Data-driven Algorithm for Scheduling with Total Tardiness

Michal Bouška[1,2][a], Antonín Novák[1,2][b], Přemysl Šůcha[1][c], István Módos[1,2][d] and Zdeněk Hanzálek[1][e]

[1]*Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Jugoslávských partyzánů 1580/3, Prague, Czech Republic*

[2]*Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Control Engineering, Karlovo náměstí 13, Prague, Czech Republic*

Keywords: Single Machine Scheduling, Total Tardiness, Data-driven Method, Deep Neural Networks.

Abstract: In this paper, we investigate the use of deep learning for solving a classical $\mathcal{NP}$-hard single machine scheduling problem where the criterion is to minimize the total tardiness. Instead of designing an end-to-end machine learning model, we utilize well known decomposition of the problem and we enhance it with a data-driven approach. We have designed a regressor containing a deep neural network that learns and predicts the criterion of a given set of jobs. The network acts as a polynomial-time estimator of the criterion that is used in a single-pass scheduling algorithm based on Lawler's decomposition theorem. Essentially, the regressor guides the algorithm to select the best position for each job. The experimental results show that our data-driven approach can efficiently generalize information from the training phase to significantly larger instances (up to 350 jobs) where it achieves an optimality gap of about 0.5%, which is four times less than the gap of the state-of-the-art NBR heuristic.

1 INTRODUCTION

The classical approaches for solving combinatorial problems have several undesirable properties. First, solving instances of an $\mathcal{NP}$-Hard problem to optimality consumes an unfruitful amount of computational time. Second, there is no well-established method how to utilize the solved instances for improving the algorithm or recycling the solutions for the unseen instances. Finally, the development of efficient heuristic rules requires a substantial time devoted to the research. To address these issues, we investigate the use of deep learning which is able to derive knowledge from the already solved instances of a classical scheduling $\mathcal{NP}$-hard Single Machine Total Tardiness Problem (SMTTP) and estimate the optimal value of an unseen SMTTP instance. This is the first successful application of deep learning to the scheduling problem; we successfully integrated the deep neural network into a known decomposition algorithm and outperformed the state-of-the-art heuristics. With this, we are able to solve instances with hundreds of jobs, which is significantly more than, e.g., an end-to-end approach (Vinyals et al., 2015) that solves Traveling Salesman Problem with about 50 nodes. Our proposed approach outperforms the state-of-the-art heuristic for SMTTP.

[a] https://orcid.org/0000-0002-8034-2531
[b] https://orcid.org/0000-0003-2203-4554
[c] https://orcid.org/0000-0003-4895-157X
[d] https://orcid.org/0000-0003-4692-1625
[e] https://orcid.org/0000-0002-8135-1296

1.1 Problem Statement

The combinatorial problem studied in this paper is denoted as $1||\sum T_j$ in Graham's notation of scheduling problems (Graham et al., 1979). Let $J = \{1,\dots,n\}$ be a set of jobs that has to be processed on a single machine. The machine can process at most one job at a time, the execution of the jobs cannot be interrupted, and all the jobs are available for processing at time zero. Each job $j \in J$ has processing time $p_j \in \mathbb{Z}_{\geq 0}$ and due date $d_j \in \mathbb{Z}_{\geq 0}$. Let $\pi : \{1,\dots,n\} \mapsto \{1,\dots,n\}$ be a bijective function representing a sequence of the jobs, i.e., $\pi(k) \in J$ is the job at position k in sequence π. For a given sequence π, tardiness of job $\pi(k)$ is defined as $T_{\pi(k)} = \max\left(0, \left(\sum_{k'=1}^{k} p_{\pi(k')}\right) - d_{\pi(k)}\right)$. The goal of the scheduling problem is to find a sequence which minimizes the total tardiness, i.e., $\sum_{j\in J} T_j$. The problem is proven to be $\mathcal{NP}$-hard (Du and Leung, 1990).

In the rest of the paper, we use the following two definitions to describe the ordering of the jobs:

1. earliest due date (edd): if $1 \leq j < j' \leq n$ then either (i) $d_j < d_{j'}$ or (ii) $d_j = d_{j'} \wedge p_j \leq p_{j'}$,
2. shortest processing time (spt): if $1 \leq j < j' \leq n$ then either (i) $p_j < p_{j'}$ or (ii) $p_j = p_{j'} \wedge d_j \leq d_{j'}$.

1.2 Contribution and Outline

This paper addresses a single machine total tardiness scheduling problem using a machine learning technique. Unlike some existing works, for example, (Vinyals et al., 2015), we do not purely count on machine learning, but we combine it with the known approaches from OR domain. The advantage of our approach is that it can extract specific knowledge from data, i.e., already solved instances, and use it to solve the new ones. The experimental results show two important observations. First, our algorithm outperforms the state-of-the-art heuristic (Holsenback and Russell, 1992), and it also provides better results on some instances than the exact state-of-the-art approach (Garraffa et al., 2018) with a time limit. Second, the proposed algorithm is capable of generalizing the acquired knowledge to solve instances that were not used in the training phase and also significantly differ from the training ones, e.g., in the number of jobs or the maximal processing time of jobs.

The rest of the paper is structured as follows. In Section 2, we present a review of literature for SMTTP and combination of operations research (OR) and *machine learning* (ML). Section 3 describes our approach integrating a regressor into the decomposition and analyzes it's time complexity. We present results for standard benchmark instances for SMTTP in Section 4. Finally, the conclusion is drawn in Section 5.

2 RELATED WORK

The first part of the literature overview is based on the extensive survey addressing SMTTP published by (Koulamas, 2010) which we further extend with the description of the current state-of-the-art algorithms. The second part maps existing work in machine learning related to solving combinatorial problems.

2.1 SMTTP

In 1977 it was shown by Lawler (Lawler, 1977) that the weighted single machine total tardiness problem is $\mathcal{NP}$-Hard. However, it took more than a decade to prove that the unweighted variant of this problem is $\mathcal{NP}$-Hard as well (Du and Leung, 1990).

Lawler (Lawler, 1977) proposes a pseudo-polynomial (in the sum of processing times) algorithm for solving SMTTP. The algorithm is based on a decomposition of the problem into subproblems. The decomposition selects the job with the maximum processing time and tries all the positions following its original position in the *edd* order. For each position, two subproblems are generated; the first subproblem contains all the jobs preceding the job with the maximum processing time and the second subproblem contains all the jobs following the job with the maximum processing time. In addition, Lawler introduces rules for filtering the possible positions of the job with the maximum processing time. This algorithm can solve instances with up to one hundred jobs. F. Della Croce *et al.* (Della Croce et al., 1998) proposed a *spt* decomposition which selects the job with the minimal due date and tries all the positions preceding its original position in *spt* order. Similarly as with the Lawler's decomposition, two subproblems are generated where the first subproblem contains all the jobs preceding the job with the minimal due date time and the second subproblem contains all the jobs following the job with the minimal due date. F. Della Croce *et al.* combined both *edd* decomposition and *spt* decomposition together, this presented algorithm is able to solve instances with up to 150 jobs. Finally, Szwarc *et al.* (Szwarc et al., 1999) integrate the double decomposition from (Della Croce et al., 1998) and a split procedure from (Szwarc and Mukhopadhyay, 1996). This algorithm was the state-of-the-art method for a long time with the ability to solve instance with up to 500 jobs.

Recently, Garraffa *et al.* (Garraffa et al., 2018) proposed Total Tardiness Branch-and-Reduce Algorithm (TTBR), which infers information about nodes of the search tree and merges nodes related to the same subproblem. This is the fastest known exact algorithm for SMTTP to this date and is able to solve instances with up to 1300 jobs.

Exact algorithms, such as the ones mentioned above, have very large computation times while the optimal solution is rarely needed in practice. Hence, heuristic algorithms are often more practical. Existing heuristics algorithm can be categorized into the following three major groups.

The first group of heuristics creates a job order and schedule the jobs according this order, i.e., list scheduling algorithms. There are various methods for creating a job order. The easiest one is to sort job by Earliest Due Date rule (*edd*). A more efficient algorithm called NBR was proposed in (Holsenback and Russell, 1992). NBR is a constructive local search heuristic which starts with job set J sorted

by *edd* and constructs the schedule from the end by exchanging two jobs. Panwalkar *et al.* (Panwalkar et al., 1993) proposes constructive local search heuristic PSK, which starts with job set J sorted by *spt* and constructs the schedule from the start by exchanging two jobs. Russel and Holsenback (Russell and Holsenback, 1997) compares PSK and NBR heuristic, and conducted that neither heuristic is inferior to another one. However, NBR finds a better solutions in more cases. The second group of heuristics is based on Lawler decomposition rule (Lawler, 1977). In this case, heuristic evaluates each child of the search tree node and the most promising child is expanded. This heuristic approach is evaluated in (Potts and Van Wassenhove, 1991) with *edd* heuristic as a guide for the search. The third group of heuristics are metaheuristics. (Potts and Van Wassenhove, 1991), (Antony and Koulamas, 1996), (Ben-Daya and Al-Fawzan, 1996) present simulated annealing algorithm for SMTTP. Genetic algorithms applied to SMTTP are described in (Dimopoulos and Zalzala, 1999), (Süer et al., 2012), whereas (Bauer et al., 1999), (Cheng et al., 2009) propose to use ant colony optimization for this scheduling problem. All the reported results in the previous studies are for instance sizes up to 100 jobs. However, these instances are solvable by the current state-of-the-art exact algorithm in a fraction of second.

2.2 Machine Learning Integration to Combinatorial Optimization Problems

The integration of ML to combinatorial optimization problems has several difficulties. As first, ML models are often designed with feature vectors having predefined fixed size. On the other hand, instances of scheduling problems are usually described by a variable number of features, e.g., variable number of jobs. This issue can be addressed by recurrent networks and, more recently, by encoder-decoder type of architectures. Vinyals (Vinyals et al., 2015) applied an architecture called Pointer Network that, given a set of graph nodes, outputs a solution as a permutation of these nodes. The authors applied the Pointer Network to Traveling Salesman Problem (TSP), however, this approach for TSP is still not competitive with the best classical solvers such as Concorde (Applegate et al., 2006) that can find optimal solutions to instances with hundreds nodes in a fraction of second. Moreover, the output from the Pointer Network needs to be corrected by the beam-search procedure, which points out the weaknesses of this end-to-end approach. Pointer Network has achieved optimality gap around 1% for instance with 20 nodes after performing beam-search.

Second difficulty with training a ML model is with acquisition of training data. Obtaining one training instance usually requires solving a problem of the same complexity like the original problem itself. This issue can be addressed with reinforcement learning paradigm. Deudon *et al.* (Deudon et al., 2018) used encoder-decoder architecture trained with REINFORCE algorithm to solve 2D Euclidean TSP with up to 100 nodes. It is shown that (i) repetitive sampling from the network is needed, (ii) applying well-known 2-opt heuristic on the results still improves the solution of the network, and (iii) both the quality and runtime are worse than classical exact solvers. Similar approach is described in (Kool and Welling, 2018) which, if it is treated as a greedy heuristic, beats weak baseline solutions (from the operations research perspective) such as Nearest Neighbor or Christofides algorithm on small instances. To be competitive in terms of quality with more relevant baselines such as Lin-Kernighan heuristics, they perform multiple sampling from the model and output the best solution. Moreover, they do not directly compare their approach with state-of-the-art classical algorithms while admitting that off-the-shelf Integer Programming solver Gurobi solves optimally their largest instances within 1.5 s.

Khalil *et al.* (Khalil et al., 2017) present an interesting approach for learning greedy algorithms over graph structures. The authors show that their S2V-DQN model can obtain competitive results on MAX-CUT and Minimum Vertex Cover problems. For TSP, S2V-DQN performs about the same as 2-opt heuristics. Unfortunately, the authors do not compare runtimes with Concorde solver.

Milan *et al.* (Milan et al., 2017) presents a data-driven approximation of solvers for $\mathcal{NP}$-hard problems. They utilized a *Long Short-Term Memory (Hochreiter and Schmidhuber, 1997)* (LSTM) network with a modified supervised setting. The reported results on the Quadratic Assignment Problem show that the network's solutions are worse than general purpose solver Gurobi while having the essentially identical runtime.

Integration of ML with scheduling problems has received a little attention so far. Earlier attempts of integrating neural networks with job-shop scheduling are (Zhou et al., 1991) and (Jain and Meeran, 1998). However, their computational results are inferior to the traditional algorithms, or they are not extensive enough to assess their quality. An alternative use of ML in scheduling domain is focused on the criterion function of the optimization problems. For example, authors in (Václavík et al., 2016) address a nurse ros-

tering problem and improved the evaluation of the solutions' quality without calculating their exact criterion values. They propose a classifier, implemented as a neural network, able to determine whether a certain change in a solution leads to a better solution or not. This classifier is then used in a local search algorithm to filter out solutions having a low chance to improve the criterion function. Nevertheless, this approach is sensitive to changes in the problem size, i.e., the size of the schedule of nurses. If the size is changed, a new neural network must be trained. Another method, which does not directly predict a solution to the given instance, is proposed in (Václavík et al., 2018). In this case, an online ML technique is integrated into an exact algorithm where it acts as a heuristic. Specifically, the authors use regression for predicting the upper bound of a pricing problem in a Branch-and-Price algorithm. Correct prediction leads to faster computation of the pricing problem while incorrect prediction does not affect the optimality of the algorithm. This method is not sensitive to the change of the problem size; however, it is designed specifically for the Branch-and-Price approach and cannot be generalized to other approaches.

3 PROPOSED DECOMPOSITION HEURISTIC ALGORITHM

In this section, we introduce *Heuristic Optimizer using Regression-based Decomposition Algorithm* (HORDA) for Single Machine Total Tardiness Problem (SMTTP). This heuristic effectively combines the well-know properties of SMTTP and the data-driven approach. Moreover, this paper proposes a methodology for designing data-driven heuristics for scheduling problems where good estimator of the optimization criterion can be obtained to guide the search.

This section is structured as follows. First of all, we summarize decompositions used in the algorithm. As the second, we describe HORDA. Next we continue by discussing the architecture of the regressor, its integration into SMTTP decompositions, and describe the training of the neural network. Finally, we analyze the time complexity of HORDA algorithm.

3.1 SMTTP Decompositions

Firstly, we describe two different decomposition approaches for SMTTP. The reason is that every state-of-the-art exact algorithm for SMTTP is based on these two decompositions.

First decomposition, introduced by Lawler (Lawler, 1977), uses *edd* (earliest due date) order in which it selects position for job $j^{p\text{-}max}$, i.e., a job with the maximal processing time from job set J (in case of tie, $j^{p\text{-}max}$ is the job with the larger index in *edd* order). Lawler proves that there exists position $k \in \{j^{p\text{-}max},\dots,n\}$ in the *edd* order such that at least one optimal solution exists where $j^{p\text{-}max}$ is preceded by all jobs $\{1,\dots,k\} \setminus \{j^{p\text{-}max}\}$ and followed by all jobs $\{k+1,\dots,n\}$. Let us denote set of positions $\{j^{p\text{-}max},\dots,n\}$ as K^{edd}. This property leads to the following exact decomposition algorithm. First, let $P^{edd} : \mathcal{P}(J) \times [1,\dots,n] \to \mathcal{P}(J)$ and $F^{edd} : \mathcal{P}(J) \times [1,\dots,n] \to \mathcal{P}(J)$ be functions which for job set J and position k return subproblem with jobs $\{1,\dots,k\} \setminus \{j^{p\text{-}max}\}$ and $\{k+1,\dots,n\}$, respectively. Where $\mathcal{P}(J)$ is powerset of J. Thus, for each eligible position $k \in \{j^{p\text{-}max},\dots,n\}$, the problem is decomposed into two subproblems defined by $P^{edd}(J,k)$ and $F^{edd}(J,k)$ such that jobs $j^{p\text{-}max}$ is neither in P^{edd} nor in F^{edd}. Let $Z(J)$ denote the optimal criterion value for job set J computed as

$$Z(J) = \min_{k \in K^{edd}} Z(J,k)\,, \tag{1}$$

where

$$Z(J,k) = Z\left(P^{edd}(J,k)\right) + \max\left(0, p_k - d_k + \sum_{j \in P^{edd}(J,k)} p_j\right) + Z\left(F^{edd}(J,k)\right). \tag{2}$$

The optimal solution to the instance is found by recursively selecting the position k with the minimal criterion Z.

The second decomposition(Della Croce et al., 1998) introduced by Della Croce *et al.* uses *spt* order in which it selects position for job $j^{d\text{-}min}$. We refer to this decomposition as *spt* decomposition. Let us define $j^{d\text{-}min}$ job as a job with the minimal due date from job set J (in case of tie, $j^{d\text{-}min}$ is the job with the smaller index in the *spt* order). Similarly as in the *edd* decomposition proposed by Lawler, Della Croce *et al.* (Della Croce et al., 1998) prove that for job $j^{d\text{-}min}$ in *spt* order there exists position $k \in \{1,\dots,j^{d\text{-}min}\}$ such that in at least one optimal solution $j^{d\text{-}min}$ is preceded by job set generated by function $P^{spt} : \mathcal{P}(J) \times [1,\dots,n] \to \mathcal{P}(J)$. $P^{spt}(J,k)$ returns job set with first k jobs selected from $\{1,\dots,j^{d\text{-}min}\}$ which are then sorted by *edd*. Job $j^{d\text{-}min}$ is followed by job set $F^{spt} : \mathcal{P}(J) \times [1,\dots,n] \to \mathcal{P}(J)$ with all the others jobs. The set of positions $k \in \{1,\dots,j^{d\text{-}min}\}$ is denoted as K^{spt}. One may use the *spt* decomposition in the same recursive way as *edd* decomposition to find the optimal solution.

The efficiency of both decomposition approaches is significantly influenced by the branching factor. Here, the branching factor is equal to the number of eligible positions where job $j^{p\text{-}max} \in K^{edd}$ ($j^{d\text{-}min} \in K^{spt}$) can be placed. The number of eligible positions can be reduced by filtering rules described in (Lawler, 1977) and (Szwarc et al., 1999). Let us denote that $\overline{K^{edd}}$, $\overline{K^{spt}}$ are the sets K^{edd}, K^{spt} filtered by rules from (Szwarc et al., 1999) respectively.

3.2 HORDA

Even though algorithms using decompositions proposed in (Lawler, 1977) and (Della Croce et al., 1998) are very efficient, their time complexity exponentially grows with the number of jobs. Our HORDA algorithm avoids this exponential growth by pruning the search tree ruled by the polynomial-time estimation of (2) produced by a neural network. The estimations of $Z(J,k)$ and $Z(J)$ are denoted as $\widehat{Z}(J,k)$ and $\widehat{Z}(J)$, respectively.

HORDA algorithm is outlined in Algorithm 1. To increase the efficiency of the solution space search, our HORDA algorithm combines the power of both decompositions (Lawler, 1977) and (Della Croce et al., 1998) in the following way. The HORDA algorithm generates (lines 5 and 6) two sets of eligible positions $\overline{K^{edd}}$ and $\overline{K^{spt}}$ by either *edd* or *spt* decomposition which are filtered by state-of-the-art rules (Szwarc and Mukhopadhyay, 1996). Then, the set with the minimal cardinality is selected (lines 7 - 12) for the recursive expansion; we refer to the selected set as K.

After obtaining positions set K, the algorithm greedily selects k^* position having the minimal estimation $\widehat{Z}$ (line 13). Next, the algorithm recursively explores job sets $P(J,k^*)$ and $F(J,k^*)$, and resulting partial sequences are stored as vectors *before* and *after* respectively (lines 14 and 15). Finally, the algorithm merges $\{before, k^*, after\}$ into one sequence, which is returned as the resulting schedule (line 17). Note that job sets with less or equal than 5 jobs are solved to optimality by an exact solver (Total Tardiness Branch-and-Reduce Algorithm (TTBR)) instead of the decomposition.

3.3 Regressor

The proposed HORDA algorithm utilizes the regressor estimation in the decomposition to guide the search by selecting position k^* that minimizes the estimated criterion $\widehat{Z}$ (see line 13). The quality of the estimation significantly affects the quality of the found solutions. However, HORDA algorithm is not sensitive to absolute error of the estimation, instead, it's relative error is important. Therefore, the proposed regressor is based on neural networks that are known to be successful for problems sensitive to relative error, for example Google (Silver et al., 2016) applied them to predict a policy in Monte Carlo Tree Search to solve game of Go.

Algorithm 1: Decomposition heuristic search (HORDA).

```
   Data: J
   Result: HORDA ordered jobs
1  Function HORDA (J):
2      if |J| ≤ 1 then
3          return toSequence(J)
4      end
       /* Generate edd and spt positions
          with respect to the filtering
          rules                              */
5      K^edd ← genEDDPos(J)
6      K^spt ← genSPTPos(J)
7      if |K̄^edd| ≤ |K̄^spt| then
8          K ← K̄^edd, P ← P^edd, F ← F^edd
9      end
10     else
11         K ← K̄^spt, P ← P^spt, F ← F^spt
12     end
       /* Where Ẑ is computed by
          regressor.                         */
13     k* ← argmin_{k∈K} (Ẑ(P(J,k)) + max(0, p_k −
         d_k + Σ_{j∈P(J,k)} p_j) + Ẑ(F(J,k)))
14     before ← HORDA (P(J,k*))
15     after ← HORDA (F(J,k*))
       /* join sequences into one            */
16     order ← (before, k*, after)
17     return order
```

The architecture of our regressor using neural network is illustrated in Figure 1. It has two main parts. The first one is the normalization of the input data, described in Section 3.3.1. The second one is the neural network, explained in Section 3.3.2.

3.3.1 Input Data Preprocessing

The speed of training and quality of the neural network is affected by the preprocessing of the input instances. There are two main reasons for the preprocessing denoted as *Norm* in Figure 1. Firstly, preprocessing of the input instance normalizes the instances, and thus reduces the variability of input data denoted $\boldsymbol{X}$. For example, two neural network inputs differing only in job order are, in fact, the same. Secondly, numerical stability of the computation is improved by

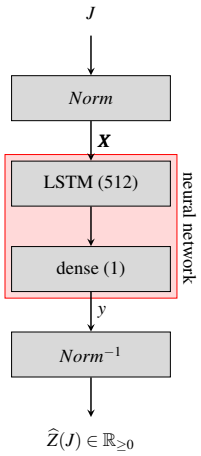

Figure 1: Regressor architecture.

the preprocessing. In our regression architecture, the preprocessing has three main parts:

1. sorting of the input: we performed preliminary experiments with various sorting options such as *edd*, *spt*, reversed *edd* and reversed *spt*, among which *edd* performed the best.
2. normalization of the input: the processing times and due dates are divided by the sum of the processing times in the instance.
3. appending additional features to the neural network: each job has one additional feature which is its position in $\boldsymbol{X}$ divided by the number of the jobs.

The best practice in the neural network training is to normalize value that is estimated by the neural network, denoted as y in Figure 1. In the training phase, the associated optimal criterion value of each instance is divided by the sum of the processing times. Alternatively, we evaluated one additional criterion normalization $Z/\left(n \cdot \sum_{j \in J} p_j\right)$. However, it performed poorly. In the HORDA the estimation produced by the neural network has to be denormalized by the inverse transformation ($Norm^{-1}$ in Figure 1) to obtain the actual estimation of the total tardiness.

3.3.2 Neural Network

The input data for our neural network have several similarities as the input data for *nature language processing* (NLP) problems. Firstly, as well as NLP, our data can be arbitraly large, i.e., the size of job set J is unbounded; similarly, sentences in NLP can be arbitrarily long. In other research fields, such as computer vision, this issue is mitigated by scaling the feature vectors to a fixed length. However, there is no simple and general way for scheduling problems how to aggregate multiple jobs into one without losing necessary information. Therefore, we use another technique of dealing with the varying length of the input which are *recurrent neural network* (RNN) (Sundermeyer et al., 2012).

Our neural network for the criterion estimation of J consists of two parts (the red box in Figure 1). The first layer is *Long Short-Term Memory (Hochreiter and Schmidhuber, 1997)* (LSTM), which receives job set J as the input. The input $\boldsymbol{X}$ is a sequence of features $\boldsymbol{x}_j$ for every job $j \in J$. Each feature vector $\boldsymbol{x}_j$ consists of p_j and d_j with additional features described below. The output of the last LSTM step is passed into a dense layer which produces estimation y of the criterion for $\boldsymbol{X}$.

3.4 Time Complexity of HORDA

In this section, we present the worst-case runtime of HORDA. The most time consuming part of HORDA is the estimation of $\widehat{Z}(J)$ by the regressor. The LSTM layer produces $\widehat{Z}(J)$ in $O(n)$ time and HORDA algorithm evaluates the regressor $2 \cdot n$ times to select position k^* from K. Thus, the evaluation of all the estimations for K takes $O(n^2)$. In the worst-case, when decomposition repetitively removes one job, HORDA algorithm makes $O(n)$ selections of position k^*. Therefore, the worst-case time complexity of HORDA algorithm is $O(n^3)$. However, we note that the constants present in the asymptotic complexit are fairly low. Hence, it is efficient in practice, as well.

4 EXPERIMENTAL RESULTS

In this section, we present the experimental results. Firstly, we describe the training of the neural network, also with the acquisition of a training dataset. Secondly, we describe the generation of the benchmark instances. Then we compare our HORDA heuristic with the state-of-the-art heuristic NBR (Holsenback

Table 1: Mean TTBR(Garraffa et al., 2018) runtimes in seconds with respect to instance parameters for $n \in \{5, \ldots, 500\}$ and $p_{max} = 100$. For parameters relative range of due dates (*rdd*), and the average tardiness factor (*tf*).

rdd/*tf*	0.2	0.4	0.6	0.8	1.0
0.2	0.07	2.16	**5.16**	1.64	0.04
0.4	0.04	0.36	1.64	0.05	0.04
0.6	0.04	0.06	0.47	0.04	0.04
0.8	0.04	0.04	0.07	0.04	0.04
1.0	0.04	0.04	0.04	0.04	0.04

Table 2: Optimality gap of HORDA, TTBR(Garraffa et al., 2018) and NBR(Holsenback and Russell, 1992) on instances with $p_{max} = 100$.

n	**TTBR**	**NBR**		**HORDA+NBR**		**HORDA+NN**	
± 25	time [s]	gap [%]	time [s]	gap [%]	time [s]	gap [%]	time [s]
225	1.05 ± 2.90	1.98 ± 0.58	0.06 ± 0.01	1.17 ± 0.47	1.19 ± 0.42	$\mathbf{0.58} \pm 0.30$	5.03 ± 8.16
275	2.45 ± 4.19	2.12 ± 0.54	0.09 ± 0.02	1.31 ± 0.44	1.91 ± 0.62	$\mathbf{0.57} \pm 0.28$	6.89 ± 9.62
325	4.72 ± 4.09	2.20 ± 0.50	0.12 ± 0.02	1.39 ± 0.43	2.87 ± 0.90	$\mathbf{0.57} \pm 0.37$	9.25 ± 11.29
375	8.42 ± 4.75	2.27 ± 0.49	0.17 ± 0.03	1.46 ± 0.44	4.15 ± 1.31	$\mathbf{1.23} \pm 0.63$	14.61 ± 13.52
425	14.42 ± 8.06	2.34 ± 0.46	0.21 ± 0.04	$\mathbf{1.55} \pm 0.41$	5.52 ± 1.71	1.71 ± 0.65	20.60 ± 17.00

Table 3: Optimality gap of heuristics on instances with $p_{max} = 5000$.

n	**TTBR**	**TTBR10s**	**NBR**	**HORDA+NBR**	**HORDA+NN**	
± 25	time [s]	gap [%]	gap [%]	gap [%]	gap [%]	time [s]
225	10.66 ± 9.20	$\mathbf{0.17} \pm 0.31$	1.91 ± 0.60	1.10 ± 0.48	0.58 ± 0.27	3.58 ± 0.81
275	40.36 ± 32.24	0.77 ± 0.69	2.00 ± 0.54	1.20 ± 0.45	$\mathbf{0.55} \pm 0.27$	4.89 ± 1.02
325	92.30 ± 56.39	1.28 ± 0.86	2.27 ± 0.53	1.36 ± 0.47	$\mathbf{0.53} \pm 0.33$	6.61 ± 1.50
375	212.69 ± 122.14	1.87 ± 0.87	2.39 ± 0.47	1.50 ± 0.48	$\mathbf{1.09} \pm 0.60$	10.32 ± 2.18
425	488.76 ± 265.88	2.64 ± 0.87	2.32 ± 0.44	$\mathbf{1.52} \pm 0.41$	1.73 ± 0.64	14.96 ± 2.00

and Russell, 1992) and exact algorithm TTBR (Garraffa et al., 2018). Finally, we discuss the advantages of our proposed heuristic.

Experiments were run on a single-core of the Xeon(R) Gold 6140 processor with a memory limit set to 8GB of RAM. HORDA and NBR algorithms were implemented in Python, and the neural network is trained in Tensor Flow 1.14 on Nvidia GTX 1080 Ti. Source codes of TTBR algorithm were provided by authors of (Garraffa et al., 2018) and it is implemented in C.

4.1 Neural Network Training

We trained the neural network with Adam optimizer, with learning rate set to 0.0001, early stop with patience equals to 5. Size of the LSTM layer is set to 512. For the neural network training, we generated instances by scheme introduced by Potts and Wassenhove (Potts and Wassenhove, 1982). The scheme uses two parameters; relative range of due dates (*rdd*), and the average tardiness factor (*tf*). The values of *rdd*, *tf* typically used in the literature are $rdd, tf \in \{0.2, 0.4, 0.6, 0.8, 1\}$. For each such *rdd*, *tf* and $n \in \{5, \ldots, 250\}$, we generated 5000 instances. Therefore, the whole training dataset consists of 30625000 instances in total. Since we use a supervised learning to train the neural network, we need optimal criterion values that acts as labels.

It is easy to see that the dataset is enormous, and it is necessary to solve millions of SMTTP instances. However, this is not an issue since a substantial amount of the instances can be solved within a fraction of a second. Moreover, the dataset can be cheaply generated in the cloud, e.g., on the Amazon EC2 cloud, the cost of generating the dataset is around 800$ and takes only ten days, which is significantly cheaper compared to the cost of a human expert developing a heuristic algorithm.

Furthermore, it is important to stress that our neural network is able to generalize to larger instance than used in the training. Therefore, it is possible to train the neural network on smaller instances and solve larger ones both in terms of the number of jobs and their parameters.

4.2 Benchmark Instances

Benchmark instances used in this paper were generated in the manner suggested by Potts and Van Wassenhove in (Potts and Van Wassenhove, 1991) and used in Section 4.1. Potts and Van Wassenhove generate processing times of jobs uniformly on the interval from 1 to 100. We define maximal processing time p_{max} and generate processing time of jobs in instance uniformly on the interval from 1 to p_{max}. For $p_{max} = 100$ and $n \in \{5, \ldots, 500\}$, we generated 25 sets of benchmarks differing in *rdd* and *tf*. Then those instances were solved by TTBR algorithm. Table 1 shows average runtimes in seconds over $(rdd, tf) \in \{0.2, 0.4, 0.6, 0.8, 1\}^2$. The results imply, that the hardest instances occur for $rdd = 0.2$ and $tf = 0.6$ (highlighted in Table 1 in bold), therefore our experiments concentrate on them. Nevertheless, it is important to stress that the neural network is trained on the whole range of values (rdd, tf). First, we do not want

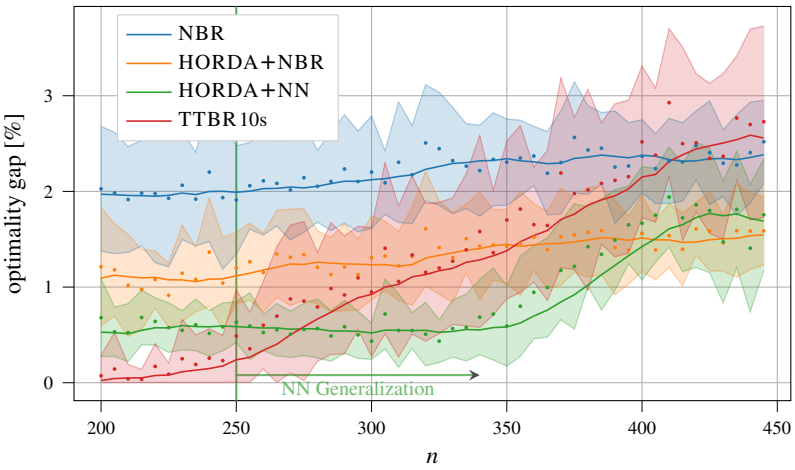

Figure 2: Optimality gap on instances with $p_{max} = 5000$.

the algorithm to be limited to a specific class of instances. Second, since our algorithm uses the decompositions, as described in Section 3.1, there is no guarantee that the subproblems have the same (rdd, tf) parameterization as the input instance. In fact, during the run of HORDA, the values of (rdd, tf) in newly emerged subproblems shift from the original ones.

4.3 Comparison with Existing Approaches

In the first experiment, summarized in Table 2, we concentrate on the comparison with NBR heuristic. The benchmark instances used in this experiment were generated with $p_{max} = 100$. Each row in the table represents a set of 200 instances of size from range $[n-25, n+25)$. The optimal solution was obtained by TTBR algorithm. The table compares NBR heuristic with HORDA algorithm where the regressor is substituted by NBR heuristic (denoted HORDA+NBR), and HORDA heuristic with the neural network regressor (denoted HORDA+NN). These three approaches are compared in terms of the average CPU time, and the average quality of solutions, measured by the optimality gap in percent. All values are reported together with their standard deviation.

Results are shown from $n = 200$. For smaller n than 200, TTBR is able to find the optimal solution under a second, and because of this, the results of heuristics are not relevant. The bold values in the table indicate the best result over all the heuristic approaches for the particular set of instances. The results show that HORDA+NN has the best performance in terms of the average optimality gap. In the case of the last data set, the second heuristic HORDA+NBR is slightly better. The reason is that the neural network was trained only on instances with $n \leq 250$. Therefore, one can see that our neural network, used in the regressor, is able to generalize the gained knowledge to instances with $n \leq 400$. On instances with $n \leq 325$, the average optimality gap of HORDA+NN is about 0.5%, which outperforms all other methods. At the same time, we have to admit that the heuristic is slower than TTBR algorithm. Nevertheless, this is true only on instances generated with $p_{max} = 100$. On larger maximum processing time, the CPU time of TTBR is significantly larger as will be seen in the next experiment.

In literature, benchmark instances for SMTTP are usually generated with $p_{max} = 100$, as it was used in the previous experiment. Since SMTTP is applicable in production and grid computing and p_{max} can be much longer in these fields, we introduce the following experiments with maximal processing time p_{max} equal to 5000. Table 3 compares our HORDA+NN and HORDA+NBR heuristics with NBR, TTBR and TTBR with runtime limited to 10 s denoted as TTBR 10s. For TTBR 10s, a 10 s limit is selected with respect to the HORDA+NN algorithm runtime, since the runtime of HORDA+NN on instances with up to $n = 350$ is under 10 s. Please note that the identical regressor as in Table 2 was used, i.e., the regressor was trained only on instances with $p_{max} = 100$. Hence, it demonstrates neural network's ability of generalization outside the training processing time range.

One can observe from Table 2 and Table 3 that the CPU time of HORDA+NN is almost the same for both types of instances. However, this is not true for TTBR where the CPU time is almost 30 times higher for $n = 425$. Also, the CPU time of TTBR is more than 30 times higher for $n = 425$ and $p_{max} = 5000$ compared to HORDA+NN. If the runtime of TTBR is limited to 10 s, then HORDA+NN outperforms TTBR 10s

on larger instances. Moreover, the optimality gap of HORDA+NN is practically the same as in the previous experiment with $p_{max} = 100$.

The same experiment is shown in the form of a graph in Figure 2. It compares the optimality gap of NBR, TTBR with a time limit, HORDA+NBR, and HORDA+NN. The bold lines in the graph represent the moving average (last 5 samples) of optimality gap of each method, and the colored areas represent their standard deviation. HORDA+NN outperforms HORDA+NBR about two times up to instances of size $n = 360$. For instances with $n \geq 405$, HORDA+NBR, is slightly better. In addition, HORDA+NN also outperforms TTBR10s from $n = 265$. Furthermore, HORDA+NN holds the average optimality gap around 0.5% for instances with up to 350 jobs. The same can be observed on instances with $p_{max} = 100$ (see Table 2). Finally, the runtime of TTBR grows exponentially with the growing size of the instance, in contrast to polynomial runtime of HORDA+NN.

Concerning the heuristic using the neural network (HORDA+NN), it is important to stress that for instances with $n > 250$ the network has to generalize the acquired knowledge since it was trained only on instances with $n \leq 250$. This fact is indicated in Figure 2 by a green vertical line. It can be seen that HORDA+NN is able to generalize results to instances having 100 more jobs than instances encountered in the training phase with 50 times larger maximal processing time (instances for the training phase were generated with $p_{max} = 100$).

5 CONCLUSION

To the best of our knowledge, this is the first paper addressing a scheduling problem using deep learning. Unlike the solution used in (Vinyals et al., 2015), which tackled the Traveling Salesman Problem, we combined a state-of-the-art operations research method with a DNN. The experimental results show that our approach provides near-optimal solutions very quickly and is also able to generalize the acquired knowledge to larger instances without significantly affecting the quality of the solutions. Our approach outperforms state-of-the-art heuristic NBR. Our approach is shown to be competitive and in some cases, superior to the previous state-of-the-art algorithms. Hence, we believe that the proposed methodology opens new possibilities for the design of efficient heuristics algorithms.

ACKNOWLEDGEMENTS

The authors want to thank Vincent T'Kindt from Universit de Tours for providing the source code of TTBR algorithm. This work was supported by the European Regional Development Fund under the project AI&Reasoning (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000466).

This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS19/175/OHK3/3T/13.

REFERENCES

Antony, S. R. and Koulamas, C. (1996). Simulated annealing applied to the total tardiness problem. *Control and Cybernetics*, 25:121–130.

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. (2006). Concorde TSP solver.

Bauer, A., Bullnheimer, B., Hartl, R. F., and Strauss, C. (1999). An ant colony optimization approach for the single machine total tardiness problem. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 2, pages 1445–1450. IEEE.

Ben-Daya, M. and Al-Fawzan, M. (1996). A simulated annealing approach for the one-machine mean tardiness scheduling problem. *European Journal of Operational Research*, 93(1):61–67.

Cheng, T. E., Lazarev, A. A., and Gafarov, E. R. (2009). A hybrid algorithm for the single-machine total tardiness problem. *Computers & Operations Research*, 36(2):308–315.

Della Croce, F., Tadei, R., Baracco, P., and Grosso, A. (1998). A new decomposition approach for the single machine total tardiness scheduling problem. *Journal of the Operational Research Society*, 49(10):1101–1106.

Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., and Rousseau, L.-M. (2018). Learning heuristics for the tsp by policy gradient. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 170–181. Springer.

Dimopoulos, C. and Zalzala, A. (1999). A genetic programming heuristic for the one-machine total tardiness problem. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 2207–2214. IEEE.

Du, J. and Leung, J. Y. T. (1990). Minimizing total tardiness on one machine is NP-Hard. *Math. Oper. Res.*, 15(3):483–495.

Garraffa, M., Shang, L., Della Croce, F., and T'Kindt, V. (2018). An exact exponential branch-and-merge algorithm for the single machine total tardiness problem. *Theoretical Computer Science*, 745:133–149.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Holsenback, J. E. and Russell, R. M. (1992). A heuristic algorithm for sequencing on one machine to minimize total tardiness. *Journal of the Operational Research Society*, 43(1):53–62.

Jain, A. S. and Meeran, S. (1998). Job-shop scheduling using neural networks. *International Journal of Production Research*, 36(5):1249–1272.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. (2017). Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358.

Kool, W. and Welling, M. (2018). Attention solves your TSP. *arXiv preprint arXiv:1803.08475*.

Koulamas, C. (2010). The single-machine total tardiness scheduling problem: Review and extensions. *European Journal of Operational Research*, 202(1):1 – 7.

Lawler, E. L. (1977). A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. In Hammer, P., Johnson, E., Korte, B., and Nemhauser, G., editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 331 – 342. Elsevier.

Milan, A., Rezatofighi, S. H., Garg, R., Dick, A. R., and Reid, I. D. (2017). Data-driven approximations to NP-hard problems. In *AAAI*, pages 1453–1459.

Panwalkar, S., Smith, M., and Koulamas, C. (1993). A heuristic for the single machine tardiness problem. *European Journal of Operational Research*, 70(3):304–310.

Potts, C. and Van Wassenhove, L. N. (1991). Single machine tardiness sequencing heuristics. *IIE transactions*, 23(4):346–354.

Potts, C. and Wassenhove, L. V. (1982). A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters*, 1(5):177 – 181.

Russell, R. and Holsenback, J. (1997). Evaluation of greedy, myopic and less-greedy heuristics for the single machine, total tardiness problem. *Journal of the Operational Research Society*, 48(6):640–646.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484.

Süer, G. A., Yang, X., Alhawari, O. I., Santos, J., and Vazquez, R. (2012). A genetic algorithm approach for minimizing total tardiness in single machine scheduling. *International Journal of Industrial Engineering and Management (IJIEM)*, 3(3):163–171.

Sundermeyer, M., Schlüter, R., and Ney, H. (2012). Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*.

Szwarc, W., Della Croce, F., and Grosso, A. (1999). Solution of the single machine total tardiness problem. *Journal of Scheduling*, 2(2):55–71.

Szwarc, W. and Mukhopadhyay, S. K. (1996). Decomposition of the single machine total tardiness problem. *Operations Research Letters*, 19(5):243–250.

Václavík, R., Novak, A., Šůcha, P., and Hanzálek, Z. (2018). Accelerating the branch-and-price algorithm using machine learning. *European Journal of Operational Research*, 271(3):1055 – 1069.

Václavík, R., Šůcha, P., and Hanzálek, Z. (2016). Roster evaluation based on classifiers for the nurse rostering problem. *Journal of Heuristics*, 22(5):667–697.

Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.

Zhou, D. N., Cherkassky, V., Baldwin, T. R., and Olson, D. E. (1991). A neural network approach to job-shop scheduling. *IEEE Transactions on Neural Networks*, 2(1):175–179.