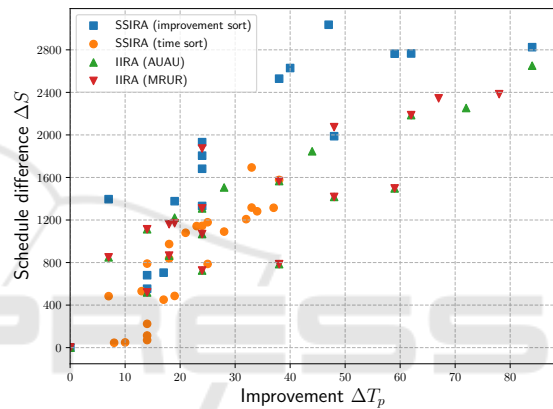[1] https://github.com/Krtiiik/RCPSPsandbox

Figure 4: An example evaluation displays schedule differences versus achieved improvement. Surprisingly, the IIRA often finds better solutions than SSIRA.

the IIRA, moreover, IIRA did not improve any instance which the SSIRA would not improve.

As expected, greater tardiness improvements generally induce larger schedule differences. The SSIRA utilizing the $\mathcal{K}_{\Delta S}$ sort key tends to propose the least favorable solutions in terms of the induced schedule difference and is the most inconsistent in finding improving solutions. The SSIRA with the $\mathcal{K}_t$ sort key finds improving solutions consistently across many instances. However, for some instances, the IIRA is able to find better solutions than the SSIRA. In Figure 4, we present an example of results concerning tardiness improvement related to induced schedule difference showcasing the aforementioned trends.

5.3 Discussion

The SSIRA finds an improvement more often than the IIRA. We believe this is because the SSIRA, unlike the IIRA, utilizes targeted relaxations focusing on the target project. However, the IIRA was still able to

find many improving solutions, sometimes even surpassing the performance of the targeted relaxations proposed by the SSIRA. This is an unexpected result, as the initial assumption was that targeted relaxations would achieve better improvements than general relaxations. It seems to us that targeted relaxations of the SSIRA might be too specific, not providing sufficient slack in the modified constraints and thus making the model too sensitive to minor variations when finding solutions for the relaxed problem. In addition, focusing only on the jobs from the left-shift closure of the target project might be a good heuristic, but it might be too restrictive. Another possibility is that the SSIRA often proposes multiple relaxations simultaneously, incorrectly assuming their independence.

6 CONCLUSION

We addressed the problem of bottleneck identification in production schedules as a computer-aided tool for production planners. First, we formulated an extension of the standard RCPSP as a simplified model of production. Then, we focused on execution-level machine bottlenecks in obtained schedules. Following the identification of such bottlenecks, we proposed constraint relaxations for related resource capacity constraints to find a solution of better quality.

We extended two well-known Job-Shop bottleneck identification indicators for the RCPSP. We proposed the IIRA, utilizing the extended indicators and untargeted relaxations. We also proposed the SSIRA, designed to utilize targeted relaxations.

We observed that the SSIRA is more consistent in finding improving solutions than the IIRA. However, for many instances, the IIRA is able to find great improvements with lower induced schedule differences than those proposed by the SSIRA. Thus, untargeted methods utilizing bottleneck identification indicators appear to be promising in the RCPSP, even for a specific (targeted) project.

Future work might involve modeling the relaxations as an optimization problem to better capture the complex dependencies of the considered constraints, or further exploring the use of bottleneck identification indicators in the RCPSP and its applications in related problems such as 3-dimensional spatial RCPSP (J. Zhang et al., 2024).

ACKNOWLEDGEMENTS

REFERENCES

Arkhipov, D. I., Battaïa, O., and Lazarev, A. A. (2017). Long-term production planning problem: scheduling, makespan estimation and bottleneck analysis. *IFAC-PapersOnLine*, 50(1):7970–7974. 20th IFAC World Congress.

Lawrence, S. R. and Buss, A. H. (1994). Shifting production bottlenecks: causes, cures, and conundrums. *Production and Operations Management*, 3(1):21–37.

Luo, J., Vanhoucke, M., and Coelho, J. (2023). Automated design of priority rules for resource-constrained project scheduling problem using surrogate-assisted genetic programming. *Swarm and Evolutionary Computation*, 81:101339.

Roser, C., Nakano, M., and Tanaka, M. (2001). A practical bottleneck detection method. In *Proceeding of the 2001 Winter Simulation Conference (Cat. No.01CH37304)*, WSC-01. IEEE.

Wang, J.-Q., Chen, J., Zhang, Y., and Huang, G. Q. (2016). Schedule-based execution bottleneck identification in a job shop. *Computers & Industrial Engineering*, 98:308–322.

Zhang, J., Li, L., Demeulemeester, E., and Zhang, H. (2024). A three-dimensional spatial resource-constrained project scheduling problem: Model and heuristic. *European Journal of Operational Research*, 319(3):943–966.

Zhang, R. and Wu, C. (2012). Bottleneck machine identification method based on constraint transformation for job shop scheduling with genetic algorithm. *Information Sciences*, 188:236–252.