Tonnius, Annika; Martin, Robert J.

**Article**

# Relaxation and self-supervised machine learning for the hybrid flow shop assignment problem

Logistics Research

**Provided in Cooperation with:**
Bundesvereinigung Logistik (BVL) e.V., Bremen

*Suggested Citation:* Tonnius, Annika; Martin, Robert J. (2023) : Relaxation and self-supervised machine learning for the hybrid flow shop assignment problem, Logistics Research, ISSN 1865-0368, Bundesvereinigung Logistik (BVL), Bremen, Vol. 16, Iss. 1, pp. 1-16, https://doi.org/10.23773/2023_6

This Version is available at:
https://hdl.handle.net/10419/297210

Logistics Research (2023) 16:6
DOI_10.23773/2023_6

# Relaxation and self-supervised machine learning for the hybrid flow shop assignment problem

A. Tonnius[1] and R. J. Martin[2,1]

Received: 31 May 2022 / Accepted: 9 March 2023 / Published online: 28 April 2023


## ABSTRACT

We present an approximation method for the hybrid flow shop scheduling problem based on relaxation and machine learning techniques. Our model combines a suitable relaxation of the objective function to a continuous solution space with a self-supervised learning method for neural networks, which does not require any labeled training data. Thereby, we avoid the pre-computation of exact solutions, which is generally not feasible for NP-hard problems such as hybrid flow shop scheduling. In terms of computational effort during the decision process, our approach outperforms other methods with similar approximation accuracy, which suggests that the considered technique of self-supervised learning is well suited for high-performance applications involving the approximate optimization of discrete NP-hard problems.

KEYWORDS: scheduling · hybrid flow shop · flowtime · makespan · relaxation · neural networks · self-supervised learning

✉ Annika Tonnius[1], Corresponding author
email: annika.tonnius@uni-due.de

Robert J. Martin[2,1]

[1] Faculty of Engineering, University of Duisburg-Essen, Friedrich-Ebert-Str. 12, 47119 Duisburg, Germany

[2] Chair for Nonlinear Analysis and Modeling, Faculty of Mathematics, University of Duisburg-Essen, Thea-Leymann-Str. 9, 45127 Essen, Germany

This article is part of a focus collection on "Dynamics in Logistics – Models and Algorithms for Optimisation, Planning and Control."

## 1 INTRODUCTION

In logistics, many classical tasks of finding optimal routes or schedules lead to NP-hard problems. While countless exact approaches to such problems have been discussed in the literature, it is often more practical to employ efficient algorithms which provide only an *approximate* solution, but require significantly less computational effort. In general, the choice of an approximation algorithm requires a compromise between the accuracy of the solution and computational performance.

The *hybrid flow shop scheduling problem*, in particular, is known to be NP-hard even in seemingly very simple special cases [8]. Both exact and approximate solution approaches to this problem have been the subject of extensive research [23], including classical methods such as linear programming [19, 33] and genetic algorithms [16, 32] as well as supervised and reinforcement machine learning [34, 6]. There remains, however, the need for highly performant optimization methods for the hybrid flow shop [33].

### 1.1 Applications of neural networks to NP-hard problems

There are several major obstacles for the application of supervised machine learning techniques to NP-hard problems such as the hybrid flow shop. Most crucially, applications of supervised learning generally require known *labels* – in this case, the actual solutions to the problem – for a large amount of input parameters to be used during the training process. Since computing these labels would not be feasible for NP-hard problems, approximate solutions acquired by other means (e.g. heuristics or simulations) have often been used as training labels instead [26, 29, 13]. However, this approach still requires a high computational effort for the generation of sufficiently large training datasets, especially since the quality of the training data labels constitutes a lower bound for the prediction quality of any machine learning method. Moreover, since different schedules can often correspond to identical

objective function values, solutions to scheduling tasks and related NP-hard problems are generally not unique, which poses additional challenges to classical supervised machine learning.

Combinatorial problems are, in general, also not directly susceptible to machine learning approaches. In an extensive overview of the topic, Bengio et al. [3] point out the need for appropriate *relaxations* of the combinatorial parameters in order to obtain feasible solutions. In addition to the direct prediction of optimal solutions via relaxation, machine learning has also been integrated into combinatorial models in order to identify objective functions or the combinatorial constraints themselves [18].

Here, we will follow a relaxation-based approach in order to optimize the loss of a hybrid flow shop problem via neural networks. In order to avoid the problem of insufficient training data, we consider a method for training the network by directly employing the objective function we aim to minimize (e.g. the makespan or the flowtime, cf. Section 2.3) as the loss function used during the training process. This approach does not require any exact labels or ground truths; instead, for each input $x$, the loss $L(x,\widehat{y})$ corresponding to the output $\widehat{y}$ predicted by the neural network is based directly on a relaxed variant of the objective function as described in Section 3.2. A suitable training algorithm (e.g. Adam) is then applied with the intention of minimizing the loss – and thus the objective function – on the training dataset, as shown in Fig. 1.

Similar approaches have previously been applied not only to NP-hard problems [20], but to a variety of tasks such as depth estimation from planar image data [7] or object recognition and tracking [25]. More generally, machine learning methods based on a problem-specific loss function instead of classically labeled training data can be characterized as *weakly supervised learning*. Related approaches, known as *self-* and *semi-supervised learning*, have previously been used in natural language processing [17] and object recognition [28, 14, 12] as well as for noise reduction via auto-encoders [24]. *Physics-informed neural networks* [21], which recently have been the subject of extensive research [2, 30, 4], also utilize a highly specific loss function for modeling and training purposes.

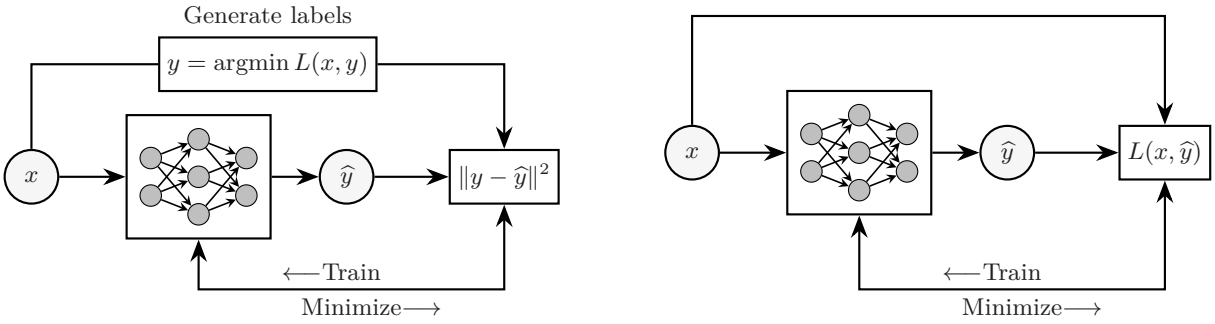
### 1.2 Overview

In the following, we will first provide a formal description of the permutation hybrid flow shop, with a focus on the machine assignment problem and the specific constraints imposed by a fixed job order. Section 3 then describes the relaxation of the problem and the objective function as well as additional penalty terms used during the learning process, according to the training procedure for neural networks as presented in Section 4 (and indicated in Fig. 1). Finally, in Section 5, we provide some numerical examples obtained from applying a trained neural network to randomly generated processing times for a simple three-stage setup to demonstrate the general viability of the approach.

## 2 THE HYBRID FLOW SHOP PROBLEM

Hybrid flow shops (also referred to as *flexible flow shops, multiprocessor flow shops* or *flexible flow lines*) are a generalization of both flow shops and parallel machine environments. The model was originally developed for production planning problems in manufacturing plants, where it is used to describe the production of different kinds of goods [23]. However, the HFS problem is also applicable to a wide variety of other fields, including port logistics [15, 27] or the allocation of requests to multi-stage computer centers [1]. For illustration, we will mostly refer to its original application in manufacturing.

The structure of a production plant following the HFS scheme is shown in Fig. 2. The plant compromises different stages $i = 1, \ldots, I$ in series, each of which contains a subset $\mathcal{M}_i$ of $M_i$ machines. Every *job* $j$ has to pass each stage in sequence and, on each stage i, needs to be processed by exactly one machine $m \in \mathcal{M}_i$.




(a) Supervised learning

(b) Self-supervised learning

*Figure 1: In contrast to classical supervised learning, the approach of self-supervised learning does not require the extensive precomputation of minimizers of the training data.*

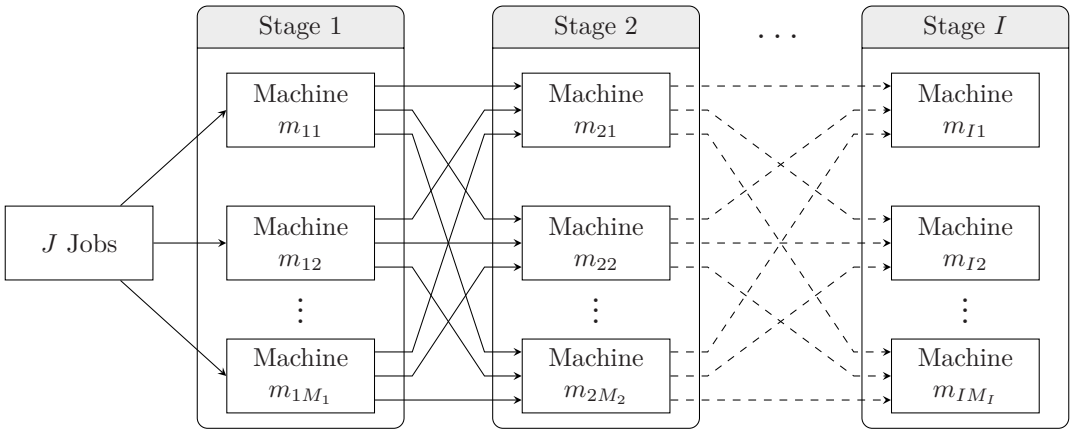



*Figure 2: Illustration of the hybrid flow shop scheduling problem with jobs $j = 1 \ldots, J$ and stages $i = 1, \ldots, I$ with $M_i$ machines on stage $i$.*

Following the definition of Ruiz et al. [23], to be called a hybrid flow shop, the problem configuration must include at least two stages, and at least one of the stages must possess more than one machine. Each job is assigned a *processing time* on each machine. In the case of *unrelated machines*, which we will consider in the following, the processing times might differ both between jobs on the same machine and between different machines on the same stage.

There are a number of constraints which must be satisfied by any *schedule*, i.e. any assignment of jobs to machines and starting times:

- on each stage, each job is assigned to exactly one machine,
- each job must be processed completely on one stage before proceeding to the next,
- a machine must fully process one job before beginning with the processing of another.

More specifically, we will focus on the *permutation* flow shop in the following. For this problem, the jobs must be processed in a specific *order* without overtaking one another, resulting in two additional constraints: if job $j$ has higher priority than job $k$, then

- no machine can begin processing job $k$ before processing of job $j$ has begun on the same stage,
- no machine can finish processing job $k$ before processing of job $j$ has finished on the same stage.

In particular, the last job in the given order will always be (among the) last to fully finish processing. Due to these two additional constraints, assuming no unnecessary gaps during the processing on any stage, the complete schedule is fully determined by the order of the jobs and the assignments of jobs to machines. An example of such a schedule is shown in Fig. 3.

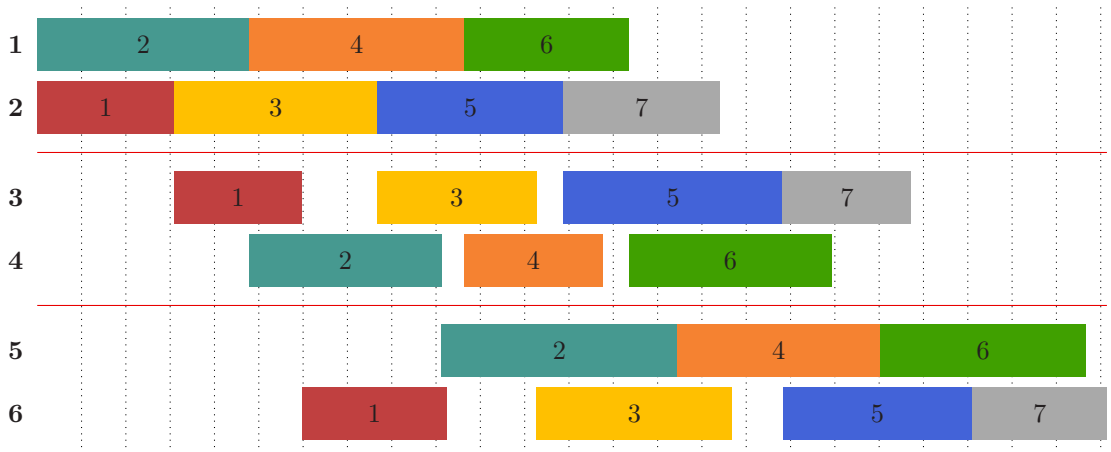



*Figure 3: An optimal schedule for a given job order and given processing times with $J = 7$ jobs, $M = 6$ machines and $I = 3$ stages with two machines each.*

For the permutation hybrid flow shop, the minimization problem now consists of finding the optimal schedule – i.e. the optimal job order and assignments – with respect to a specific objective function which measures the total processing time. Most commonly, either the *makespan* or the *flowtime* is chosen as the objective (cf. Section 2.3).

### 2.1 Divide and conquer

The permutation flow shop problem can be divided into two subproblems:

1. determine the best sequence (permutation) in which the jobs are processed,
2. determine the machine on each stage on which a job is processed (machine assignments).

Of course, the two subproblems are mutually dependent; changing the job order might account for a change in optimal assignments of the jobs to the machines and vice versa. For an approximate solution, however, it has previously been demonstrated that, following a classical *divide-et-impera* strategy, these two problems can indeed be addressed separately [34]. By solving the subproblems individually and combining the results to a solution of the global problem, the solution space can be reduced significantly [33].

For the first subproblem, i.e. for determining the job order, an efficient machine-learning-based approach has already been established [34]. We will therefore focus on the second subproblem of finding optimal machine assignments for jobs in a given fixed order. Note that a solution to this subproblem is also directly applicable by itself, for example in cases where a strict prioritization of jobs – such as a *first-in-first-out* (FIFO) policy – needs to be respected [1].

### 2.2 Notation

In the following, we will assume that the following parameters are fixed as part of the problem specification:

| | |
|---|---|
| $J$ | (number of jobs), |
| $I$ | (number of stages), |
| $M$ | (number of machines), |
| $\mathcal{M}_i$ | (machines on stage $i$), |
| $M_i = \lvert\mathcal{M}_i\rvert$ | (number of machines on stage $i$). |

More accurately, $\mathcal{M}_i$ denotes the set of machine *indices* for the stage $i \in \{1, \ldots, I\}$, i.e. $m \in \mathcal{M}_i$ if and only if the $m$-th machine is part of stage $i$. The input variable of the problem is given by the matrix

$$P \in \mathbb{R}_+^{J\times M} \quad \text{(processing times)}$$

such that $P_{jm}$ denotes the time required for machine $m$ to process job $j$ on the stage $i$ with $m \in \mathcal{M}_i$.

Finally, a *schedule* is fully defined by the two matrices

$$A \in \{0,1\}^{J\times M} \text{ (assignments)}, \qquad S \in \mathbb{R}_{\geq 0}^{J\times I} \quad \text{(starting times)}.$$

Note that $S_{ji}$ denotes the starting time of job $j$ on the $i$-th stage (not the $i$-th machine). The assignment of jobs to machines is represented by the *assignment matrix* $A$: if job $j$ is assigned to machine $m$, then $A_{jm} = 1$; otherwise, $A_{jm} = 0$. Since in the classical hybrid flow shop problem, each job is assigned to exactly one machine on each stage, the requirement $A \in \mathcal{A}$ must hold, where

$$\mathcal{A} = \left\{ A \in \{0,1\}^{J\times M} \;\middle|\; \sum_{m\in\mathcal{M}_i} A_{jm} = 1 \quad \text{for all } j \in \{1,\ldots,J\},\, i \in \{1,\ldots,I\} \right\} \tag{2.1}$$

denotes the set of admissible assignment matrices. An extension of this set for the *assignment-relaxed* problem will be discussed in Section 3.

**Example 2.1.** We consider a basic example in order to clarify the notation employed in the following. Assume that for a hybrid flow shop problem, $M = 5$ machines are employed on $I = 2$ stages with

$$\mathcal{M}_1 = \{1, 2, 3\}, \qquad \mathcal{M}_2 = \{4, 5\},$$

i.e. 3 machines on the first and 2 machines on the second stage. Moreover, let the processing times for $J = 3$ jobs be given by the processing times matrix

$$P = \begin{pmatrix} 1.5 & 3.0 & 0.5 & 2.1 & 1.7 \\ 3.5 & 1.0 & 4.2 & 0.4 & 1.4 \\ 2.0 & 2.5 & 3.0 & 1.0 & 0.5 \end{pmatrix}.$$

Then

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}, \qquad S = \begin{pmatrix} 0.0 & 1.5 \\ 0.5 & 2.8 \\ 0.5 & 3.5 \end{pmatrix}$$

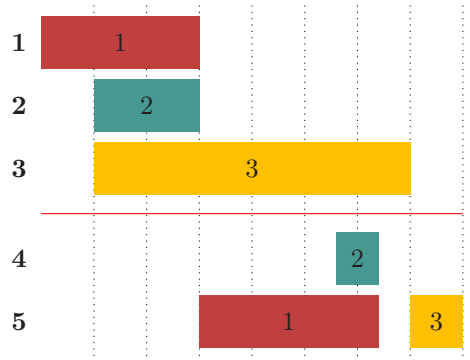



*Figure 4: Schedule from Example 2.1.*

is an admissible combination of assignments and starting times, with the resulting schedule shown in Fig. 4. In particular, if the job order is fixed according to the order of the rows of $P$, then the shown schedule satisfies all constraints for the permutation hybrid flow shop problem without any unnecessary gaps; note that neither job 2 nor job 3 can begin any earlier due to the two constraints for sequential processing, while job 3 starts on the second stage as soon as its processing is finished on stage 1. The starting times $S$ are therefore already implicitly determined by the order of jobs and the assignment matrix $A$.

**Remark 2.2.** In general, the starting times for a given job order can easily be obtained from the assignments $A$ and the processing times $P$: First, let $M_0 = \{\}$ and

$$S_{ji} = 0\,, \quad E_{jm} = 0 \qquad \text{for all} \quad i \in \{1,\dots,I\}\,, \\ m \in \{1,\dots,M\} \quad \text{and} \quad j \in \{1,\dots,J\}\,;$$

here, $E_{jm}$ represents the time at which the processing of job $j$ on machine $m$ ends. Then, for each job $j = 1,\dots,J$ (in sequential order), consider each stage $i = 1\dots,I$ in increasing order, and let $m$ denote the unique machine on stage $i$ which processes job $j$. Set

$$S_{ji} = \max\Big(\max_{n\in\mathcal{M}_{i-1}} E_{jn}\,,\ \max_{k\le j} E_{km}\,, \\ \max_{k\le j} S_{ki}\,,\ \max_{k\le j} E_{ki} - P_{jm}\Big) \tag{2.2}$$

and

$$E_{jm} = S_{ji} + P_{jm}\,.$$

This procedure ensures that processing starts at the earliest admissible time on each stage for each job under the constraints of the permutation hybrid flow shop. The four "inner" maxima in (2.2) ensure, respectively, that a job $j$ does not start on stage $i$ if it has not been fully processed on stage $i-1$, that machine $m$ has finished all previous processing and that no previous job $k$ is overtaken either at the start or the end of the stage.

### 2.3 Objective functions

Once the full schedule (consisting of the assignments and the starting times) is known, the two most common objective functions are easily computed: The *makespan* $C_{\max}$ is defined as the time at which the final job is finished on the last stage, i.e.

$$C_{\max}(P,A,S) = \max_{j\in\{1,\dots,J\}} S_{jI} + \sum_{m\in\mathcal{M}_I} P_{jm}A_{jm} \\ = \max_{j\in\{1,\dots,J\}} S_{jI} + \max_{m\in\mathcal{M}_I} P_{jm}A_{jm}\,.$$

For the permutation problem, the makespan is simply given by

$$C_{\max}(P,A) = \sum_{m\in\mathcal{M}_I} E_{Jm}A_{Jm} \\ = \max_{m\in\mathcal{M}_I} E_{Jm}A_{Jm} = \max_{m\in\mathcal{M}_I} E_{Jm}\,,$$

where $E$ is computed via the algorithm in Remark 2.2. Similarly, the *flowtime* $C_{\text{sum}}$ is given by

$$C_{\text{sum}}(P,A,S) = \sum_{j=1}^{J} S_{jI} + \sum_{m\in\mathcal{M}_I} P_{jm}A_{jm} \\ = \sum_{j=1}^{J} S_{jI} + \max_{m\in\mathcal{M}_I} P_{jm}A_{jm}$$

and represents the sum of the final finishing times of all jobs. Again, the equality can be expressed in terms of $E$ via

$$C_{\text{sum}}(P,A) = \sum_{j=1}^{J}\sum_{m\in\mathcal{M}_I} E_{jm}A_{jm} \\ = \sum_{j=1}^{J}\max_{m\in\mathcal{M}_I} E_{jm}A_{jm} = \sum_{j=1}^{J}\max_{m\in\mathcal{M}_I} E_{jm}$$

for the permutational case.

## 3 RELAXATION OF THE HYBRID FLOW SHOP PROBLEM

Note hat the equalities in Section 2.3 rely on the *discreteness* of the assignments, i.e. on the requirement that all but one of the values $A_{jm}$ are zero on any given stage. However, as indicated in Section 1, this discreteness is one of the major obstacles for the application of machine learning techniques, since any loss function defined on a discrete set is not directly suitable for gradient-based optimization procedures. In the following, we will therefore consider a relaxed approach by allowing *fractional* assignments as well and extending the loss function to the resulting larger domain of admissible matrices.

### 3.1 Assignment relaxation

Recall from (2.1) that for the classical hybrid flow shop problem, the set of admissible assignments is given by

$$\mathcal{A} = \Big\{A\in\{0,1\}^{J\times M}\,\Big|\,\sum_{m\in\mathcal{M}_i} A_{jm} = 1 \ \text{ for all} \\ j\in\{1,\dots,J\},\, i\in\{1,\dots,I\}\Big\}\,, \tag{3.1}$$

since each job must be assigned to exactly one machine on each stage. In the *assignment-relaxed* hybrid flow shop model, we allow for a *split* (or *fractional*) assignment of a job to multiple machines on a single stage in the form of a convex combination, which is described by the extended set

$$\mathcal{A}_{\mathrm{R}} = \Big\{A\in[0,1]^{J\times M}\,\Big|\,\sum_{m\in\mathcal{M}_i} A_{jm} = 1 \ \text{ for all} \\ j\in\{1,\dots,J\},\, i\in\{1,\dots,I\}\Big\} \tag{3.2}$$

of admissible assignment matrices. The notion of jobs being processed simultaneously on multiple machines has previously been discussed in the literature, both as a relaxation technique [31] and as an optimization problem in its own right [5]. Although our focus lies on the former approach, i.e. on viewing relaxation as a mathematical tool for obtaining classical solutions, it is nevertheless important to distinguish between different *interpretations* of fractional assignments in order to determine how the objective function can be reasonably extended to $\mathcal{A}_{\mathrm{R}}$.

**Job splitting**
First, we can consider an assignment of a job $j$ to multiple machines $m_1, \ldots, m_n$ on a stage $i$ as a split of the job itself into multiple parts. In this case, the weight factor $A_{jm} \in [0, 1]$ determines the fraction of the job which is being processed by machine $m$. Then the processing time for this part is naturally given by $A_{jm} \cdot P_{jm}$. During this time, machine $m$ is considered to be fully occupied. The total processing time $\Delta t$ for job $j$ on stage $i$ is then given by

$$\Delta t = \max\{A_{jm} \cdot P_{jm} \,|\, m \in \mathcal{M}_i\}\,, \tag{3.3}$$

i.e. the job is finished on the stage once all parts have been processed.

**Machine splitting**
Alternatively, we can interpret a fractional assignment as a partial occupation of a particular machine such that $A_{jm} \in [0, 1]$ represents the fraction of machine $m$ which is dedicated to the processing of job $j$. In this case, the job itself is considered to be undivided, with the required processing time $\Delta t$ on stage $i$ given by the weighted harmonic mean[1]

$$\Delta t = \frac{\sum\limits_{m\in\mathcal{M}_i} A_{jm}}{\sum\limits_{m\in\mathcal{M}_i} \frac{A_{jm}}{P_{jm}}} = \frac{1}{\sum\limits_{m\in\mathcal{M}_i} \frac{A_{jm}}{P_{jm}}}\,. \tag{3.4}$$

**Example 3.1.** For $J = 3$, $M = 5$ and $I = 2$ with $\mathcal{M}_1 = \{1, 2, 3\}$ and $\mathcal{M}_2 = \{4, 5\}$, we consider the same processing times matrix $P$ as in Example 2.1. For the relaxed assignment $A$ and the starting times $S$ with

$$A = \begin{pmatrix} 1 & 0 & 0 & 0.3 & 0.7 \\ 0 & 1 & 0 & 0.7 & 0.3 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \text{ and } S = \begin{pmatrix} 0.0 & 1.5 \\ 0.5 & 2.69 \\ 0.5 & 3.5 \end{pmatrix},$$

we obtain the schedules shown in Figs. 5 and 6 for the interpretation of the relaxation as job splitting and machine splitting, respectively.

Since the notion of machine splitting would introduce additional rectification discontinuities for the purely assignment-relaxed hybrid flow shop (cf. 4.3), we will mainly focus on the case of job splitting in the following.

### 3.2 The assignment-relaxed permutation hybrid flow shop problem

In order to extend the definition of the two main objective functions $C_{\max}$ and $C_{\mathrm{sum}}$ from Section 2.3 to the alignment relaxed case, we assume again that the jobs are given in a fixed order. Then, for given assignments $A$, the schedule can be created analogously to the procedure described in Remark 2.2. However, following the job-splitting interpretation of non-binary assignments, the actual processing time on each stage needs to be modified according to eq. (3.3). Furthermore, a single job might occupy multiple

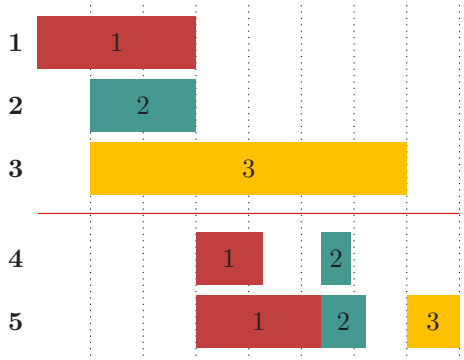



*Figure 5: Schedule with job splitting.*

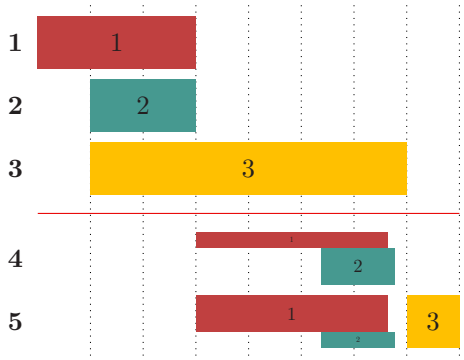



*Figure 6: Schedule with machine splitting.*

[1] The use of the harmonic mean is due to the interpretation of $A_{jm}$ as the ratio of the machine's performance $p_{jm}$, which is related to the processing time via $P_{jm} = 1/p_{jm}$.

machines on a single stage for different time intervals, which needs to be taken into account for the scheduling as well.

Once the schedule is set up, the relaxed makespan $C^{\mathrm{R}}_{\max}$ and the relaxed flowtime $C^{\mathrm{R}}_{\mathrm{sum}}$ are easily computed, respectively, via the maximum and the sum of the end-of-processing times over all jobs on the final stage. This procedure is formalized in Definition 3.2 with an algorithm in tensorial form, as is required by many machine learning frameworks.

**Definition 3.2.** The *relaxed makespan* and the *relaxed flowtime* are defined by Algorithm 1:

**Algorithm 1:** Relaxed Loss

**RelaxedLoss** $(P, A)$
  **inputs :** Processing times matrix $P$ of size $J \times M$,
    Assignment matrix $A$ of size $J \times M$
  **output:** The relaxed losses $C^{\mathrm{R}}_{\max}, C^{\mathrm{R}}_{\mathrm{sum}}$
  $P^* \leftarrow P \cdot A$ // elementwise multiplication
  jobs $\leftarrow$ rows of $P^*$
  MO $\leftarrow \vec{0}_{1\times M}$ // MO: Machine Occupation
  $\mathrm{SO}_{start} \leftarrow \vec{0}_{1\times I}$ // SO: Stage Occupation
  $\mathrm{SO}_{end} \leftarrow \vec{0}_{1\times I}$
  **foreach** $j \in$ jobs **do**
    $\mathrm{JO}_j \leftarrow 0$
    **foreach** $i \in$ stages **do**
      $\mathrm{SOstart}_i \leftarrow$ SOstart[i]
      $\mathrm{SOend}_i \leftarrow$ SOend[i]
      $\mathrm{JO}_j \leftarrow$ SOend[i]
      **foreach** $m \in \mathcal{M}_i$ **do**
        $\mathrm{P}^*_{jm} \leftarrow$ P*[job, $m$] // Weighted processing time for job j on machine m
        $\mathrm{MO}_m \leftarrow$ MO[m]
        $\mathrm{SOend}^{\mathrm{shift}}_i \leftarrow \mathrm{SOend}_i - \mathrm{P}^*_{jm}$
        $\mathrm{startTime}_{jm} \leftarrow \max(\mathrm{JO}_j, \mathrm{MO}_m, \mathrm{SOstart}_i, \mathrm{SOend}^{\mathrm{shift}}_i)$
        $\mathrm{C}_{jm} \leftarrow \mathrm{startTime}_{jm} + \mathrm{P}^*_{jm}$ // Completion time of job j on machine m
        MO[m]$\leftarrow$ C[j,m]
        SOstart[i] $\leftarrow \max(\mathrm{SOstart}_i, \mathrm{C[j,m]} - \mathrm{P}^*_{jm})$
        SOend[i] $\leftarrow \max(\mathrm{SOend}_i, \mathrm{C[j,m]})$
    $C^{\mathrm{R}}_{\max} \leftarrow \max$(SOend[I])
    $C^{\mathrm{R}}_{\mathrm{sum}} \leftarrow \sum$ SOend[I]
**return** $C^{\mathrm{R}}_{\max}, C^{\mathrm{R}}_{\mathrm{sum}}$

Note that for any classically admissible assignment $A \in \mathcal{A} \subset \mathcal{A}_{\mathrm{R}}$, the relaxed makespan is equal to the classical makespan. Furthermore, the domain in Definition 3.2 ensures that the jobs are indeed "split" among the machines on a stage, which is implicitly assumed by the algorithm.

If, for a neural network, we ensure via a suitable choice for the output layer that the output values remain in the domain of the relaxed makespan, then this function can directly be used as the loss function for the training process.

### 3.3 Starting time relaxation

In addition to the assignments, we can also relax the various constraints on the starting times, e.g. that no two jobs $j_1, j_2$ can occupy the same machine at the same time or that processing of job $j$ on a stage $i$ cannot start before $j$ has been fully processed on all previous stages $1, \ldots, i-1$. Most generally, we can admit any matrix $S \in \mathbb{R}^{J\times I}_{\geq 0}$ as *relaxed starting times*. Alternatively, we can prescribe some simple additional constraints which are independent of the processing times, such as the requirement that $S_{j,i} \geq S_{j,i-1}$ for all $j \in \{1, \ldots, J\}$ and all $i \in \{2, \ldots, I\}$, i.e. that no job can be processed on a stage before it has even started on the previous one. In that case, the extended set of admissible starting times is given by

$$\mathcal{S}_{\mathrm{R}} = \left\{ S \in \mathbb{R}^{J\times I}_{\geq 0} \mid S_{ji} \leq S_{jk} \text{ for all } j \in \{1, \ldots, J\} \text{ and } i, k \in \{1, \ldots, I\} \text{ with } i \leq k \right\}.$$

### 3.4 Penalty functions

In general, optimal solutions to relaxed problems are not admissible for the classical, non-relaxed hybrid flow shop problem; for example, the machine assignment for which the relaxed makespan attains its minimum might require the assignments 0.3 and 0.7, respectively, on two machines of the first stage for the first job, whereas a single machine would need to be selected in the classical problem. While a conversion to the "nearest" admissible solution (cf. Section 4.3) is always possible – in this case by selecting the machine with the highest assignment value – it cannot be ensured that the resulting classical solution is in any sense optimal, especially if the relaxed solution is not "sufficiently close" to any classical one. Therefore, the relaxed objective function needs to be supplemented by additional *penalty terms* which penalize deviations from the set of classically admissible solutions [11].

#### *3.4.1 Penalties for the assignment relaxed case*

We first consider the case of assignment relaxation. A straightforward penalty of the split between assignments is given by the function

$$p_0 \colon \mathcal{A}_\mathrm{R} \times \mathbb{R}_+^{J\times M} \to \mathbb{R}\,,$$

$$p_0(A,P) = \sum_{j=1}^{J}\sum_{i=1}^{I} \sum_{\substack{m\in\mathcal{M}_i \\ m\neq \operatorname{argmax}_{n\in\mathcal{M}_i} A_{jn}}} A_{jm}\cdot P_{jm}\,.$$

For a given assignment $A$ and a processing times matrix $P$, the term $p_0(A, P)$ represents the sum over all processing times *except* the ones on machines with the maximum assignment value for a given stage-job combination. Therefore, $p_0$ penalizes all processing on multiple machines and thereby all deviations from a *discrete*, admissible assignment on each stage.

Alternatively, we can consider the penalty

$$p_1 \colon \mathcal{A}_\mathrm{R} \to \mathbb{R}\,,$$

$$p_1(A) = \sum_{j=1}^{J}\sum_{i=1}^{I} \sum_{m,n\in\mathcal{M}_i m\neq n} A_{jm}\cdot A_{jn}\,,$$

which does not depend on the processing times $P$ and is differentiable at any $A \in \mathcal{A}_\mathrm{R}$. For every combination of jobs and stages, the function $p_1$ additively weighs the split of the job between any combination $m$, $n$ of machines on the stage. Note that

$$\sum_{m\in\mathcal{M}_i} A_{jm} = 1$$

for every $j \in \{1, \ldots, J\}$ and $i \in \{1, \ldots, I\}$ due to the choice of $\mathcal{A}_\mathrm{R}$ as the domain of definition for $p_1$, which ensures that $p_1(A) \geq 0$ for all $A \in \mathcal{A}_\mathrm{R}$ and that $p_1(A) = 0$ if and only if $A \in D$, i.e. $p_1$ attains the minimum value exactly at the classically admissible assignments.[2]

#### *3.4.2 Penalties for the starting-time relaxed case*

If the starting times are relaxed as well, two additional issues need to taken into account: first, a job might start being processed on a machine without having completed the previous stage; second, two different jobs might be assigned (in parts) to the same machine over some intersecting time intervals. To penalize the former, we introduce the penalty function

$$p_2 \colon \mathcal{A}_\mathrm{R} \times \mathcal{S}_\mathrm{R} \to \mathbb{R}\,,$$

$$p_2(A,S) = \sum_{j=1}^{J}\sum_{i=1}^{I-1} \max\{0, S_{j,i} + \Delta t(A,j,i) - S_{j,i+1}\}$$

where (cf. eq. (3.4))

$$\Delta t(A,j,i) = \frac{1}{\sum\limits_{m\in\mathcal{M}_i} \frac{A_{jm}}{P_{jm}}}$$

denotes the (relaxed) processing time of job $j$ on stage $i$ for the assignment $A$. The penalty $p_2$ therefore measures the overlap between processing intervals on consecutive stages; recall that due to the choice of $\mathcal{S}_\mathrm{R}$ in the domain of definition, it can be assumed a priori that the starting times $S_{ji}$ are increasing for each job $j$.

Finally, we consider the penalty function

$$p_3 \colon \mathcal{A}_\mathrm{R} \times \mathcal{S}_\mathrm{R} \to \mathbb{R}\,,$$

$$p_3(A,S) = \sum_{\substack{j,k\in\{1,\ldots,J\} \\ j\neq k}} f_d(A,S,j,k,i)\cdot f_s(A,S,j,k,i),$$

where $f_d(A, S, j, k, i)$

with

$$f_d(A,S,j,k,i) = \max(0, S_\mathrm{max} - E_\mathrm{min}),$$
$$S_\mathrm{max} = \max(S_{ji}, S_{ki}),$$
$$E_\mathrm{min} = \min(S_{ji} + \Delta t(A,j,i), S_{ki} + \Delta t(A,k,i))$$

is the duration of a possible overlap[3] between the processing of jobs $j$ and $k$ on stage $i$ and

$$f_s(A,B,j,k,i) = \frac{1}{\sqrt{\left(\sum\limits_{m\in\mathcal{M}_i} A_{jm}^2\right)\left(\sum\limits_{m\in\mathcal{M}_i} A_{km}^2\right)}} \cdot \sum_{m\in\mathcal{M}_i} A_{jm}A_{km}$$

[2] It is also easy to see that $p_1$ does not attain any other local minima and that $\sum_{m\neq n} A_{jm}\cdot A_{jn}$ is maximal if and only if $A_{jm} = \frac{1}{|\mathcal{M}_i|}$ for all $m \in \mathcal{M}_i$.

[3] Here, $S_\mathrm{max}$ and $E_\mathrm{min}$ denote the later starting times and the earlier end time of two jobs, respectively, so that $S_\mathrm{max} - E_\mathrm{min}$ is positive if and only if there is an overlap between the two processing intervals, in which case, the difference is equal to the length of the overlap.

measures to what degree the two jobs occupy the same machines during any overlap; note that the function value $f_d(A, B, j, k, i) \in [0, 1]$ is equal to 1 if and only if the two jobs have the exact same assignment on the stage and equal to 0 if and only if they share no machines at all.

## 4 HYBRID FLOW SHOP SCHEDULING WITH NEURAL NETWORKS

While the relaxation of the hybrid flow shop circumvents the difficulties which arise from the discreteness of the problem, applying machine learning methods to predict optimal schedules remains challenging due to the lack of accurately labeled training data. As indicated in Section 1.1, we therefore employ a self-supervised training method which uses the relaxed objective functions $C^{\mathrm{R}}_{\max}$ and $C^{\mathrm{R}}_{\mathrm{sum}}$ directly as part of the loss function for an artificial neural network.

In traditional supervised learning, a neural network is generally trained by minimizing the deviation of the network's output $\widehat{y}$ from the known *labels* (or g*round truths*) $y$ associated with the training data $x$. Most commonly, this deviation is measured by the *mean square error*, i.e. by

$$L(y, \widehat{y}) = \frac{1}{N} \sum_{n=1}^{N} |\widehat{y}_n - y_n|^2 \,,$$

where $y_1, \ldots, y_N$ and $\widehat{y}_1 \ldots, \widehat{y}_N$, denote the labels and the output predicted by the neural network for the corresponding input data, respectively.

While machine learning methods are often applied to empirical measurement data in order to provide a data-driven model in cases where no analytical model is available, the same methods are applicable to the task of finding approximate solutions to analytical minimization problems. In this case, finding the "true" solutions $y_1, \ldots, y_N$ to the minimization problem by other (numerical) means would be necessary in order to provide a labeled training dataset for the neural network. Of course, depending on the complexity of the minimization task, generating a sufficiently large set of labels might be computationally too expensive for any practical applications; for the NP-hard problems considered here, this is clearly the case.

In order to employ neural networks to find optimal solutions to the hybrid flow shop problem, we again assume that the number of jobs, stages and machines ($J$, $I$ and $M$) as well as the distribution $\mathcal{M}_i$ of machines to the stages $i = 1, \ldots, I$ remain fixed (cf. Section 4.5). Then the only input parameter of the neural network is the processing times matrix $P \in \mathbb{R}_+^{J \times M}$.

In the following, we will focus on the pure assignment problem: assuming that the order in which the jobs need to be processed is fixed, we aim to find the optimal assignment strategy, i.e. the assignment $A \in \mathcal{A}$ such that the resulting objective function – either the makespan or the flowtime – is minimal.[4] For the relaxed problem, the neural network itself can then be described as a mapping

$$F\colon\ \mathbb{R}_+^{J \times M} \to \mathcal{A}_{\mathrm{R}} \,, \quad P \mapsto \widehat{A} = F(P) \,,$$

where $\widehat{A}$ denotes the predicted relaxed assignment for the processing times $P$. In a final step, the matrix $\widehat{A}$ is then rectified (cf. Section 4.3) in order to obtain a classical admissible schedule $A \in \mathcal{A}$.

### 4.1 Training process

In order to train the neural network, i.e. to determine the network parameters such that the prediction $\widehat{A} = F(P)$ represents an optimal assignment, we directly employ the relaxed makespan function from Definition 3.2 as the loss function for the training process. Thereby, it is sufficient to create a random training dataset $\mathcal{T} \subset \mathbb{R}_+^{J \times M}$ without pre-computing any labels. For given input $P \in \mathcal{T}$, the loss $L$ is then simply defined as[5]

$$L(P, \widehat{A}) = C^{\mathrm{R}}_{\max}(P, \widehat{A}) + p(P, \widehat{A}) \,,$$

where $\widehat{A} = F(P)$ denotes the predicted assignment matrix and $p(P, \widehat{A})$ is a suitable penalty term, as described in Section 3.4. Due to the continuous structure of both the output and the input space, and since the $C^{\mathrm{R}}_{\max}$ is sufficiently regular, it is then possible to employ gradient-descent based methods in order to find optimal network parameters such that the loss $L$ is minimized on the training dataset, i.e. the loss

$$L(\mathcal{T}) = \sum_{P \in \mathcal{T}} L(P, F(P))$$

is minimized over the parameter space. By the choice of $L$, such a minimization ensures that the relaxed objective function value, e.g. $C^{\mathrm{R}}_{\max}(\widehat{A})$, is sufficiently small and, via the penalty $p$, that $\widehat{A} = F(P)$ is close to a classical assignment $A \in \mathcal{A}$ for any $P \in \mathcal{T}$; note again that $C^{\mathrm{R}}_{\max}(A) = C_{\max}(A)$ if $A \in \mathcal{A}$. Of course, whether the rectification of an assignment $\widehat{A} = F(P)$ also results in an optimal or close-to-optimal solution for $P \notin \mathcal{T}$ should be monitored during the training process on a suitable validation dataset and – most importantly – after the training is complete by using a separate set of test data.

[4] Note again that if a neural network is indeed able to identify an optimal (or a near-optimal) assignment for a given job order, it can easily be combined with additional dispatching rules which optimize the job order itself, such as the machine learning approach presented in [34].

[5] Of course, the training method can be performed analogously if the relaxed makespan $C^{\mathrm{R}}_{\max}$ is replaced with the relaxed flowtime $C^{\mathrm{R}}_{\mathrm{sum}}$.

Although this approach seems remarkably straightforward, it has, to the best of our knowledge, not found a widespread use for discrete optimization problems. We will demonstrate the viability of the method in Section 5.

### 4.2 Strict constraints

While the requirement of discrete machine selections has been relaxed in our loss function, we still require the condition

$$\sum_{m\in\mathcal{M}_i} \widehat{A}_{jm} = 1 \tag{4.1}$$

for all $j \in \{1,\dots,J\}$, $i \in \{1,\dots,I\}$ to be strictly satisfied by the assignment output of the neural network. Note that, in particular, the penalty functions introduced in Section 3.4 are no longer viable for arbitrary assignment values on the individual stages.

This strict constraint can be ensured by a suitable choice of the network's output layer. More specifically, if the final layer of the neural network applies the *softmax function*

$$\sigma\colon \mathbb{R}^M \to \mathbb{R}^M,\quad \sigma(x_1\dots,x_M)_m = \frac{e^{x_m}}{\sum_{k=1}^M e^{x_k}}$$

to the assignment values for each stage and each job, then the output is properly constrained: Since the range of the softmax function is given by

$$\sigma(\mathbb{R}^M) = \{(t_1,\dots,t_M)\in(0,1)^M \mid \textstyle\sum_{m=1}^M t_m = 1\},$$

each assignment matrix $\widehat{A}$ predicted by the neural network must then satisfy the constraint (4.1) by construction.[6]

### 4.3 Rectification

Since the output of the neural network is, in general, not an admissible schedule due to the employed relaxation, it is necessary to apply a *rectification* method in order to obtain a classical prediction from the ANN. For the assignment matrix, a simple rounding technique can be used by stagewise assigning each job to the machine with the highest relaxed assignment value:

$$A_{jm} = \begin{cases} 1 & \text{if } \widehat{A}_{jm} = \max\limits_{n\in\mathcal{M}_i} \widehat{A}_{jn}\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

Note that the computational overhead due to this conversion is negligible and that output matrices which are already admissible are invariant under these rectification methods, i.e. if both $\widehat{A}$ and $\widehat{S}$ satisfy the hybrid flow shop constraints, then $A = \widehat{A}$ and $S = \widehat{S}$.

### 4.4 Applications

The main advantage of the proposed method is the very low computational effort required for determining the processing times once the neural network is trained. This performance advantage is particularly important in cases where high-frequency scheduling is required, such as applications in scheduling of processor services [1]. The simulation of logistical processes also requires extremely fast decision processes: if a hybrid flow shop is used as part of a more extensive model, approximate solutions can be used during the simulation to provide a plausible model of the scheduling. Since numerous repetitions might be required for various simulations, the performance offered by neural-network-based predictions would be highly advantageous.

### 4.5 Limitations

In general, it should not be expected that neural networks are able to predict globally optimal schedules for given processing times: Both the applied relaxation techniques and the machine learning methods themselves introduce possible sources of error. Although gradient-descent-based optimization methods, such as the *Adam* algorithm employed in the numerical examples in Section 5, are capable of finding local minima for very general classes of functions, they usually cannot ensure that global minima are identified. The neural-network-based approach presented therefore does not provide an exact solution, but rather an approximation method for the hybrid flow shop scheduling problem.

Furthermore, while the computational cost for finding a schedule using a trained neural network is extremely low when compared to other optimization methods (cf. Section 5), the effort required during the training process needs to be considered as well. However, the training does not need to be performed "online" during high-frequency scheduling applications; instead, the network can be prepared at any point, and the training can be continued in the background over time to further improve results.

Finally, it should be noted that, due to the fixed topology of the trained neural network, the number of jobs, stages and machines per stage cannot be changed without restarting the training process. However, for a reduced number of jobs, a schedule can still be predicted via zero padding of the input, i.e. by including additional jobs with zero processing time. The number of stages can be reduced in a similar fashion, whereas the number of machines per stage could be decreased by adding machines with prohibitively large processing times to the input ("infinity padding"). The maximum number of jobs, stages and machines for which the network is applicable, however, remains bounded by the selected training data. Due to the increasing number

[6] Note that this method is closely related to the classical approach of minimizing a loss function $L$ under strict constraints by finding a minimizer of $L \circ g$, where the range of $g$ is the proper (constrained) domain of $L$.

of parameters and the resulting computational effort in training the network for larger input dimensions, a limit for these numbers needs to be carefully chosen beforehand.

## 5 NUMERICAL EXAMPLES

To demonstrate the viability of the proposed method, we implemented and trained a neural network for the prediction of assignments – as described in Section 4 – in Python using the *TensorFlow* library[7] and evaluated the assignments predicted by the neural network in terms of the classical makespan (cf. Remark 2.2).

For small problem sizes (up to 6 jobs), the computed schedules are compared directly to exact solutions, which were obtained via mixed integer linear programming. For larger numbers of jobs, finding an optimal solution was no longer feasible. We therefore compare our results for larger input sizes to other high-performance approximation methods: the Monte Carlo method, which simply selects the best outcome out of multiple randomly generated assignments, and time-limited linear programming. We also show that an already trained neural network can effectively be applied to smaller numbers of jobs via zero padding, as described in Section 4.5.

### 5.1 Distribution of generated processing times

For all numerical experiments, a hybrid flow shop consisting of three stages with two machines each was considered. For the machines on each stage $i$, we generated normally distributed processing times $P_{jm}$ which were then rounded to integer values. For the specific distributions $P_{jm} \sim \mathcal{N}(\mu_i, \sigma_i^2)$ on the stages, we randomly selected a mean value $\mu_i \in [10, 900]$ between 10 and 900 as well as a standard deviation $\sigma_i^2 \in [1, \mu_i - 10]$; samples with $P_{jm} < 1$ were rejected. This randomization method results in problem matrices with varied processing times.

**Example 5.1.** The matrices

$$P_1 = \begin{pmatrix} 112 & 113 & 471 & 465 & 28 & 28 \\ 112 & 111 & 469 & 471 & 27 & 25 \\ 110 & 110 & 470 & 467 & 24 & 23 \\ 113 & 109 & 471 & 467 & 24 & 27 \\ 111 & 110 & 471 & 466 & 25 & 25 \\ 111 & 110 & 473 & 468 & 26 & 24 \end{pmatrix} \quad \text{and}$$

$$P_2 = \begin{pmatrix} 195 & 142 & 855 & 436 & 275 & 486 \\ 53 & 902 & 1048 & 1121 & 445 & 70 \\ 1899 & 661 & 1130 & 1167 & 2206 & 842 \\ 1983 & 907 & 536 & 1178 & 1706 & 1261 \\ 347 & 1067 & 785 & 478 & 2051 & 168 \\ 248 & 490 & 451 & 1798 & 93 & 1292 \end{pmatrix}$$

represent two examples of processing times matrices generated by the above method with different mean values and standard deviations on the stages for $J = 6$ jobs. The processing times on each stage are normally distributed. The stages in $P_1$ exhibit very small standard deviations which emphasizes the division between the stages, whereas the high standard deviations in $P_2$ result in a less structured problem.

### 5.2 Structure of the neural network

The model of the neural network created to predict the assignment matrix $A$ combines elements of convolutional neural networks and recurrent neural networks.[8] We first perform convolutions over the machines on each stage of each job. The subsequent recurrent part of the network consists of so-called *long short-term memory* layers as proposed by Hochreiter et al. [10]. These layers are most commonly used for time series analysis, where they are employed to take into account datasets preceding the current one; here, we intended to exploit the well-defined job order for the considered problem, which suggests that the individual processing times can be regarded as time-wise dependent datasets.

Afterwards, the model is split into several parallel dense layers. Each dense layer represents the machines on a specific stage of a specific job. Our choice of $J = 6$ jobs, $I = 3$ stages and two machines on each stage results in $J \cdot I = 18$ layers with two nodes each. Fig. 7 shows a visualization of the neural network's structure for $J = 3$, with $J \cdot I = 9$ parallel dense layers; some normalization, reshape and dropout layers were omitted for simplicity. As described in Section 4.2, we apply a softmax activation function on the outputs of these parallel layers, so that the output values of each layer are positive numbers and add up to 1. Eventually, the parallel layers are concatenated to obtain the relaxed assignment matrix $\widehat{A}$, which can then be converted to the final assignment $A$ predicted by the neural network.

Note that the number of nodes in the neural networks scales with the problem size. Therefore, for a larger number $J$ of jobs, the complexity of the neural network and thus the computation time required both for the training process and the prediction increases.

### 5.3 Evaluation

For $I = 3$ stages with two machines each (i.e. $M = 6$ machines total) and up to $J = 6$ jobs, the exact solution to the assignment problem for a fixed job order could still be computed in a reasonable amount of time with the resources available. For 1000 randomly generated instances $P_{\text{opt}}$ of processing times, we therefore computed the optimal minimum makespan $C_{\max}^{\text{opt}}$ using the commercial optimization tool *Gurobi* [9].

[7] Some obstacles had to be overcome during the implementation; note that Algorithm 1 still requires some care in order to remain compatible with tensorial operations, especially with respect to variable assignments.

[8] The results presented in Section 5.4 were obtained with the neural network structure described and exceeded earlier attempts with generic networks consisting purely of densely connected layers.

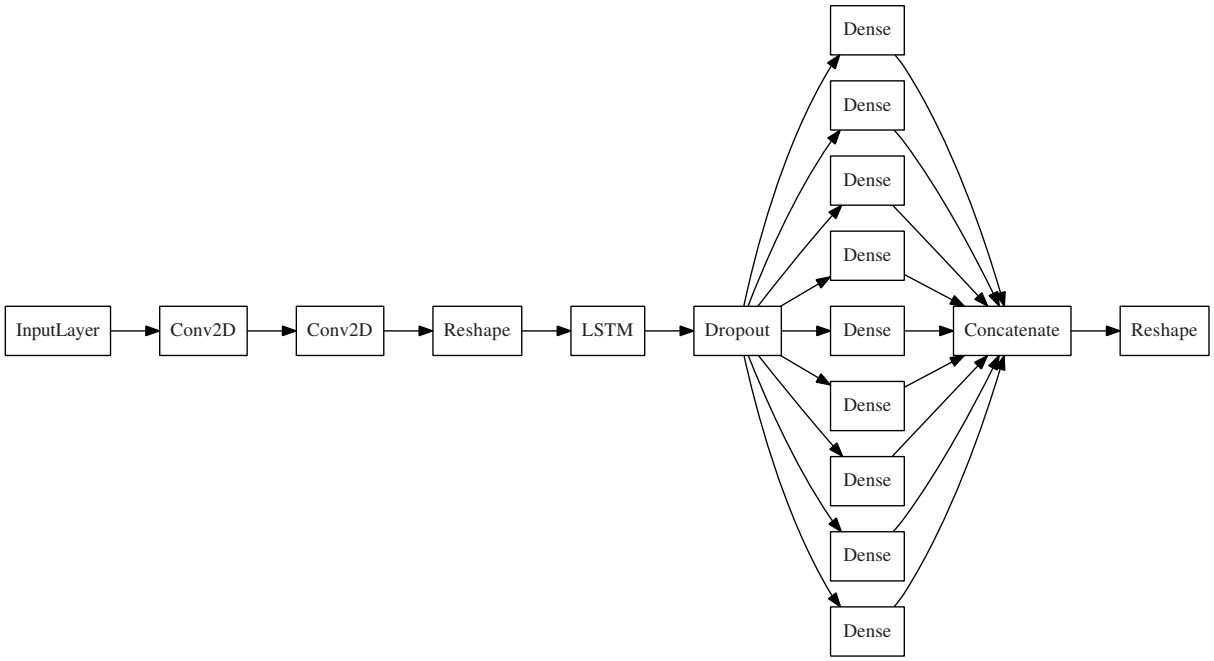



*Figure 7: Overview of the structure of the neural network model to predict the assignment matrix A for J = 3.*

For larger problem sizes, we compare our results to approximate solutions obtained via the Monte Carlo method and time-limited mixed linear integer programming.

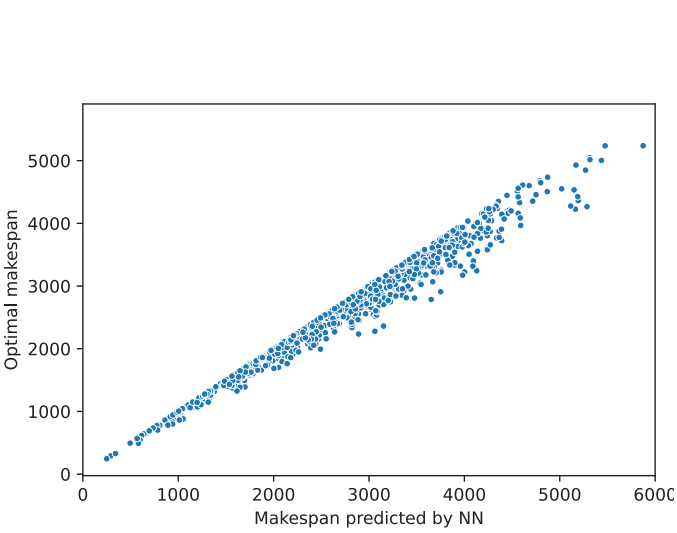
**Monte Carlo method**
For each input $P$, we apply a simple Monte Carlo algorithm by selecting 1000 possible (classical) assignment matrices $A \in \mathcal{A}$ and compute the makespan for each. The best of these assignment matrices is then chosen as the output. Although one of the most simple algorithms imaginable, this method of *guessing* solutions tends to yield remarkably good results in practice, especially comparatively small problem sizes. The limit of 1000 generated input matrices ensures a limit to the computation time; note that determining the makespan for each of the selected assignment matrices must be performed according to the scheduling procedure from Remark 2.2.

**Mixed integer linear programming**
For our test dataset, we also compute an approximate solution to the scheduling problem by applying a mixed integer linear programming solver to the input $P$ for a limited amount of time. As for the Monte Carlo method, no relaxation is required in this case, since integer programming allows for constraints on the solution variables which ensure that the resulting assignment matrix $A$ contains only integers or, in this case, the binary values 0 and 1. Note that for unlimited computation time, this method would always yield an exact solution. For high performance applications, however, such a limit would be unavoidable, especially for larger problem sizes. For our numerical experiments, we use the commercial mathematical solver *Gurobi* which, due to its high degree of optimization [22], should be considered a state-of-the-art competitor in terms of computational performance.

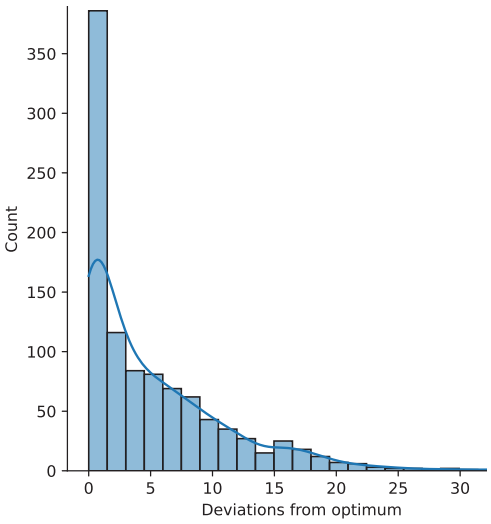
**5.4 Results**
We begin by training the neural network outlined in Section 5.2 according to the procedure from Section 4 for the input size of $J = 6$ jobs, $I = 3$ stages and two machines on each stage. The resulting neural network is then applied to a separate test dataset of 1000 processing time matrices for which optimal solutions have been determined beforehand. All experiments are performed on a 3.4 Ghz Intel i7 CPU with 4 cores and 16 GB of memory.

Figure 8 shows a direct comparison of the schedules predicted by the neural network compared to the true optimal results. In particular, for 20% of all predicted assignments, the corresponding makespan is indeed globally optimal. Overall, the mean deviation of the makespan for the predicted schedule is 4.9%.

The network is then trained for the larger problem size of $J = 12$ jobs. The trained network is then applied to the same test dataset via *zero padding* (cf. Section 4.5); in this case, six rows of zeros concatenated to the matrices in order to create a matrix of size $12 \times 6$. The accuracy of this prediction is comparable to the one obtained for the network specifically trained for this input size: a global optimizer is predicted in 16% of the cases, with a mean deviation of 7% from the globally optimal solution. These results suggest that networks trained for larger input sizes are indeed applicable




(a) Predicted makespan vs. globally optimal makespan.




(b) Distribution of deviations from the globally optimal makespan.

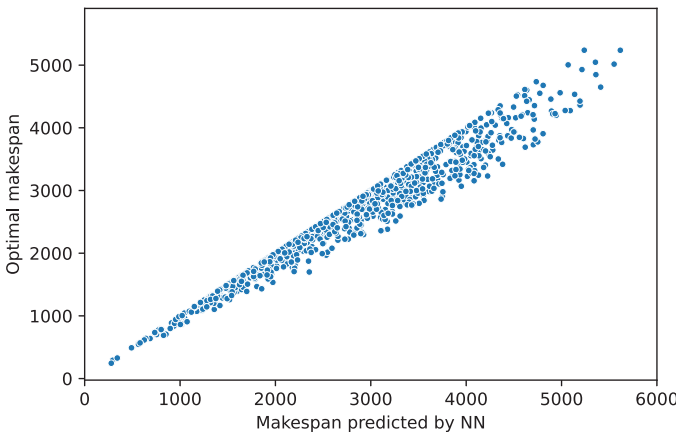
*Figure 8: Results for the assignment of $J = 6$ jobs predicted by a neural network.*

to a smaller number of jobs as well, which implies a higher degree of flexibility for this approach, since the necessity of preparing for every possible combination of input sizes is thereby avoided.

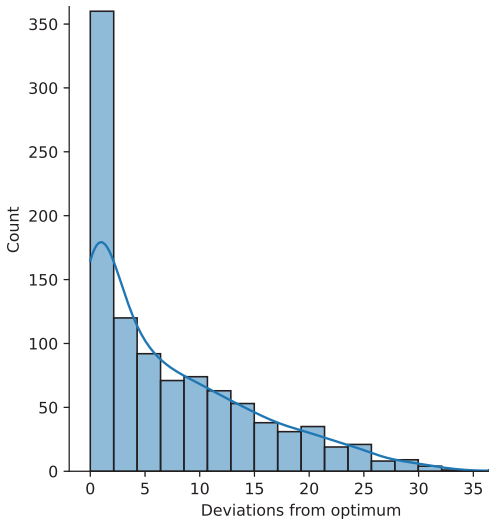
Finally, we apply the neural network to a test dataset of 1000 random problem matrices with $J = 6$ jobs, $I = 3$ stages and two machines on each stage. For each processing time $P$ in the dataset, assignments are also computed using the Monte Carlo method and Gurobi, with the latter limited to a computation time of 0.2 seconds.

A comparison of the results is shown in Fig. 10. The approximation quality of the three different methods is indeed very similar: Compared to the Monte Carlo heuristic, the assignment predicted by the neural network results in a lower makespan for ~ 60% of all processing times, while a direct comparison between the linear programming method and the neural network shows that each provides a better result in ~ 50% of the cases.

It is important to note that for this “break-even point”, the computation time for a single prediction of




(a) Predicted makespan vs. globally optimal makespan.




(b) Distribution of deviations from the globally optimal makespan.

*Figure 9: Results for the assignment of $J = 6$ jobs, predicted by a neural network trained for $J = 12$ jobs.*

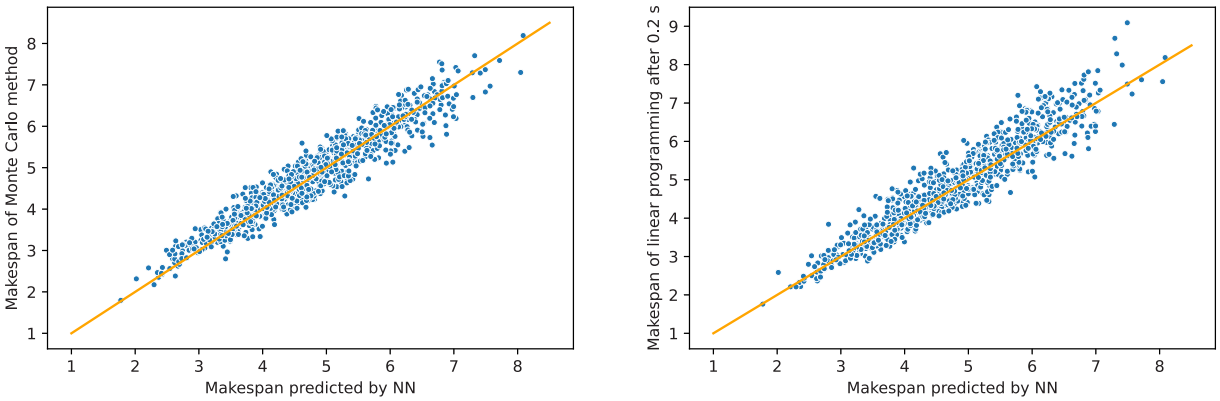



*Figure 10: Comparison of neural network predictions to time-limited mixed integer linear programming and the Monte Carlo method.*

the neural network is *significantly* lower than the 0.2 seconds allocated to Gurobi; in fact, while applying the Monte Carlo method and the linear programming solver to the entire dataset requires – on our test system – more than 15 minutes and more than 3 minutes, respectively, all 1000 assignments can be predicted by the neural network in less than 4 seconds, including even some additional overhead for loading the model parameters.

Our results suggest that for highly time-critical problems, the proposed method exceeds the performance of both alternatives significantly. For larger instances, these advantages should be expected to increase even further due to the differences in scaling between the methods.

## 6 SUMMARY AND OUTLOOK

Our numerical results show that neural networks can provide a highly performant method for finding approximate solutions to the hybrid flow shop scheduling problem for a given job order. The approach presented here, which combines the relaxation of binary assignments to fractional values via job splitting with a self-supervised learning technique that directly employs a relaxed objective function instead of labeled training data, when compared to the Monte Carlo method and a time restricted MILP approach with comparable accuracy, was able to outperform both in terms of computational time required for the decision process. With additional computational resources or improvements to the neural network structure, it may also be possible to significantly improve upon our results. It should be noted, however, that a further improvement of the neural network's accuracy would indeed require such structural changes or a more extensive training process, whereas both alternatives considered here could be scaled more conveniently to a higher expected accuracy, albeit at the cost of even higher computational time requirements.

Further investigation of the applicability of this promising approach to other NP-hard problems is strongly suggested by these results. In particular, adaptations to more complex extensions of the hybrid flow shop problem, such as tooling constraints, could be considered by a suitable adaptation of the loss function or by adding corresponding penalty terms. Stochastic neural networks could also be investigated with respect to the problem of uncertain arrival times. Finally, the scalability to larger problem sizes should be examined more closely if the computational capacity is available.

## 7 REFERENCES

[1] A. Allahverdi and F. S. Al-Anzi. "Scheduling multi-stage parallel-processor services to minimize average response time". J*ournal of the Operational Research Society* 57.1 (2006). Pp. 101–110. issn: 0160-5682. doi: 10.1057/palgrave.jors.2601987.

[2] M. A. Aragon-Calvo and J. Carvajal. "Self-supervised learning with physics-aware neural networks – I. Galaxy model fitting". *Monthly Notices of the Royal Astronomical Society* 498.3 (2020). Pp. 3713–3719.

[3] Y. Bengio, A. Lodi, and A. Prouvost. *Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon*. 2018. url: http://arxiv.org/pdf/1811.06128v2.

[4] T. Beucler, M. Pritchard, S. Rasp, J. Ott, P. Baldi, and P. Gentine. "Enforcing analytic constraints in neural networks emulating physical systems". *Physical Review Letters* 126.9 (2021). P. 098302.

[5] F. M. Defersha. "A comprehensive mathematical model for hybrid flexible flowshop lot streaming problem". *International Journal of Industrial Engineering Computations* 2.2 (2011). Pp. 283–294. issn: 19232926. doi: 10.5267/j.ijiec.2010.07.006.

[6] Y. C. Fonseca-Reyna, Y. Martinez-Jimenez, A. V. Cabrera, and E. A. R. Sanchez. "Optimization of heavily constrained hybrid-flexible flowshop problems using a multi-agent reinforcement learning approach". *Investigación operacional* 40.1 (2019). issn: 0257-4306.

[7] R. Garg, V. K. Bg, G. Carneiro, and I. Reid. "Unsupervised CNN for single view depth estimation: Geometry to the rescue". *European conference on computer vision*. Springer. 2016, pp. 740–756.

[8] J. N. D. Gupta. "Two-Stage, Hybrid Flowshop Scheduling Problem". *The Journal of the Operational Research Society* 39.4 (1988). Pp. 359–364. issn: 01605682, 14769360. url: http://www.jstor.org/ stable/2582115 (visited on 05/17/2022).

[9] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2022. url: https://www.gurobi.com.

[10] S. Hochreiter and J. Schmidhuber. "Long Short-term Memory". *Neural computation* 9 (Dec. 1997). Pp. 1735–80. doi: 10.1162/neco.1997.9.8.1735.

[11] J. J. Hopfield and D. W. Tank. ""Neural" computation of decisions in optimization problems". *Biological cybernetics* 52.3 (1985). Pp. 141–152. issn: 0340-1200. doi: 10.1007/BF00339943.

[12] L. Jing and Y. Tian. "Self-supervised visual feature learning with deep neural networks: A survey". *IEEE transactions on pattern analysis and machine intelligence* (2020).

[13] A. W. de Jong, J. I. U. Rubrico, M. Adachi, T. Nakamura, and J. Ota. "Big data in automation: Towards generalized makespan estimation in shop scheduling problems". eng. *2017 13th IEEE Conference on Automation Science and Engineering* (CASE). IEEE, 2017, pp. 1516–1521. isbn: 9781509067817.

[14] J. D. Lee, Q. Lei, N. Saunshi, and J. Zhuo. "Predicting what you already know helps: Provable self-supervised learning". *arXiv preprint arXiv:2008.01064* (2020).

[15] B. Li, W.-f. Li, and S. Voß. "Modeling Container Terminal Scheduling Systems as Hybrid Flow Shops with Blocking Based on Attributes". *Logistik Management*. Ed. by S. Voß, J. Pahl, and S. Schwarze. Heidelberg: Physica-Verlag HD, 2009, pp. 413–434. isbn: 978-3-7908-2361-5. doi: 10 . 1007 / 978 - 3 -7908-2362-2_21.

[16] F.-T. Lin, C.-Y. Kao, and C.-C. Hsu. "Applying the genetic approach to simulated annealing in solving some NP-hard problems". *IEEE Transactions on Systems, Man, and Cybernetics* 23.6 (1993). Pp. 1752–1767. doi: 10.1109/21.257766.

[17] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang. "Self-supervised learning: Generative or contrastive". *IEEE Transactions on Knowledge and Data Engineering* (2021).

[18] M. Lombardi and M. Milano. "Boosting Combinatorial Problem Modeling with Machine Learning" (2018). Pp. 1270–1276. doi: 10.24963/ijcai.2018/177. url: http://arxiv.org/pdf/1807.05517v1.

[19] L. Meng, C. Zhang, X. Shao, B. Zhang, Y. Ren, and W. Lin. "More MILP models for hybrid flow shop scheduling problem and its extended problems". *International Journal of Production Research* 58.13 (2020). Pp. 3905–3930. issn: 0020-7543. doi: 10.1080/00207543.2019.1636324.

[20] A. Milan, S. H. Rezatofighi, R. Garg, A. Dick, and I. Reid. "Data-driven approximations to NP-hard problems". *Thirty-first AAAI conference on artificial intelligence*. 2017.

[21] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". *Journal of Computational Physics* 378 (2019). Pp. 686–707.

[22] S. F. Roselli, K. Bengtsson, and K. Åkesson. "SMT Solvers for Job-Shop Scheduling Problems: Models Comparison and Performance Evaluation". *2018 IEEE 14th International Conference on Automation Science and Engineering* (CASE). 2018, pp. 547–552. doi: 10.1109/COASE.2018.8560344.

[23] R. Ruiz and J. A. Vázquez-Rodríguez. "The hybrid flow shop scheduling problem". *European Journal of Operational Research* 205.1 (2010). Pp. 1–18. issn: 03772217. doi: 10.1016/j.ejor.2009.09.024.

[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[25] R. Stewart and S. Ermon. "Label-free supervision of neural networks with physics and domain knowledge". *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[26] L. Tang, W. Liu, and J. Liu. "A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment". *Journal of Intel ligent Manufacturing* 16.3 (2005). Pp. 361–370. issn: 0956-5515. doi: 10.1007/s10845-005-7029-0.

[27] R. Tavakkoli-Moghaddam, S. Fatemi-Anaraki, D. Abdolhamidi, and B. Vahedi-Nouri. "Integrated Waterway Scheduling, Berth Allocation and Quay Crane Assignment Problem by Using a Hybrid Flow Shop Concept". eng. *2019 International Conference on Industrial Engineering and Systems Management (IESM)*. IEEE, 2019, pp. 1–5. isbn: 1728115663.

[28] F. Wang and H. Liu. “Understanding the behaviour of contrastive loss”. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2495–2504.

[29] H. Wang, V. Jacob, and E. Rolland. “Design of efficient hybrid neural networks for flexible flow shop scheduling”. *Expert systems* 20.4 (2003). Pp. 208–231.

[30] R. Wang and R. Yu. “Physics-Guided Deep Learning for Dynamical Systems: A Survey”. *arXiv preprint arXiv:2107.01272* (2021).

[31] F. Yalaoui and C. Chu. “New exact method to solve the P m/rj /ΣCj schedule problem”. *International Journal of Production Economics* 100 (Feb. 2006). Pp. 168–179. doi: 10.1016/j.ijpe.2004.11.002.

[32] Yi Wu, Min Liu, and Cheng Wu. “A genetic algorithm for solving flow shop scheduling problems with parallel machine and special procedure constraints”. *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693)*. Vol. 3. 2003, 1774–1779 Vol.3. doi: 10.1109/ICMLC.2003.1259784.

[33] M. Zacharias. “Combining heuristics and machine learning for hybrid flow shop scheduling problems”. PhD thesis. Duisburg and Essen.

[34] M. Zacharias, A. Tonnius, and J. Gottschling. “Machine Learning in Hybrid Flow Shop Scheduling with Unrelated Machines”. *2019 International Conference on Industrial Engineering and Systems Management (IESM)*. 2019, pp. 1–6. doi: 10.1109/IESM45758.2019.8948113.