Data-driven Single Machine Scheduling Minimizing Weighted Number of Tardy Jobs[⋆] (preprint)

Nikolai Antonov[0009−0002−0156−3561], Přemysl Šůcha[0000−0003−4895−157X], and Mikoláš Janota[0000−0003−3487−784X]

Czech Technical University in Prague, Czech Republic

Abstract. We tackle a single-machine scheduling problem where each job is characterized by weight, duration, due date, and deadline, while the objective is to minimize the weighted number of tardy jobs. The problem is strongly NP-hard and has practical applications in various domains, such as customer service and production planning. The best known exact approach uses a branch-and-bound structure, but its efficiency varies depending on the distribution of job parameters. To address this, we propose a new data-driven heuristic algorithm that considers the parameter distribution and uses machine learning and integer linear programming to improve the optimality gap. The algorithm also guarantees to obtain a feasible solution if it exists. Experimental results show that the proposed approach outperforms the current state-of-the-art heuristic.

Keywords: Data-driven · Heuristic · Machine Learning · Scheduling

1 Introduction

We address an optimization problem with a number of practical applications in everyday life, including parcel delivery, crop harvesting, and customer service [6]. To illustrate the problem's essence, imagine a production line that produces various orders or batches, which we call *jobs*. Assume that the technical process imposes limitations that only one item can be produced at a time, and no interruptions are allowed until a product is completed. The production of every particular good is assigned with two deadlines: soft (also known as *due date*) and hard. Missing the due date is allowed but results in a loss or a penalty. However, failing to meet the hard deadline is strictly unacceptable and may result in catastrophic failures on other production lines or even bring the entire production to a halt. The goal is to manufacture all products before their hard

⋆ This work was supported by the Czech MEYS under the ERC CZ project POSTMAN no. LL1902, by the Grant Agency of the Czech Technical University in Prague, grant No. SGS22/167/OHK3/3T/13 and by the Grant Agency of the Czech Republic under the Project GACR 22-31670S. This article is part of the RICAIP project that has received funding from the EU's Horizon 2020 research and innovation programme under grant agreement No 857306.

deadline while minimizing the total penalty incurred. In scheduling theory, the problem is known as $1|\tilde{d}_i|\sum w_iU_i$ in Graham's notation [5].

The problem is typically addressed in the literature using two common methods. The first involves creating an integer linear programming (ILP) model and handling it with a general solver. Another option is to use the state-of-the-art exact branch-and-bound algorithm by Baptiste et al. [1], developed specifically for this problem. Since both approaches have their limitations, heuristics can be a suitable alternative. Indeed, solving the ILP model may have volatile running times, while a heuristic works quickly and reliably. An efficient heuristic can also improve the branch-and-bound technique by providing tighter bounds for quicker solutions. Although the state-of-the-art approach can handle up to 30,000 jobs within an hour, we observed that the algorithm struggles with smaller instances of 1000–5000 jobs, exceeding a one-hour time limit. The literature also reports specific instances of 250 jobs that the algorithm was unable to solve within the same time limit [7]. This paradox is primarily due to the heuristic algorithm inside the branch-and-bound, which may not provide a tight enough bound on the objective. The fact that a heuristic method can be efficient for some instances but not others inspired us to create an algorithm that will benefit from the distribution of job parameters.

Problem Formulation. Let us have a machine (system) capable of doing some work divided into pieces, which we call *jobs*. The machine follows three basic assumptions: it handles a single job at a time, never interrupts a started job and does not idle, i.e., after processing a job, it immediately moves to the next one until all the assigned jobs are completed. We are given a set of jobs $N = \{1, 2, ..., n\}$ with *durations* p_i, *due dates* d_i and *deadlines* $\tilde{d}_i$ for all $i \in N$. We assume that p_i, d_i, $\tilde{d}_i$ are positive integers and $p_i \le d_i \le \tilde{d}_i$ for all $i \in N$. In addition, each job has a *weight* (or *cost*), which is a positive integer w_i, $i \in N$ that represents how valuable a particular job is. All the jobs are available from the very beginning (time moment 0). Let the jobs be processed according to the permutation π of N and completed at time moments C_i^π, $i \in N$. In scheduling terminology, π is called a *schedule*. We define the set of *early* jobs $E_\pi = \{i \in N \mid C_i^\pi \le d_i\}$ completed before the due date, and the set of *tardy* jobs $T_\pi = \{i \in N \mid d_i < C_i^\pi \le \tilde{d}_i\}$ completed after the due date, but before the deadline. A schedule π is called *feasible*, if $C_i^\pi \le \tilde{d}_i$ for every job $i \in N$, and in terms of introduced sets that is equivalent to $E_\pi \cup T_\pi = N$. Following [1], we assume an equivalent *maximization* problem instead of minimization. Our goal is to maximize the weighted number of early jobs while every job must meet its deadline. That means we want to find a schedule π^* maximizing $f(\pi) = \sum_{i \in E_\pi} w_i$, so that $E_\pi \cup T_\pi = N$.

Literature Review. The problem is known to be strongly NP-hard [13]. The state-of-the-art exact method for solving the problem is the algorithm proposed by Baptiste et al.[1]. As it is mentioned above, the efficiency of this algorithm varies for different types of instances; for example, for one class of specific instances, it faces difficulties solving instances with 250 jobs. This class was studied in [7], where the authors improved the algorithm from [1] such that it can solve 5000 jobs within the same time limit.

The state-of-the-art heuristic for the studied problem is also proposed by Baptiste et al. [1]. Essentially, it is a part of the exact algorithm presented there. It starts by solving a max-profit flow relaxation of the original problem and then determines if a job is early or tardy using ILP and variable fixing techniques. A common rule-based heuristics for solving $1|\tilde{d}_i|\sum w_i U_i$ are *EDF (Earliest Deadline First)*, *EDD (Earliest Due Date first)* and *ATC (Apparent Tardiness Cost)* [12]. Although they are fast and easy to implement, they show a large optimality gap in practice, and only *EDF* can guarantee meeting all job deadlines if a solution exists. According to [1], exact algorithms and relaxation heuristics are the primary sources of improvements for the studied problem. Although meta-heuristic applications have been mentioned in the past for related problems, the literature on this topic is significantly outdated, and therefore we do not discuss them further.

Our approach is based on supervised machine learning (ML) and inspired by the work [2] who have demonstrated the remarkable benefits of applying ML to a wide range of combinatorial optimization problems. However, we are not aware of any ML applications to the studied problem. The closest related work is [3], which addresses the $1||\sum T_i$ problem of minimizing the total violation of due dates. The authors propose to estimate the objective value using LSTM-based neural networks. However, their problem does not assume deadlines, and the approach depends on Lawler's decomposition, which cannot be applied to our case. In addition, a standard LSTM-based neural network requires significant running time. Structured learning is highlighted in [11] for minimizing the completion time of jobs with release times, and [9] addresses online single-machine scheduling using Q-learning techniques. Many applications of reinforcement learning to combinatorial optimization problems are described in the survey by [10]. For a simple setup of supervised machine learning, refer to [8].

Our Contributions. We introduce a novel scheduling heuristic to minimize the weighted number of tardy jobs on a single machine. Our approach consists of three interconnected components that work in synergy to achieve optimal or near-optimal results. The first component leverages machine learning as a decision-making oracle. Unlike traditional methods that rely on a single neural network to predict directly from extracted features, we use two separate networks to estimate different aspects of the problem, achieving more accurate results. Secondly, we refine our predictions through ILP, using an empirically proven job selection strategy. Lastly, we develop a framework based on the fundamental problem properties that can transform any sequence of predictions into a feasible solution if one exists. Our experiments demonstrate that the proposed algorithm outperforms state-of-the-art heuristics in [1] when the input data distribution is known.

2 Solution Approach

With complete information on whether a job is early or tardy in a given instance, the $1|\tilde{d}_i|\sum w_i U_i$ problem can be solved in polynomial time. Indeed, it is sufficient to schedule the jobs in ascending order of D_j $(j \in N)$, which will be a due date $(D_j = d_j)$ for *early* job and a deadline $(D_j = \tilde{d}_j)$ for *tardy* job. More details can be found in [12]. Thus, the main challenge is to decide whether a given job is early or tardy. In this paper, we use supervised machine learning to make such a decision. A typical naive ML approach (sometimes denoted as an end-to-end approach in the literature) decides purely based on the job features. However, in our method, the decision is made differently. Consider the following theorem, presented in [1].

Theorem 1 (Dominance theorem). *Let π^* be an optimal schedule and for jobs i and j holds $w_j > w_i$, $p_j \leq p_i$, $d_j \geq d_i$, $\tilde{d}_j \leq \tilde{d}_i$. Then if the job i is early in π^*, the same holds for j; and if the job j is tardy in π^*, the same holds for i.*

We can see that if there is a certain relation between the parameters of two jobs, then a decision about one of them can be propagated to another. Drawing an analogy, we formulate this in terms of apriori and conditional probabilities. Our goal is to estimate the likelihood $\widetilde{Pr}(j)$ of job j being early, given that we know the probability of job i being early or tardy. Assume that job i has an apriori probability $Pr(i)$ of being early (i_E). Then, it is tardy (i_T) with a probability $1 - Pr(i)$ since earliness and tardiness are mutually exclusive. Let $Pr(j \mid i_E)$ and $Pr(j \mid i_T)$ denote the conditional probability of j being early if i happened to be early or tardy, respectively. We then express the desired probability $\widetilde{Pr}(j)$ as a marginal probability:

$$\widetilde{Pr}(j) = Pr(j \mid i_E)\ Pr(i) + Pr(j \mid i_T)\ (1 - Pr(i)) \tag{1}$$

We utilize two neural network oracles to predict the values on the right side of the equation. The first oracle estimates the apriori probability $Pr(i)$, and the second does the same for conditional probabilities $Pr(j \mid i_E)$ and $Pr(j \mid i_T)$. A rounded average of different marginal estimates $\widetilde{Pr}(j)$, computed with respect to different jobs i, represents the decision about job j.

The proposed machine learning approach has several advantages. First, it takes into account the combinatorial side of the problem by considering the context provided by other jobs rather than just relying on individual job parameters. Secondly, our approach is more balanced as it incorporates both apriori and conditional probability estimates made by two independent oracles. Our observations show that using both oracles positively impacts the final objective value $f(\pi)$, resulting in a 7–10% improvement compared to using only apriori probabilities. Lastly, our decision-making method can be easily combined with Theorem 1: when the theorem can be applied directly, the oracle does not need to be called to estimate conditional probabilities.

Classification procedure. We have introduced the concept of our oracle and discussed its features and benefits. Now, we will explain how our oracle aids decision-making in the problem instance by classifying jobs as early or tardy. We formalize this procedure in Algorithm 1. The first step is to compute apriori estimates $Pr(j)$ for all $j \in N$ using P_{apr} oracle. Next, we randomly select a subset $S \subseteq N$ of k jobs and compute the conditional probabilities for all pairs of jobs $j \in N$ and $i \in S$. Here we check the preconditions of the dominance theorem: if they hold for some jobs i and j (this fact is denoted in the algorithm's pseudocode with I_j^E and I_j^T), then either $Pr(j \mid i_E)$ or $Pr(j \mid i_T)$ are known with certainty; otherwise, both conditional probabilities are computed by P_{cond} oracle. Finally, we compute a sequence of $|S| = k$ marginal probabilities $\widetilde{Pr}(j)$ for each $j \in N$. The predicted class c_j is obtained by rounding off the average value of these marginal probabilities to the closest integer, where 1 represents early, and 0 represents tardy.

Algorithm 1 *Classify* function

```
Require: set of jobs $N = \{1; 2; ...; n\}$;  oracles $P_{apr}$, $P_{cond}$;  $k \in \mathbb{N}$
 1: $Pr(j) \leftarrow P_{apr}(j), \quad j \in N$                          ▷ making apriori estimates
 2: $S \leftarrow Subset(N)$, $|S| = k$                                   ▷ random subset of $k$ jobs
 3: for $j \in N$, $i \in S$ do                                          ▷ making conditional estimates
 4:     $I_j^E \leftarrow (w_j > w_i)$ & $(p_j \le p_i)$ & $(d_j \ge d_i)$ & $(\tilde{d}_j \le \tilde{d}_i)$
 5:     $I_j^T \leftarrow (w_j < w_i)$ & $(p_j \ge p_i)$ & $(d_j \le d_i)$ & $(\tilde{d}_j \ge \tilde{d}_i)$
 6:     $Pr(j \mid i_E) \leftarrow 1$ if $I_j^E$ = "true" else $P_{cond}(j, i_E)$     ▷ predict by oracle or DT
 7:     $Pr(j \mid i_T) \leftarrow 0$ if $I_j^T$ = "true" else $P_{cond}(j, i_T)$
 8: end for
 9: for $j \in N$ do
10:     $\widetilde{Pr}(j) \leftarrow \frac{1}{k} \sum_{i \in S} Pr(j \mid i_E)\ Pr(i) + Pr(j \mid i_T)\ (1 - Pr(i))$   ▷ marginal estimates
11:     $c_j \leftarrow$ "early" if $\widetilde{Pr}(j) \ge 0.5$ else "tardy"          ▷ final decision
12: end for
13: return $c_1, ..., c_n$; $\widetilde{Pr}(1), ..., \widetilde{Pr}(n)$     ▷ predicted classes and marginal estimates
```

Prediction by neural networks. In this section, we provide details about the implementation and training of our neural network oracle, complementing the general perspective presented in the previous sections. At first, we used the Autogluon framework [4], which provides various models for tabular predictions, including KNN, neural networks, LightGBM trees, random forests, and XGBoost. After fitting our data to these different models, we have settled on the neural network model as one of the most accurate. We employed a fully connected multi-layer perceptron with a *Tanh* activation function. This model consists of 8-8-2 neurons in the input-hidden-output layers for estimating apriori probabilities and 17-64-2 neurons for estimating conditional probabilities. We experimented with various configurations, including 3-5 layers with up to 64 neurons each, and tried both *Tanh* and *ReLU* activation functions before settling on this final configuration.

A job j is represented by an eight-dimensional vector of features $h(j)$, which includes its weight w, duration p, due date d, deadline $\tilde{d}$, and four derived features: $\frac{w}{p}$, $w-p$, $\frac{d}{\tilde{d}}$, $\tilde{d}-d$. All features are normalized to [0; 1] interval. The network that estimates the apriori probability takes $h(j)$ as input for a given job j. The network that estimates the conditional probability takes the vector $h(i)$ concatenated with $h(j)-h(i)$, where the subtraction is performed componentwise. The idea is to determine whether we end in a point labeled "early" if we start in $h(i)$ and move along the vector $h(j)$ - $h(i)$. We avoid directly concatenating $h(i)$ and $h(j)$ to prevent the network from acting like an apriori NN. Finally, we concatenate $[h(i), h(j)-h(i)]$ with a boolean value 0 or 1 depending on which probability we are estimating: $P(j \mid i_E)$ or $P(j \mid i_T)$. The optimal solution's components serve as labels. A job is labeled as 1 if it's considered early in the optimal solution and 0 otherwise. While we experimented with more complex features, like histograms based on job weights, duration, due dates, and deadlines, we found that the assembly of the eight features described above generalizes better for larger instances.

We trained a neural network to estimate apriori probabilities using instances of 50 to 5000 jobs. Obtaining labels for most instances of this size was easy using an exact solver. The feature-label pairs were split into training and validation sets with a ratio of 80:20, resulting in approximately 1.7 million pairs. For the second neural network that estimates conditional probabilities, we focused on instances with 1000 jobs only. We used sampling to cover more instances and obtained 25 million feature-label pairs in total. Both models were trained for 20 epochs using the AdamW optimizer with a learning rate of 10^{-3}.

Solving subproblems with ILP. In the previous two subsections, we discussed how our oracle classifies jobs on early or tardy. Suppose we have executed Algorithm 1 on a given problem instance and obtained predicted classes c_j and probabilities $\widetilde{Pr}(j)$ for each $j \in N$. However, relying solely on these predictions for scheduling can be risky, as even a single incorrect prediction may lead to significant deviations from optimal value $f(\pi^*)$. Therefore, we aim to use the predictions differently, focusing on the reduction theorem described in [1]. Suppose a given job $j \in N$ is known to be early $(D_j = d_j)$ or tardy $(D_j = \tilde{d}_j)$ in an optimal solution. A *reduced* problem is formulated on the set of jobs $N' = N \setminus \{j\}$ with the data modified as follows:

$$w'_i = w_i, \;\; p'_i = p_i \;\; (i \in N') \tag{2}$$

$$d'_i = \begin{cases} \min(d_i, D_j - p_j), & if \;\; d_i \le D_j \\ d_i - p_j, & otherwise \end{cases} \qquad (i \in N') \tag{3}$$

$$\tilde{d}'_i = \begin{cases} \min(\tilde{d}_i, D_j - p_j), & if \;\; \tilde{d}_i \le D_j \\ \tilde{d}_i - p_j, & otherwise \end{cases} \qquad (i \in N') \tag{4}$$

Theorem 2 (Reduction theorem). *There exists a feasible schedule π with an early set of jobs E_π if and only if there exists a feasible schedule π' with early set of jobs $E'_\pi = E_\pi \setminus \{j\}$ for the reduced problem.*

We aim to leverage our oracle to reduce the problem to itself, but of a smaller size, removing the jobs with reliable predictions from the original instance. We can apply the reduction theorem to those jobs and solve the obtained subproblem to optimality with some general ILP solver (LINGO, Gurobi, etc.). Combining the reliable predictions of our neural network with an optimal solution to the reduced problem provides updated predictions for the original problem. We first analyze how reliable are the predictions from our neural network. To address this, we conducted an experiment using the training set, as shown in Figure 1.

Let's consider the left sub-figure first. The x-axis displays predicted probabilities of a job being early, with a bin size of a histogram equal to 0.01. The y-axis shows the empirical frequency of prediction errors for each probability, given by the fraction $(\frac{\nu}{r})_q$, where q is the predicted probability, r is the total number of samples with that probability, and ν is the number of incorrectly classified samples. We employed a total of 500,000 samples uniformly distributed with respect to q. The resulting graph shows that the neural network is the most reliable when predicting probabilities close to 0 or 1, and most of the errors occur when the predicted probabilities are close to 0.5. Additionally, the network's error distribution appears to be approximately normal.

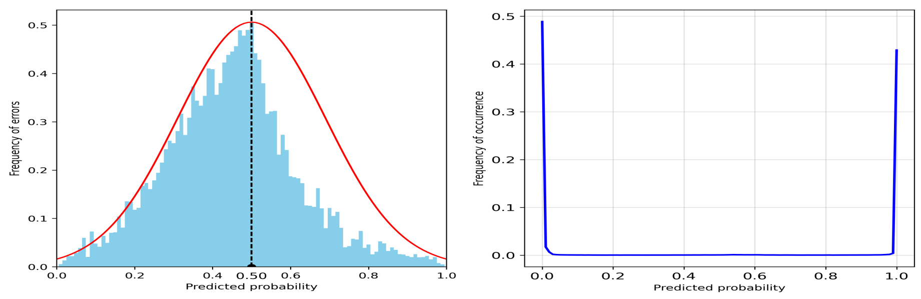

Fig. 1. Frequency of errors (left) and frequency of predicted probabilities (right)

The right-hand side of Fig. 1 shows how frequently our oracle predicts a random job with one or another probability given by the x-axis. We can see that the neural network almost always predicts the jobs having high confidence, e.g., most of $\widetilde{Pr}(j)$ probabilities are close to 0 or 1. Analyzing both graphs, we observe that the network achieves the lowest error rates for predictions with probabilities close to 0 or 1, which also comprise the majority of all predictions. This is a positive outcome, indicating that we can trust the network when it makes such predictions. However, the error rate increases as the predicted probability approaches 0.5, demonstrating that the network is more error-prone in this range.

Algorithm 2 *Update* function

```
Require: set of jobs $N = \{1; 2; ...; n\}$;  $\gamma \in \mathbb{N}$ $(0 \leq \gamma \leq n)$
Require: jobs predicted classes $c_1, ..., c_n$; predicted probabilities $Pr(j)$, ..., $Pr(j)$
 1: $(j_1, ..., j_n) \leftarrow Sort(N, |Pr(j) - 0.5|)$,  $j \in N$      ▷ order jobs by $|Pr(j) - 0.5|$ asc.
 2: $N' = N \setminus \{j_{\gamma+1}, ..., j_n\}$                        ▷ reduce the original instance
 3: $(s_1, ..., s_\gamma) \leftarrow ILP(N',\ time \leq 60s)$            ▷ try to solve reduced problem by ILP
 4: $(c_{j_1}, ..., c_{j_\gamma}) \leftarrow (s_1, ..., s_\gamma)$ if $\neq$ ILP solution exists
 5: return $c_1, ..., c_n$                         ▷ update predicted classes if a solution was found
```

The ideas outlined above are formalized in Algorithm 2, which we refer to as the *Update* function. At the start of the algorithm, we choose the number of jobs γ to be solved by a general ILP solver. This value can be arbitrarily chosen between 0 and n. Assuming that the neural network has just returned the predicted classes c_j and predicted probabilities $Pr(j)$ for each $j \in N$ (as described in Algorithm 1), we begin by sorting the jobs in ascending order of the criterion $|Pr(j) - 0.5|$, $j \in N$; this rearranges the jobs as $j_1, ..., j_n$ (line 1). Next, we apply the reduction theorem (line 2), removing jobs $j_{\gamma+1}, ..., j_n$ (which are predicted closer to 0 or 1) and keeping the remaining jobs to be solved by ILP. We then solve the reduced instance by ILP with a time limit of 60 seconds (line 3). If the solution $s_1, ..., s_\gamma$ was found, it replaces the corresponding predictions made by the neural network (line 4). Otherwise, we keep all the predictions made by the neural network unchanged.

Scheduling Algorithm. Assume we have executed Algorithm 1 followed by Algorithm 2 and thus obtained the predicted classes $c_1, ..., c_n$. This sequence of predictions does not necessarily lead to a feasible schedule, and the final step is to construct one based on the predictions we have. Further on, we use the fact that a given problem is feasible *if and only if* scheduling jobs in ascending order of their deadlines yields a feasible solution [12]. We refer to this check as the *EDF check* (*Earliest Deadline First check*).

Algorithm 3 formalizes the scheduling of classified jobs. First, we check if the instance is feasible. If it holds, we sort the jobs based on values D_i $(i \in N)$, which is a due date if a job is predicted as early and the deadline otherwise. This results in a permutation π, where the job with the smallest D value stands in the first (leftmost) position, and the job with the largest D value is in the last (rightmost) place. We introduce a cursor m and start with the first job j in permutation π. If j is predicted as tardy, we schedule it immediately and move to the next job (lines 8, 11, 14–16). If j is predicted as early, we perform an *EDF* check to determine if we can schedule the remaining unscheduled jobs (line 9). If the check passes, we schedule j and move to the next job (lines 14–16). If the check fails, we do not schedule j. Instead, we change the predicted class c_j to tardy, update the sorting key of j to deadline $(D_j = \tilde{d}_j)$ and push j to a new position in π such that the permutation is sorted again (lines 17–19). We repeat the algorithm steps until all jobs are scheduled. In the end, the cursor stands to the right of the last job in π, which is the output schedule.

Algorithm 3 Scheduling algorithm

```
Require: set of jobs $N = \{1; 2; ...; n\}$;  predicted classes $c_1, ..., c_n$
 1: return $\emptyset$ if $EDF(N) =$ "infeasible"
 2: $D_i \leftarrow d_i$ if $c_i =$ "early" else $\tilde{d}_i$  $(i \in N)$
 3: $\pi \leftarrow Sort(N, D_j)$                        ▷ jobs ordered by $D_i$ $(i \in N)$ ascending
 4: $S \leftarrow \emptyset$                                  ▷ a set of scheduled jobs $S$
 5: $m \leftarrow 1$                                          ▷ a cursor $m$
 6: while $m \leq n$ do
 7:     $j \leftarrow \pi(m)$                             ▷ consider the $m$-th job $j$ from $\pi$
 8:     if $c_j =$ "early" then                          ▷ if it is predicted as early
 9:         $\alpha = EDF(N \setminus (S \cup \{j\}))$          ▷ could we schedule the rest by $EDF$
10:         $schedNow \leftarrow true$ if $\alpha =$ "feasible" else $false$
11:     else
12:         $schedNow \leftarrow true$
13:     end if
14:     if $schedNow$ then          ▷ schedule if it's early and passes $EDF$ or if it's tardy
15:         $S \leftarrow S \cup \{j\}$
16:         $m \leftarrow m + 1$
17:     else                                         ▷ otherwise, put $j$ further in $\pi$
18:         $D_j \leftarrow \tilde{d}_j$
19:         $\pi \leftarrow Push(j, \pi)$             ▷ a new order of jobs where $j$ is placed by $D_j = \tilde{d}_j$
20:     end if
21: end while
22: return $\pi$
```

Proposition 1. *Algorithm 3 halts and produces a feasible schedule if one exists.*

Proof. The algorithm terminates after at most $2n$ steps because on each step a job is either scheduled immediately or forced to become tardy and will be scheduled when the cursor reaches it the second time.

Assume that we are given a feasible instance. To prove that the algorithm always produces a feasible schedule, we need to show that scheduling a job j allows us to schedule the remaining unscheduled jobs without violating their deadlines. There are three mutually exclusive cases:

Case 1. A job j has an early predicted class and passes the *EDF* check. In this case, the *EDF* check confirms that scheduling the remaining jobs after j will not violate any deadlines. Thus, scheduling j preserves the ability to construct a feasible schedule.

Case 2. A job j has an early predicted class but fails the *EDF* check. In this case, j is not scheduled at this moment, and only the permutation π can change. So, if there was an opportunity to construct a feasible schedule, it would remain.

Case 3. A job j has a tardy predicted class. Here we make two observations. First, the jobs in π are always kept sorted during the algorithm, so the sorting key

D_j of job j is always the smallest value of D among the remaining unscheduled jobs. Second, since j has a tardy predicted class, $D_j = \tilde{d}_j$. Therefore, scheduling j works as the very first step of scheduling all the remaining jobs by the *EDF* rule. It preserves the opportunity to construct a feasible schedule, as the remaining unscheduled jobs can still be scheduled by running the *EDF* until the end. This completes the proof.

3 Experimental Results

Table 1. Comparison of optimality gaps and the numbers of optimal solutions

(0.1-3)	Optimal solutions (·/100)				Avg optimality gap, %			
n	Ours	Bapt.	Rand	Early	Ours	Bapt.	Rand	Early
500	**71**	22	0	0	0.0684	**0.0264**	66.888	73.037
1000	**74**	13	0	0	0.0199	**0.0127**	65.857	73.574
2000	**70**	27	0	0	0.0091	**0.0050**	66.775	73.462
3000	**60**	27	0	0	0.0064	**0.0037**	66.779	74.042
4000	**61**	32	0	0	0.0039	**0.0025**	66.625	73.808
5000	**56**	37	0	0	0.0041	**0.0011**	66.487	73.543
(0.1-7)	Optimal solutions (·/100)				Avg optimality gap, %			
n	Ours	Bapt.	Rand	Early	Ours	Bapt.	Rand	Early
500	**86**	20	0	0	0.3472	**0.0167**	38.097	59.836
1000	**92**	19	0	0	**0.0026**	0.0077	38.048	60.528
2000	**99**	17	0	0	**0.0001**	0.0031	37.653	60.512
3000	**76**	20	0	0	**0.0005**	0.0020	37.790	60.316
4000	**57**	11	0	0	**0.0008**	0.0014	37.601	60.360
5000	**63**	26	0	0	**0.0008**	0.0009	37.602	60.146
(0.3-5)	Optimal solutions (·/100)				Avg optimality gap, %			
n	Ours	Bapt.	Rand	Early	Ours	Bapt.	Rand	Early
500	**88**	30	0	0	0.3040	**0.0162**	40.957	49.378
1000	**94**	22	0	0	**0.0012**	0.0072	40.730	49.160
2000	**93**	35	0	0	**0.0005**	0.0027	40.861	49.212
3000	**92**	35	0	0	**0.0009**	0.0019	41.226	49.321
4000	**83**	50	0	0	**0.0003**	0.0013	40.884	49.460
5000	**82**	52	0	0	**0.0003**	0.0007	40.976	49.354
(0.3-7)	Optimal solutions (·/100)				Avg optimality gap, %			
n	Ours	Bapt.	Rand	Early	Ours	Bapt.	Rand	Early
500	**93**	23	0	0	**0.0033**	0.0117	33.365	40.763
1000	**95**	27	0	0	**0.0011**	0.0055	33.756	41.085
2000	**94**	28	0	0	**0.0007**	0.0035	33.611	40.637
3000	**94**	42	0	0	**0.0007**	0.0016	33.643	40.353
4000	**85**	45	0	0	**0.0005**	0.0009	33.656	40.416
5000	**79**	44	0	0	**0.0003**	0.0007	33.670	40.526
(0.5-7)	Optimal solutions (·/100)				Avg optimality gap, %			
n	Ours	Bapt.	Rand	Early	Ours	Bapt.	Rand	Early
500	**90**	21	0	0	**0.0018**	0.0133	31.994	30.422
1000	**99**	32	0	0	**0.0001**	0.0036	32.210	30.134
2000	**96**	42	0	0	**0.0010**	0.0018	32.240	30.339
3000	**88**	46	0	0	**0.0001**	0.0011	32.092	30.273
4000	**87**	59	0	0	**0.0001**	0.0006	32.255	30.338
5000	**85**	56	0	0	**0.0001**	0.0003	32.122	30.473

To ensure a fair comparison with [1], we use their instance generation method, where weights and durations are random natural numbers uniformly distributed on the interval [1, 100]. We also use the same distribution of due dates, which are

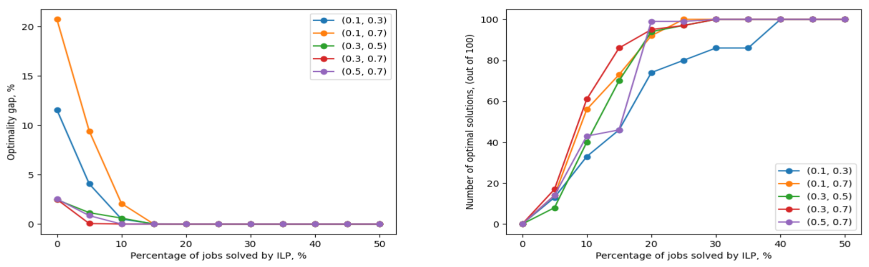

Fig. 2. Impact of γ on the optimality gap and number of optimal solutions

random numbers between $u \sum_{i=1}^{n} p_i$ and $v \sum_{i=1}^{n} p_i$, where (u, v) are selected from the set $(0.1, 0.3), (0.1, 0.7), (0.3, 0.5), (0.3, 0.7), (0.5, 0.7)$. Our implementation of the proposed algorithm is in Python, and we tested it in Google Colab. Both the code and data are available at https://github.com/CTU-IIG/EPIA.

We compare the results of our approach with those of Baptiste et al.'s state-of-the-art heuristic and two other methods that are identical to ours but use different oracles: in the first method (Rand), jobs are predicted to be early or tardy randomly with 0.5 probability; in the second approach (Early), every job is predicted to be early with probability 1. We use two evaluation criteria: optimality gap and the number of optimal solutions achieved out of 100 instances. The optimality gap is the ratio $\frac{f(\pi^*) - f(\pi)}{f(\pi^*)} \cdot 100\%$, where π and π^* represent the constructed and optimal schedules, respectively. We first conduct an experiment on our algorithm alone to demonstrate how the optimality gap and the number of optimal solutions change when we increase the fraction of jobs solved by the ILP, i.e., the γ parameter in Algorithm 2. The results are presented in Fig. 2. The instance size is fixed to 1000 jobs, and the classification of each job in a given instance uses $k = 500$ jobs to estimate conditional probabilities.

Table 1 presents a comparison of our approach with the other heuristics. Our algorithm significantly outperforms them in terms of the number of optimal solutions, always achieving more than half, with a maximum of 99 out of 100. For most distributions and instance sizes, our approach also demonstrates the best optimality gap. However, we should note that Baptiste's heuristic is still superior in terms of running time: 1 second for 500–3000 jobs, 1.5 seconds for 4000 jobs, and 3 seconds for 5000 jobs. The respective running times of our algorithm are 2–15, 19, and 28 seconds; the same for Rand and Early approaches. Finally, we attempted to execute the simple rule-based heuristics *EDD*, *ATC*, and *EDF*. However, the first two did not produce feasible solutions for any instance, and the optimality gap of *EDF* is similar to the random oracle and equals 64–66%.

4 Conclusion

We have proposed a novel heuristic algorithm that employs a combination of machine learning and ILP to minimize the weighted number of tardy jobs on a single machine. Our approach guarantees a feasible solution and outperforms the current state-of-the-art heuristic by considering the distribution of the parameters. Our experiments demonstrate promising results, including a high percentage of optimal solutions and a low optimality gap, indicating the efficiency of our approach in handling practically-sized instances.

References

1. Baptiste, P., Croce, F.D., Grosso, A., T'kindt, V.: Sequencing a single machine with due dates and deadlines: an ILP-based approach to solve very large instances. J. Sched. **13**(1), 39–47 (2010)
2. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: A methodological tour d'horizon. Eur. J. Oper. Res. **290**(2), 405–421 (2021)
3. Bouška, M., Šůcha, P., Novák, A., Hanzálek, Z.: Deep learning-driven scheduling algorithm for a single machine problem minimizing the total tardiness. European Journal of Operational Research (2022)
4. Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., Smola, A.J.: AutoGluon-tabular: Robust and accurate AutoML for structured data. CoRR **abs/2003.06505** (2020)
5. Graham, R., Lawler, E., Lenstra, J., Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Hammer, P., Johnson, E., Korte, B. (eds.) Discrete Optimization II, Annals of Discrete Mathematics, vol. 5, pp. 287–326. Elsevier (1979)
6. Hariri, A.M.A., Potts, C.N.: Single machine scheduling with deadlines to minimize the weighted number of tardy jobs. Management Science **40**(12), 1712–1719 (1994)
7. Hejl, L., Šůcha, P., Novák, A., Hanzálek, Z.: Minimizing the weighted number of tardy jobs on a single machine: Strongly correlated instances. Eur. J. Oper. Res. **298**(2), 413–424 (2022)
8. Karimi-Mamaghan, M., Mohammadi, M., Meyer, P., Karimi-Mamaghan, A.M., Talbi, E.G.: Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. European Journal of Operational Research **296**(2), 393–422 (2022)
9. Li, Y., Fadda, E., Manerba, D., Tadei, R., Terzo, O.: Reinforcement learning algorithms for online single-machine scheduling. In: 2020 15th Conference on Computer Science and Information Systems (FedCSIS). pp. 277–283 (2020)
10. Mazyavkina, N., Sviridov, S., Ivanov, S., Burnaev, E.: Reinforcement learning for combinatorial optimization: A survey. Comp. & Op. Research **134** (2021)
11. Parmentier, A., T'Kindt, V.: Structured learning based heuristics to solve the single machine scheduling problem with release times and sum of completion times. European Journal of Operational Research (2022)
12. Pinedo, M.L.: Scheduling. Theory, Algorithms, and Systems. Springer New York, NY, 233 Spring St, New York, NY USA (2012)
13. Yuan, J.: Unary NP-hardness of minimizing the number of tardy jobs with deadlines. J. Sched. **20**(2), 211–218 (2017)