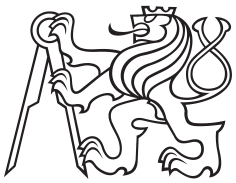
Master's thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Data-driven algorithm for single machine scheduling problem

Michal Bouška

Supervisor: Ing. Antonín Novák
May 2019

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bouška** Jméno: **Michal** Osobní číslo: **420084**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Daty řízený algoritmus pro rozvrhování na jednom stroji

Název diplomové práce anglicky:

Data-driven algorithm for single machine scheduling problem

Pokyny pro vypracování:

This thesis addresses data driven approaches to scheduling problems.Specifically, it investigates a single machine scheduling problem where the objective is to minimize total tardiness.The aim is to propose an algorithm that will exploit solutions of previously solved instances to find a solution to a given instance.The particular objectives of the thesis are:
1) Review the existing works addressing single machine scheduling problems with total tardiness. Furthermore, review existing data driven approaches addressing combinatorial problems.
2) For problem 1||sum Tj, devise and implement a data driven scheduling algorithm. Try to use fundamental properties of the problem.
3) Devise and implement a generator of couples (instance, solution) and use it the data-driven algorithm. Put emphasis on the efficiency of the generator in terms of its speed.
4) Compare the algorithms with results published in the literature. For benchmarking use standard benchmark instances.

Seznam doporučené literatury:

[1] Michele Garraffa, Lei Shang, Federico Della Croce, Vincent T'Kindt. An exact exponential branch-and-merge algorithm for the single machine total tardiness problem. Theoretical Computer Science, Elsevier, 2018, 745 (1), pp.133 - 149.
[2] Szwarc, W. , Grosso, A. and Croce, F. D., Algorithmic paradoxes of the single-machine total tardiness problem. J. Sched. 2001, 4: 93-104.
[3] Michael L. Pinedo. Scheduling: Theory, Algorithms, and Systems (3rd ed.). Springer Publishing Company, Incorporated. 2008.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Antonín Novák, katedra řídicí techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **28.01.2019** Termín odevzdání diplomové práce: **24.05.2019**

Platnost zadání diplomové práce: **20.09.2020**

Ing. Antonín Novák podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Acknowledgements

I would like to thank the CTU in Prague for being a very good *alma mater*.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, May 23, 2019

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 23. května 2019

Abstract

In this thesis, we present a method for solving a single machine total tardiness problem a classical $\mathcal{NP}$-hard scheduling problem. We investigate the use of deep learning method for this problem. We utilize known decomposition method from operation research, and we derive data-driven method. We introduce a deep neural network that predicts the objective value and acts as a polynomial-time oracle. Oracle drive decomposition method in each step. Our data-driven method outperforms the state-of-the-art heuristic in terms of optimality gap.

Keywords: single machine total tardiness, data-driven method, deep neural network

Supervisor: Ing. Antonín Novák

Abstrakt

V této práci představuje metodu pro řešení problému Single machine total tardiness problem. Analyzovali jsme užití Deep learning metod, pro Single machine total tardiness problem. Využili jsme rozděl a panuj algoritmu a odvodili z něj data-driven přístup. Vytvořili jsme hlubokou neuronovou síť se schopností predikovat hodnotu kritéria problému. Tato neuronová síť jedná jako optimální orákulum s polynomiální dobou běhu. Orákulum řídí rozděl a panuj metodu v každém kroku. Náš data-driven přístup překonává state-of-the-art heuristiku ve kvalitě řešení.

Klíčová slova: rozvrhování na jednom stroji, daty řízená metoda, hluboké neuronové sítě

Překlad názvu: Daty řízený algoritmus pro rozvrhování na jednom stroji

Contents

Figures

Tables

Chapter 1

Introduction

For the last 40 years, Machine Learning (ML) has been the go-to tool for various engineering and scientific problems for which designing the handcrafted algorithms is a complicated process. The general idea of ML is to learn from data and then use the learned knowledge for predictions. Although several ML paradigms and models have been proposed over the years, the current attention is mainly on deep neural networks (DNNs) [Sch15]. Inspired by the success of DNNs, AI researchers have become attracted in the recent years by applications of DNNs to the problems of Combinatorial Optimization and Operations Research (OR), such as Traveling Salesman Problem (TSP).

Traditionally, the OR community is focused mostly on developing algorithms exploiting properties and the structure of these optimization problems. Over the course of the past 70 years, many sophisticated algorithms were developed for these classical problems. Typically, the design of an efficient classical algorithm utilizes two components. The first one is rigorous knowledge of problem structure, such as decomposition rules, dominance theorems, and bounds that help to prune useless part of search space and mitigate symmetries. The second component is a heuristic, often incorporating ad-hoc strategies and rules of thumb acquired by the designer of the algorithm based on the experience and behavior of the algorithm with solving realistic instances. While the rigorous component provides essentially free speed-up of the algorithm, the design of guiding rules is a tedious job for the designer, often limited by the experience and resources. Moreover, heuristics are tailored to a specific class of problem instances, and, due to *no free lunch theorem*, they are likely to perform poorly on other classes of instances.

On the other hand, a significant part of the ML community tackles these problems more or less blindly. They often design end-to-end machine learning models that try to solve the problem as a whole. These models do not utilize state-of-the-art knowledge from OR, and, hence, the models are handicapped due to ignoring the known rigorous knowledge, or they are unable to figure out these theorems from the data by themselves. In both cases, often the result is that these methods frequently fall short behind state-of-the-art algorithms from OR in terms of speed and quality of solutions. However, it is clear that ML has its potential in the second component of successful algorithms – the heuristic component. Hence, the successful application of ML to solving

combinatorial problems lies in efficient integration with existing classical state-of-the-art methods.

1.1 Motivation

The classical approach for solving combinatorial problems have several undecisable properties. First, solving an instance of $\mathcal{NP}$-Hard problem to optimality consumes an unfruitful amount of computer time. Second, there is no well-established way how to utilize the found solution for improving the algorithm or recycling solution for the next instances. Heuristic rules from expert knowledge are often developed for worst case, only from a few instances and it takes a massive amount of expert time. We are expecting that the neural network has a prerequisite for estimating criterion of well known scheduling problem and we are evaluating ability to estimate criterion by neural network on $1||\sum T_j$. Additionally, there exists decomposition rule for the $1||\sum T_j$ problem, which enables to utilize of criterion estimation of neural network. This property avoids the end-to-end approach, and instead of this, it effectively builds solution only from criterion estimation.

1.2 Related work

This section will be split into two subsections. The first subsection introduces classical methods of solving the total tardiness scheduling problem. The second section subsection summarizes the current research of applying data driven approaches to combinatorial problems.

1.2.1 Single Machine Total Tardiness Problem

There is a large volume of published studies investigating the exact algorithm for the single machine total tardiness problem. In 1977 it was shown [Law77] that weighted single machine total tardiness problem is $\mathcal{NP}$-Hard problem. However it took more than 20 years to prove, the unweighted variant of the problem is $\mathcal{NP}$-Hard problem too [DL90]. [Kou10] introduce exhaustive survey for $1||\sum T_j$, we are bellow presenting most important part of this survey. And extend information with new state-of-the-art algorithm.

Lawler *et al.* [Law77] proposes a pseudo-polynomial (in the sum of processing times) algorithm for solving single machine total tardiness problem. This algorithm is based on a decomposition of the problem into subproblems, where the decomposition selects the job with maximum processing time and tries all eligible position. Lawler decomposition split problem to two subproblems, first subproblem contains all jobs preceding job with maximum processing time and second subproblem contains all jobs following job with maximum processing time, and this split generate for all position of job with maximum processing time, where this job can be place to position following his original position in EDD. Also Lawler introduces rules for filtering possible

position of job with maximum processing times. This algorithm can solve instances up to one hundred jobs. Mukhopadhyay *et al.* [SM96] introduces a technique for solving the special cases of instances where it is possible to find an optimal solution by Lawler rules. F. Della Croce *et al.* [DCTBG98] proposed a new approach, which is based on Lawler decomposition, but in one step applies decomposition over maximum processing time job. Moreover, the decomposition was generalized to work over due dates. Szwarc *et al.* [SDCG99] combines double decomposition from [DCTBG98] and split from [SM96], and also improves runtime of split procedure to quadratic time. This algorithm was for a long time state-of-the-art with the ability to solve instance up to 500 jobs. Recently, Tkindt *et al.* [GSDCT18] proposes branch and merge algorithm with inferring information about nodes of the search tree and merge nodes related to the same subproblem. This algorithm is able to solve instances up to 1300 jobs.

Exact algorithms have very large computation times for large instances and the optimal solution is rarely needed in practice. Hence, heuristic algorithms are often more practical. Existing heuristics algorithm can be derived into three major groups.

The first group of heuristics creates a job order and schedule jobs according this order, i.e., list schedule algorithms. There are various methods for creating a job order. The easiest one, although not very efficient is sort job by Earliest Due Date rule (EDD). A more efficient algorithm called NBR was proposed in [HR92]. This heuristic is initialized with EDD order and repeatedly try to swap the last job with previous job to minimize tardiness. NBR heuristic overcome older heuristic approaches both in quality and runtime. Panwalkar *et al.* [PSK93] propose PSK heuristic which starts with jobs sorted by shortest processing time and iteratively swaps unscheduled job with shortest processing time to later positions. Rusell *et al.* [RH97] compares PSK and NBR heuristic, and conducted that neither heuristic is inferior to the another one. However, NBR returns a better result in more cases. The second group of heuristics is based on Lawler decomposition rule [Law77]. In this case, heuristic is evaluated each child of tree search node and most promising child is expanded. This heuristic approach is evaluated in [PVW91] with EDD heuristic. The third group of heuristic is metaheuristics. [KGV83], [PVW91], [AK96], [BDAF96] present using of simulated annealing for $1||\sum T_j$, [DZ99], [SYA+12] present using of genetic algorithm for $1||\sum T_j$ and [DDC99], [BBHS99], [CLG09] present using of ant colony optimization for $1||\sum T_j$.

1.2.2 Existing data driven approach

Application of ML in OR has its challenges, namely handling inputs of variable size and obtaining training data. Obtaining training data can be difficult procedure, usually obtaining one training instance request to solve problem with same complexity like final problem. The first issue can be addressed by recurrent networks and, more recently, by encoder-decoder type

of architectures. [VFJ15] applied an architecture called Pointer Networks that, given a set of graph nodes, outputs a solution as a permutation of these nodes. The authors applied the pointer networks to TSP, however, this approach for TSP is still not competitive with the best classical solvers such as Concorde [ABCC06], that can find optimal solutions to instances with 50 nodes in a fraction of second. Moreover, the output from the Pointer Network needs to be corrected by the beam search procedure, which indicates the weaknesses of Pointer Networks in approximating the problem well on its own. Pointer network has gap around 1% for instance with 20 nodes from optimum.

The problem with obtaining the training set can be addressed with reinforcement learning paradigm. [DCL+18] 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 solver. Similar approach is described in [KW18], which, if treated as greedy heuristic, beats weak baseline solutions (from OR 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 compare directly 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.

[KDZ+17] 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.

[MRG+17] presents a data-driven approximation of solvers for $\mathcal{NP}$-hard problems. They utilized an 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 [ZCBO91] and [JM98]. 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 [LBQ12] address a nurse rostering problem and describe a way to evaluate the quality of the solutions without calculating their exact criterion values. They propose a classifier, implemented as a simple 1-layer or 2-layer 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. Their approach was later on improved by a faster and more accurate neural network classifier proposed in [VŠH16]. Nevertheless, both approaches are 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 [VNŠH18]. In this case, 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 for 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.

1.3 Contribution and thesis outline

Combinatorial optimization is massively studied part of science for a long time and attention for the neural network in the last decade is heavy. Combination of these two approaches is relatively new and not studied into depth. In the literature, few works are applying the only end-to-end technique for solving combinatorial problem [VFJ15], [DCL$^+$18]. The end-to-end approach is in this time ineffective and not comparable to long developed methods from the literature. An alternative way to end-to-end approach is to combine knowledge from combinatorial problem with the neural network. In this case, work on solving the problem can be distributed between the classical algorithm and neural network. We have introduced a heuristic method for $1||\sum T_j$, based on a combination of decomposition approach from [Law77] and neural network as a regressor. The method can scale to the state of the art size of instances, and it is comparable with the result of state-of-the-art heuristic.

Chapter 2

Problem statement

The problem addressed in this thesis is denoted in the literature as $1||\sum T_j$, i.e., single machine total tardiness problem. Let $J = \{1, \ldots, n\}$ be a set of jobs that have to be processed on a single machine, each job $j \in J$ has processing time $p_j \in \mathbb{N}$ and due date $d_j \in \mathbb{N}_0$. All jobs are available at time zero, machine can process only one job at the time and the preemption of the job is not allowed. Let $\pi : \{1, \ldots, n\} \mapsto \{1, \ldots, n\}$ be a bijective function representing a sequence of the jobs, i.e., $\pi(k)$ is the job on position k in sequence π. For a given sequence π, completion time of job $\pi(k)$ is defined as $C_{\pi(k)} = \sum_{k'=1}^{k} p_{\pi(k')}$. Finally, tardiness of job $\pi(k)$ is defined as $T_{\pi(k)} = \max(0, C_{\pi(k)} - d_{\pi(k)})$. The problem is to find a sequence which minimizes the total tardiness, i.e., $\sum_{j \in J} T_j$. It is good to stress that every order of jobs is sustainable. For easier notation we assume jobs having ordered according to EDD (Earliest Due-date First).

2.1 ILP formulation

Problem can be formalized by Integer Linear Programing (ILP), ILP is mathematical framework for formalizing and solving problem by set of integer linear equation. Problem is formalized in a relative order ILP model.

$$\text{minimize} \quad \sum_{i=1}^{n} t_i \tag{2.1}$$

$$\text{s.t.} \quad t_i \geq 0 \qquad \forall i \in J \tag{2.2}$$

$$t_i \geq s_i + p_i - d_i \qquad \forall i \in J \tag{2.3}$$

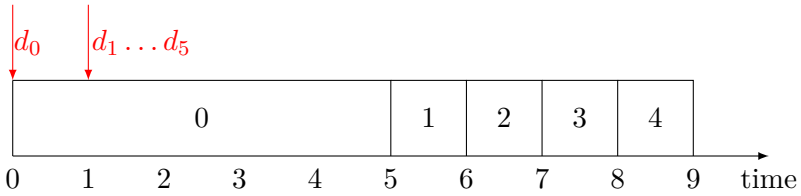
$$p_i + s_i - s_j \leq M \cdot x_{i,j} \qquad \forall i \in J, \forall j \in \{1, \ldots, i-1\} \tag{2.4}$$

$$s_j + p_j - s_i \leq M \cdot (1 - x_{i,j}) \qquad \forall i \in J, \forall j \in \{1, \ldots, i-1\} \tag{2.5}$$

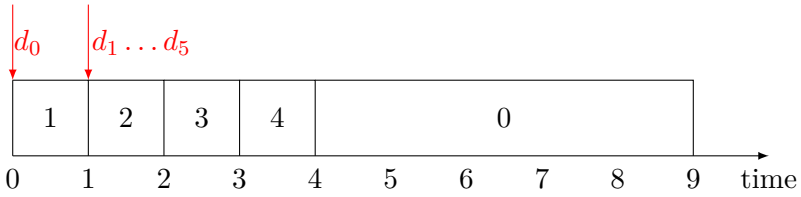
$$\text{where} \quad x_{i,j} \in \{0, 1\} \tag{2.6}$$

$$t_i \in \mathbb{N}_0 \tag{2.7}$$

Variable s_i represents start time of job i, variable t_i represents tardiness of job i, variable $x_{i,j}$ defines order of job i and j and constant M is equal to sum of all processing times. Equations (2.2) and (2.3) transform start time of a job to tardiness of the job. Equations (2.4) enforce that if job i precedes



(a) : Visualization of the EDD schedule with total tardiness equal to 31.



(b) : Visualization of the optimal schedule with total tardiness equal to 18.

Figure (2.1): Visualization of $1||\sum T_j$ instance.

to job j, then the completion time of job i is at most the start time of job j. Equations (2.5) solve the opposite case to (2.4). But it is important to note, that ILP is able to solve only small instance.

2.2 Example

An example bellow illustrates an instance of problem $1||\sum T_j$ with 5 jobs. Their processing times and due dates are described in Table 2.1. Figure 2.1a shows a solution obtained by the EDD rule while Figure 2.1b shows the optimal solution. Although the problem has a straightforward structure, contains only one long job and four short jobs with the same due dates, EDD provides almost two times worst criterion value. It is because this instance is an example of a scenario when EDD may return an arbitrary bad result [DCGP04].

i	1	2	3	4	5
p_i	5	1	1	1	1
d_i	0	1	1	1	1

Table (2.1): Table of processing times and due dates for the example instances

Chapter 3

Scheduling algorithms for $1||\sum T_j$

In this chapter, the individual solvers for $1||\sum T_j$ will be described. In section 3.1 we describe an algorithm based on decomposition over the job with the longest processing time. In the next chapter, we are utilizing this algorithm in our data-driven heuristic approach [Law77]. In section 3.2 we describe algorithm based on double decomposition over a job with the longest processing time and also a job with the minimal due date, state-of-the-art algorithm is based on decomposition from this algorithm [SDCG99]. In section 3.3 we describe the state-of-the-art algorithm based on memorization of states, we are using this algorithm for comparing with our heuristic approach [GSDCT18]. In section 3.4 we describe state-of-the-art heuristic.

For the purpose of this chapter, we need to define a few terms.

Definition 3.1. Jobs J are sorted according to the EDD, if $d_1 \leq d_2 \leq \cdots \leq d_n$. In case of ties ($d_i = d_{i+1}$), the jobs are sorted according the processing times, i.e., $p_i \leq p_{i+1}$.

Definition 3.2. Jobs J are sorted according to the SPT, if $p_1 \leq p_2 \leq \cdots \leq p_n$. In case of ties ($p_i = p_{i+1}$), the jobs are sorted according the processing times, i.e., $d_i \leq d_{i+1}$.

Let us denote the k-th job in jobs set according EDD or SPT rule as EDD_k or SPT_k, respectively. And SPT_{EDD_k} is position in SPT of job k from EDD order. From example in Table 2.1 for example EDD_1 is job with processing time 5 and due date 0, SPT_1 is job with processing time 1 and due date 1 and SPT_{EDD_1} is 5.

3.1 Lawler decomposition

The main idea proposed by [Law77] is look on $1||\sum T_j$ problem over a job with the longest processing time. [Law77] proves, that for jobs set ordered in EDD and job j^* with the longest processing time at least one optimal solution exists with j^* between position j^* and n. This property leads to the decomposition of the problem, where for each eligible position k for job j^*, the problem is decomposed into two subproblems — one with $k-1$ preceding jobs, and second with $n-k$ following jobs. Both subproblems are solved recursively using the same decomposition algorithm.

More formally let J be indexed by EDD. Let $j^* = \arg\max_{j \in J} p_j$ be most right job with longest processing time. [Law77] prove that there exists position $k \in \{j^*, \ldots, n\}$ such that at least one optimal solution exist where j^* is preceded by all jobs $\{1, \ldots, k\} \setminus \{j^*\}$ and followed by all jobs $\{k+1, \ldots, n\}$. This property leads to the following exact decomposition algorithm. In each node of the search tree, find the longest job j^*. For each eligible position $k \in \{j^*, \ldots, n\}$, the problem is decomposed into two subproblems P_k and F_k with jobs $\{1, \ldots, k\} \setminus \{j^*\}$ and $\{k+1, \ldots, n\}$, respectively. Both subproblems are solved recursively using the same decomposition algorithm; let $Z(P_k)$ and $Z(F_k)$ denote the optimal criterion value for subproblem P_k and F_k, respectively.

The speed of searching optimal solution is rapidly affected by the branching factor in decomposition. Here, the branching factor is equal to the number of eligible positions where it is possible to place a job with the longest processing time. [PW82] propose rules (3.1) and (3.2) that reduce the number of position where it is possible to place that job and the resulting algorithm is still optimal. If any of the following rules hold for k, it is not necessary to decompose this k

$$k \in \{j^*, \ldots, n-1\} \text{ and } \sum_{i=1}^{k} p_i \geq d_{k+1} \qquad \text{[PW82]} \qquad (3.1)$$

$$k \in \{j^*+1, \ldots, n\} \text{ and } \sum_{i=1}^{k-1} p_i < d_k. \qquad \text{[PW82]} \qquad (3.2)$$

[SDCG99] add additional three rules for filtering eligible positions of the job with the longest processing time. [SDCG99] also consider same rules with modified due dates, where due dates are replace with *maximally increased due dates* defined in [Law77]. Define $TT(n, l)$ as the tardiness of a sequence ordered in EDD, but with job n moved to position l, then we have:

$$\exists l \in \{j^*+1, \ldots, i-1\} \text{ and } \sum_{i=1}^{k} p_i < d_l + p_l \qquad \text{[SDCG99]} \quad (3.3)$$

$$k \in \{j^*, \ldots, n\} \text{ and } T(n,k) > T(n,k+1) \qquad \text{[SDCG99]} \quad (3.4)$$

$$k \in \{j^*, \ldots, n\}, \exists l \in \{j^* \ldots k\} \text{ and } T(n,k) \geq T(n,l) \qquad \text{[SDCG99]} \quad (3.5)$$

3.2 SDD by Swartz *et al.*

[SDCG99] propose an algorithm for $1||\sum T_j$ that has been state-of-the-art till year 2018. This algorithm is based on two ideas, the first one is called *double decomposition* introduced in [DCTBG98], and the second one called *split* originally introduced in [SM96] and improved by [SDCG99] to run in time $O(n^2)$.

For us, the most important is the idea of double decomposition, since we will utilize it later. [DCTBG98] define *double decomposition* as follows. Select the first job in EDD order, and denote it as $\alpha = SPT_{EDD_0}$. Let J^α be a subset of J, defined as $J^\alpha = \{J_{SPT_1}, \ldots, J_{SPT\alpha}\}$. Now job α can be at

position $k = \{1, \ldots, J^{\alpha}_{SPT_{\alpha}} = |J^{\alpha}|\}$ and preceding subproblem P is defined as $P_{\alpha} = \{J^{\alpha}_{EDD_1}, \ldots, J^{\alpha}_{EDD_{k-1}}\}$, and following subproblem is defined like $F_{\alpha} = \{J^{\alpha}_{EDD_{k+1}}, \ldots, |J^{\alpha}|\}$. [DCTBG98] show that from rules introduced in [Emm69] is implied that α have to precede j^*, or have to be the same job in optimal sequence. Due to this, we can decompose $J \setminus \{\alpha, j^*\}$ into three subset P_{α}, $F_{\alpha} \cap P_{j^*}$ and F_{j^*}. This subset we solve recursively with same decomposition rules.

3.3 TTBR by T'Kint *et al.*

[GSDCT18] utilize *double decomposition* and rules to prune possible position of job j^* and α and propose algorithm based on merging of search space states of *double decomposition*.

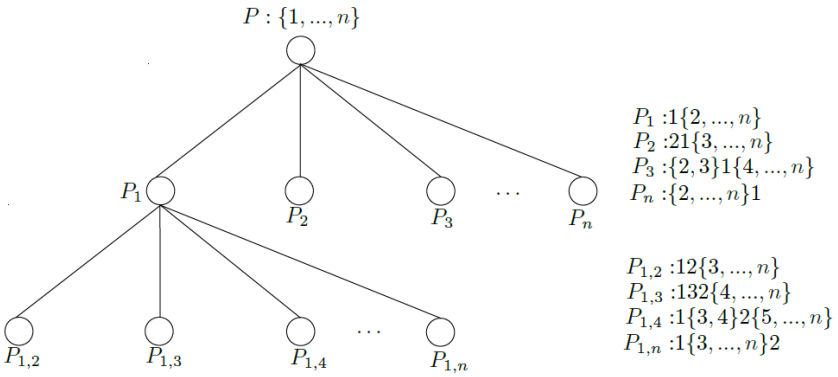


Figure (3.1): The branching scheme [GSDCT18]

To illustrate the idea behind the merging technique consider Figure 3.1. In this example, nodes P_2 and $P_{1,2}$ are identical except for the initial subsequence (21 vs 12). This fact implies, in this particular case, that the problem of scheduling jobset $\{3 \ldots n\}$ at time $p_1 + p_2$ is solved twice. This kind of redundancy can however be eliminated by merging node P_2 with node $P_{1,2}$ and creating a single node in which the best sequence among 21 and 12 is scheduled at the beginning and the jobset $\{3 \ldots n\}$ [GSDCT18].

3.4 NBR

NBR algorithm proposed in [HR92] is based on reallocation last job (in EDD) to position with the biggest reward. Reward is called NBR and it is derived from Emmons rules [Emm69]. In each iteration, algorithm takes the latest job by EDD, calculate NBR for preceding positions and swap latest job with job on position with the biggest NBR-criterion. At the end of iteration, algorithm removes last job from job set. Algorithm is described by Python like pseudo

code in algorithm 1. As an input, the algorithm gets an array of due dates due and processing times proc and return NBR order.

Algorithm 1: NBR

```
   Data: due, proc
   Result: NBR order
 1 n ← len(due)
 2 order ← list(range(n))
 3 for 0 ≤ i < n do
 4 |   due[i] ← max(due[i], proc[i])
 5 end
 6 due, proc, order = sort_arrays(due, proc, order)
 7 for k from n to 0 do
 8 |   completeness ← cumulative_sum(proc)
 9 |   tardiness ← [max(0, comp - tard) for comp, tard in
   |    zip(completeness, due)]
10 |   p_k ← proc[k]
11 |   if tardiness[k] > p_k then
12 |   |   for j from k to 0 do
13 |   |   |   p_j ← proc[j]
14 |   |   |   d_j ← due[j]
15 |   |   |   c_j ← completeness[j]
16 |   |   |   if p_j > p_k then
17 |   |   |   |   nbr_j ← max(0, d_j - c_j)
18 |   |   |   |   for l from j+1 to k+1 do
19 |   |   |   |   |   nbr_j ← nbr_j + min(p_j, tardiness[l]) - proc[l]
20 |   |   |   |   end
21 |   |   |   |   if nbr_j > nbr_max then
22 |   |   |   |   |   nbr_max ← nbr_j
23 |   |   |   |   |   best_pos ← j
24 |   |   |   |   end
25 |   |   |   |   else if nbr_j == nbr_max and proc[best_post] < p_j
   |   |   |   |    then
26 |   |   |   |   |   best_post ← j
27 |   |   |   |   end
28 |   |   |   end
29 |   |   |   if nbr_max > 0 then
30 |   |   |   |   due.insert(k, due.pop(best_post))
31 |   |   |   |   proc.insert(k, proc.pop(best_post))
32 |   |   |   |   order.insert(k, order.pop(best_post))
33 |   |   |   end
34 |   |   end
35 |   end
36 end
37 return order
```

Chapter 4

Data-driven approach

Let us recall the main disadvantages of the classical approach for $\mathcal{NP}$-hard problems. Solving of $\mathcal{NP}$-hard problem instance to optimality consume an unfruitful amount of time. A heuristic approach is derived to work in the worst case, in many application is more useful to focus on the most common case. Classical algorithm is not able to reuse solution of older instances and improve results or runtime. We introduce a data-driven approach for the $1||\sum T_j$ problem, which utilizes the known solution of instances to solve instances efficiently in time.

In this chapter, we introduce an approach to use Lawler decomposition as a heuristic utilizing data-driven approach. In section 4.1, we compare different algorithm sand possibility to integrate these algorithms with data-driven approach. In section 4.2, we present changes in Lawler decomposition to be able work with regressor presented in section 4.3.

4.1 Analysis of state-of-the-art-algorithms concerning use of a data-driven approach

In this section, we will describe ways of integrating data-driven approach into existing classical algorithms and shortly discuss advantages and disadvantages of integration of algorithms. On the other hand it is possible to develop heuristic for $1||\sum T_j$ by restriction of evaluated position in decomposition.

As first let us recall the common property of Lawler, SDD and TTBR. These algorithms are exact algorithms based on the decomposition rules. This common property implies that the integration with data driven approach is possible in a similar way with all of these algorithms. The integration of a data driven-approach and Lawler algorithm is possible with an oracle estimating criterion value of jobs set J. With this oracle, we can estimate criterion of each eligible decomposition and the select best one. We introduce in details this approach in section 4.2. Criterion oracle is also applicable for SDD an TTBR. The major advantage of integration Lawler with data driven approach is simplicity of Lawler decomposition, but still, Lawler decomposition provides powerful framework to utilize data-driven approach. SDD introduces double decomposition for which it is possible to use the same

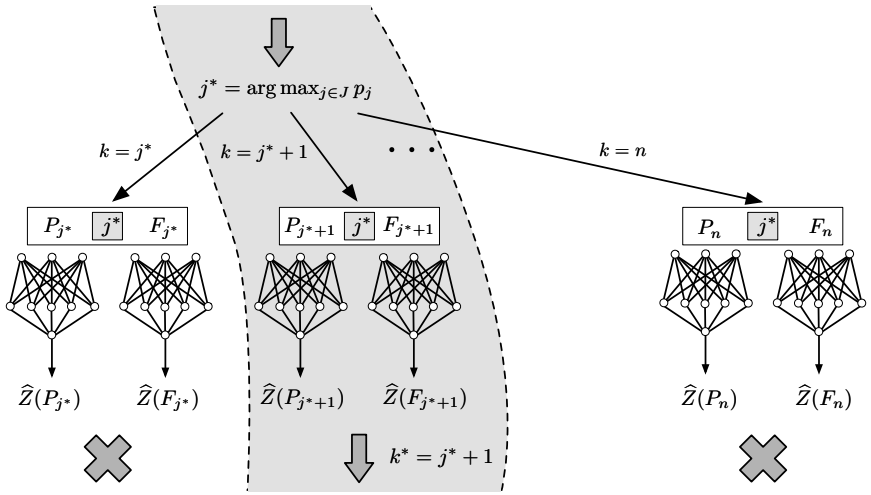


Figure (4.1): An evaluation of one node of the search tree in the algorithm integrating the decomposition with the DNN. Position k^* is the position that minimizes the approximated criterion value for job j^*.

Oracle for each decomposition. However, double decomposition complicate implementation, and therefore Lawler is a better candidate for data driven approach. TTBR utilizes memorization in double decomposition. In combination with oracle as heuristic, there is no possibility of memorization and due to that TTBR is likely not good candidate for data driven-approach. NBR have different position from algorithms above. NBR is heuristic and do not utilize decomposition approach. To combine NBR with data driven approach we have to implement oracle returning NBR-criterion. But it is important to denote that this oracle have to be better than function from NBR in sense of final schedule criterion. Due to this, this method is not appropriate for data driven approach.

4.2 Lawler heuristic search

In this section, we present an efficient greedy heuristic combining Lawlers decomposition, introduced in previous chapter, with a DNN acting as an aproximation of the criterion oracle. Let us call it Lawler heuristic search (LHS).

We begin by observing that the exact algorithm described in section 3.1 can be easily turned into a heuristic one by not branching on every eligible position. For example, if we approximate the values $Z(P_k), Z(F_k)$ by $\widehat{Z}(P_k), \widehat{Z}(F_k)$, respectively, then the heuristic may expand only a few, selected branches that have the smallest approximated criterion value

$$\widehat{Z}(P_k) + \max\left(0, \sum_{j \in \{1,\dots,k\} \setminus \{j^*\}} p_j - d_{j^*}\right) + \widehat{Z}(F_k) \tag{4.1}$$

thus decreasing the overall runtime of the algorithm. Moreover, if only one

branch having the smallest approximated criterion value is expanded, we say that such a heuristic is greedy. Notice, that if the approximation is exact, i.e., $Z(P_k) = \widehat{Z}(P_k)$ and $Z(F_k) = \widehat{Z}(F_k)$, the greedy heuristic always finds an optimal solution thus it would be an exact algorithm. However, computing these exact quantities is an $\mathcal{NP}$-hard problem. Therefore, we use a deep neural network as a polynomial-time approximation of the oracle, that "knows" the exact value.

The idea of the proposed integration of the DNN into the decomposition algorithm is simple: in every node of the search tree, use the DNN to compute the approximated values $\widehat{Z}(P_k), \widehat{Z}(F_k)$ for every eligible position, i.e., use the DNN to predict the values $Z(P_k), Z(F_k)$. The heuristic then greedily selects eligible position k^* which minimizes the approximated criterion value, fixes the position of the longest job j^* to k^* and recursively solve two new subproblems P_k and F_k; see a visualization in Figure 4.1. In algorithm 2 is proposed python like pseudo code for this heuristic, as input is expected array of processing times and due dates and regressor function which return expectation for criterion value for given subproblem defined by arrays of processing times and due dates. The total $\mathcal{O}(n^2)$ predictions are made during the run of the algorithm.

Similar approach is proposed in [OSL17] they are defeat DQN architecture in Atari Games with neural network used as oracle, where this neural network predict future rewards. Major differences from our approach are that theirs instance represent states of game instead of subproblems and they are train Q-Value as criterion.

4.2.1 State space search

In our algorithm we want to rate every possible decomposition, with our oracle Z, (every positions of job j^*) of J but we want to visit just small subset of this decompositions, because in other case we are on complexity of optimal algorithm. Let us denote the search tree of the $1||\sum T_j$ as a tree where every node is subproblem of J defined by jobs and a start time. In Lawler decomposition for one node of the search tree and for each eligible position of j^*, there are two nodes in tree. One consist from preceding jobs P and starting time is equal to parent node. The second consists from following jobs F and start time is equal to starting time of the parent node plus the sum of processing times of jobs in the first child. Let us call this two child as *child pair*. We assume, that each child of node is rated by oracle Z.

There are several methods how to select a child to be expanded by Lawler decomposition. The first is a greedy method, that selects only the best rated pair of child to expand. A small change of the greedy method leads to k-greedy method. This method selects in every node k best child pairs to expand. Unfortunately, this method is still computationally expensive because the count of expanded nodes grow exponentially with the size of the instance. Usually it is possible to address this behaviour by limiting the maximal number of nodes in the queue. However in case of our usage of

Algorithm 2: Lawler heuristic

```
   Data: proc, due, regressor
   Result: jobs order
 1 Function Main(proc, due):
 2     n ← len(due)
 3     j* ← right_argmax(proc)
 4     minimal_criterion ← inf
 5     for k from j* to n do
 6         left_processing_times ← proc[0:j*] + proc[j*+1:k]
 7         left_due_dates ← due[0:j*] + due[j*+1:k]
 8         right_processing_times ← proc[k+1:n]
 9         right_due_dates ← due[k+1:n]
10         c ← max(0, sum(left_processing_times) - due[j*])
11         c ← c + regressor(left_processing_times, left_due_dates)
12         c ← c + regressor(right_processing_times, right_due_dates)
13         if minimal_criterion < c then
14             left_min_proc ← left_processing_times
15             left_min_due ← left_due_dates
16             right_min_proc ← right_processing_times
17             right_min_due ← right_due_dates
18             min_pos ← k
19             minimal_criterion ← c
20         end
21     end
22     left_order = main(left_min_proc, left_min_due)
23     right_order = main(left_min_proc, left_min_due)
24     return join_order(min_pos, left_order, right_order)
25 return main(proc, due)
```

Lawler decomposition it is limited by child pair. Because it is necessary to hold both child of child pair in queue, or remove both childs from child pair from queue and check this property recursively to top of tree search. The other option is Limited discrepancy search (*LDS*). This method introduced in [HG95] is based on limiting the number from greedy approach. LDS is characterized by the count of allowed discrepancies d. LDS with $d = 0$ yields the same solution as the greedy method. LDS with $d = 1$ yields a solution from greedy method and in each node of greedy solution expand one more child pair. In general sense, LDS with d enables in each node to expand one more child pair than LDS with $d - 1$, where next child to expand is selected by heuristic. New expanded child pair continue in expanding only for best child pair by rank. LDS in [HG95] is defined for binary search tree (search tree where every node have maximally two child), let us define that position i in rating is equal to discrepancy $i - 1$. See Figure 4.2 for LDS with $d \in \{0, 1, 2, 3\}$ and with search tree with only two child for node.

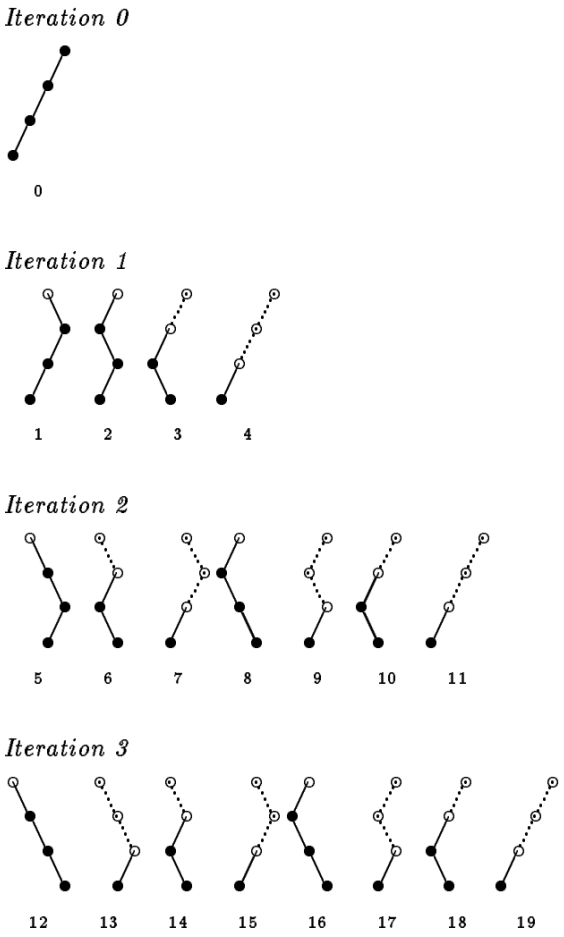


Figure (4.2): Execution trace of LDS [HG95]

4.3 Regressor

In this section, we propose regressor architecture that acts as a criterion oracle. Our motivation to propose regressor is to predict the optimal criterion value for a set of jobs, i.e., P_k and F_k.

Regressor consists from a basic neural network model, this model is proposed in the following subsection. The result of a neural network model can be strongly affected by the prepossessing of input data. Types of input normalization for neural network are discussed in the following paragraphs. Types of canonical form for neural network are discussed in the following paragraphs. Types of criterion normalization for neural network are discussed in the following paragraphs.

Network architecture

The network in its own setting solves a regression problem — predicting values from a continuous interval. Since the sizes (i.e., the number of jobs) of

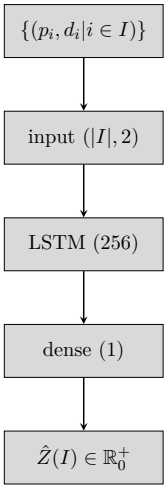


Figure (4.3): Visualization of the regressor with criterion normalization by constant.

the subproblems change during the execution of the decomposition algorithm, we propose a recurrent neural network to deal with varying input size; see its architecture in Figure 4.3 The input is given by a set of jobs I such that I is either P_k or F_k. Each job $j \in I$ is iteratively passed to the Long Short-Term Memory (LSTM) layer [HS97]. Job j is characterized by two features — processing time p_j and due date d_j. The LSTM has a hidden layer of size equal to 256 and is followed by a dense layer with one bias unit. After the last job of I is passed, the network outputs a single scalar value.

Normalization of network inputs

The result of the neural network is likely to be affected by normalization of input data. For neural networks it is often good choice to have the input feature values in interval from zero to one. We are introduce two methods for input normalization. Let us recall, that the input consists from jobs J, $n = |J|$ Each job $j \in J$ has a processing time $p_j \in \mathbb{N}$ and a due date $d_j \in \mathbb{N}_0$. It is unfruitful to normalize processing times and due dates with different constant, since there is direct connection between processing times and due dates over all J.

Let us define method a *1PSMD* (processing time sum and due dates maximum). This method divides all processing times and all due dates in J by a value $\max\{\sum_{i \in J} p_i, \max_{i \in J} d_i\}$. Method 1PSMD normalizes processing times and due dates to interval from zero to one. Next, let us define a method *NPS0D* (n times processing time sum). This method divide all processing times and all due dates in J by $n \cdot \sum_{i \in J} p_i$). Method NPS0D normalizes processing times and due dates in benchmark instances (introduced in section 4.4) to the interval from zero to one.

Canonical forms of input data

Neural networks are often sensitive to the form of input data. Even it is possible to input raw data to neural network, it usually benefits from a prepossessing step. We propose two prepossessing steps, the first part of prepossessing data for neural network is sorting of jobs set J. The second prepossessing step is to add some additional information for neural network, like add information about position of job in instance.

There are several methods for presorting of job set J. Two of the possible method are denoted above EDD and SPT. Furthermore it is possible to reverse order of these methods, i.e., *LDD*, and *LPT* respectively. Let us remark, that neural network iteratively reads the input vector of processing times and due dates over all jobs set J. We can add to the vector of processing times and due dates an additional item representing position in the instance. Let us define linear position embedding as a method adding number from 0 to $n-1$ divided by n. It is possible to transform vector of embedding by function. Let us denote logarithmic embedding as embedding where embedding on position i is equal to $e_i = log_n(i+1)$. This embedding make big differences between starting position and small differences between end position. Let us denote reversed logarithmic embedding as embedding where embedding on position i is equal to $e_i = 1 - log_n(n-i)$. This embedding make small differences between starting position and big differences between end position. Let us denote combined logarithmic embedding as embedding where embedding on position i is equal to $e_i = \frac{1-log_n(i+1)+log_n(n-i)}{2}$. This embedding makes big differences between starting position, big differences between end position and small difference between middle position.

Criterion normalization

In similar way like subsubsection 4.3 it is fruitful to normalize the target value. First way how to normalize the criterion is to divide it by the same number which is used the normalization of the input. This approach preserves information between processing times, due dates and criterion. But to the detriment of either dividing relatively small number of processing times and due dates by number near to criterion. In this case, the criterion value is near to one, or dividing instance by a relatively small number near to each of due dates and processing times, but have criterion much larger than one. Let us remark, that we can use 1PSMD and NPS0D.

The second way is introduce a different function of normalization and therefore lose connection between processing times, due dates and criterion. Let us define gap value as $\frac{\hat{y}-y}{\hat{y}} \cdot 100$. Let us denote inverse gap as $\frac{1}{1+gap}$. It is important to denote, that inverse gap is in range from zero to one. Now we can calculate inverse gap of EDD and use it as criterion to train by neural network. From our experiments, we know that average gap of EDD is under 40%, in case that maximal gap is 2 it will be criterion on interval from 0.5 to 1. Diagram of regressor with EDD inverse gap is illustrated in Figure 4.4. In principle it is possible as well to create the inverse gap of NBR or different

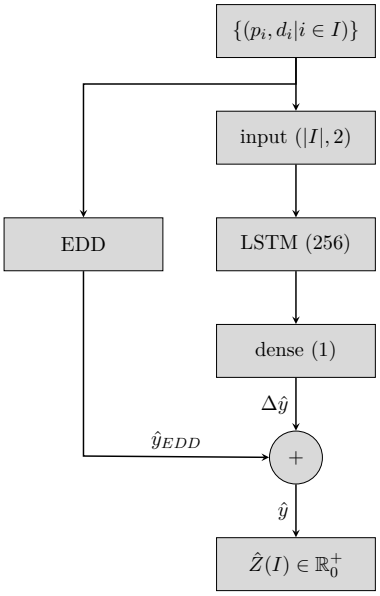


Figure (4.4): Visualization of the regressor scheme, with estimation of inverse gap criterion

heuristic, but we note that this calculation should not be significantly slower than neural network.

4.4 Data set generation

In this section, we introduce a way how to generate data set, define data set parameters and discuss the estimation of required data set size.

In the literature, the standard benchmark for $1||\sum T_j$ is defined over three parameters RDD, TF and p_{max}. Random benchmark instance of $1||\sum T_j$ with size n is generated as n random numbers. Processing time are in interval $[0, p_{max}]$. Define P as the sum of processing times. Now due dates are generated as random numbers from interval $[P(1 - TF - \frac{RDD}{2}), P \cdot (1 - TF + \frac{RDD}{2})]$. In benchmark instances RDD and TF goes from 0.2 to 1 with step 0.2, this make 25 pairs of parameters. p_{max} is typically consider to be 100.

Usually it is hard to collect bigger data set for other machine learning problems. For us is relatively easy to generate data set, since generation of new instance consume only computer time. We have generated data set only with standard benchmark parameters RDD, TF and p_{max}.. But from our experiments implies, that RDD for subproblem generated by Lawler decomposition can grow to higher number than 1.

4.4.1 Generate and evaluate

Classical approach to generate data set is to generate instance and solve it by solver. This approach is straitforward but consume huge amount of computer time. We are implemented Lawler and SDD solver for generating

data training set, also we are used TTBR solver for data set generation. All implemented algorithm run in single thread, we are using Xeon® Gold 6140 and limiting memory to 32 GB.

External implementation of TTBR solver is able to solve instance up to 1200 jobs with average time 120 s. It is important to denote, that time of solving instance is strongly influenced by instance parameters *RDD* and *TF*. For $RDD = 0.2$ and $TF = 0.6$ and size 1200 is average time 1500 s. We have generate our training data set by TTBR solver, with one CPU core and 32 GB ram. Data set generation consume 77 Ms CPU time, which is around 900 days.

Our best implementation of Lawler is able to solve instance up to 200 jobs with average time 20 s, and likewise TTBR there is differences of solution time over parameters *RDD* and *TF*. For ilustration TTBR is able to solve instance with 800 jobs under 20 s. This implementation returns also order of jobs in optimal solution.

4.4.2 Partial generator of training instances

In this section, we propose an alternative way of how to generate a training set. Although Tkindt introduces an efficient and optimized algorithm, obtaining training data set with over this algorithm is extremely slow. However, Lawler decomposition gives us an alternative way of how to generate training data set with much less effort.

The basic idea is to limit the maximal span where it is possible to place a job with maximal processing time. Let us generate random array of processing time and due dates. Fix $span \in \mathbb{N}$ and expect due dates and processing times sorted increasingly, call this as input set. Define a schedule as an empty sequence. Define the j^* as a job with the biggest processing time from the input set and select due date of this job as *span* biggest one from the input set. Generate $span - 1$ random job with smaller processing time and bigger due dates from the input set, call this active set. Remove these jobs from the input set. Find the optimal solution for this subproblem with starting time equal to the sum of remaining processing times. This will generate partially schedule. Split generated partial schedule to two parts, first preceding j^* and second following j^* and also including j^*. Pre extends existing schedule with the second part of the partial schedule, remove the first part of the partial solution from the active set. If the first part of partial schedule contains job with longer processing time than input set solve this part, else repeat this from including jobs to input set, but include jobs only to the size of the input set to the *span*. The algorithm is described in algorithm 3.

Algorithm 3: Partial instance generator

```
   Data: input_set, slack, solve_from_time
   Result: jobs set, jobs order
 1 active_set ← set()
 2 p_max_task ← input_set.get_task_with_higher_p(slack)
 3 active_set.append(p_max_task)
 4 for i in range(slack-1) do
 5 |   active_set.append(input_set.get_next_task())
 6 end
 7 while len(active_set) > 0 do
 8 |   starting_time ← sum(input_set.proc)
 9 |   p_max_position, order ← solve_from_time(start_time,
   |     active_set)
10 |   for k in range(len(order) - 1, p_max_position, - 1) do
11 |   |   schedule.insert(active_set[k])
12 |   |   active_set.remove[k]
13 |   end
14 |   if max(active_set) >= input_set.p[-1] then
15 |   |   continue
16 |   end
17 |   offset ← min(len(input_set), max(1, - len(active_set)))
18 |   tj ← input_set.get_task_with_higher_p(offset)
19 |   while len(active_set) < slack do
20 |   |   active_taskset.append(input_set.get_next_task())
21 |   end
22 end
23 return main(proc, due)
```

Chapter 5

Experimental Results

In this section we evaluate the ability of our data-driven approach to solve $1||\sum T_j$ problem. In the first part, we describe the testing environment. In the second part, we evaluate the ability of the neural network to estimate the optimal value criterion of an instance of $1||\sum T_j$ problem. In the third part, we evaluate the quality of the estimated solution created by Lawler heuristic search.

Let us define several key metrics for the evaluation of estimators for assessment of the quality of results.

Definition 5.1. Mean absolute error (MAE) of vectors of estimations $\hat{\boldsymbol{y}}$ and optimal values $\boldsymbol{y}$ with length $l > 0$ is defined as $\frac{\sum_{i=1}^{l} |y_i - \hat{y}_i|}{l}$.

Definition 5.2. Relative mean absolute error (RELMAE) of vectors of estimations $\hat{\boldsymbol{y}}$ and optimal values $\boldsymbol{y}$ with length $l > 0$ is defined as $\frac{\sum_{i=1}^{l} |y_i - \hat{y}_i|}{l \cdot \sum_{i=1}^{l} y_i}$.

Definition 5.3. Gap of estimate $\hat{y}$ and the optimal value y is defined as $100 \cdot \frac{\hat{y}-y}{\hat{y}}$. Optimality gap expects only estimation $\hat{y}$ greater than optimal values y.

5.1 Experimental setup

We are use a computer cluster for generation of training, testing and validation data sets. This cluster is contain from node with two Xeon® Gold 6140 and 192GB. For generating optimal solutions of data set instances we utilized TTBR algorithm single CPU core and 32GB of RAM. We trained neural network on Tesla V100 graphic card. For training we have use TensorFlow in version 1.12. For the evaluation of neural networks we have use single CPU core of Xeon® Gold 6140 and 32GB of RAM, same as setup for generating of training data set. The code is implemented in Python in version 3.7.

5.2 Instance space

Let us recall, that the standard benchmark for instances $1||\sum T_j$ problem are characterized by RDD, TF and p_{max} parameters. We generate instance with

parameter for RDD equal to $\{0.2, 0.4, 0.6, 0.8, 1.0\}$, TF equal to $\{0.2, 0.4, 0.6, 0.8, 1.0\}$ and p_{max} equal to 100, for n from 5 to 100 for the training set. Where n is count of job in instance. For testing set we generate instances with n from 5 to 1200. Our training data set consists from 5000 instances for each combination of RDD, TF and n. The process of generation of instances with defined RDD, TF and p_{max} is defined in section 4.4. Optimal criterion of $1||\sum T_j$ problem is directly impacted by the size of jobs set n. Let us demonstrate mean value of criterion in Figure 5.1. In this figure, the growing character of $1||\sum T_j$ criterion on instance with size from 5 to 100 is shown. This characteristic has impact on the performance metrics of the regressor.

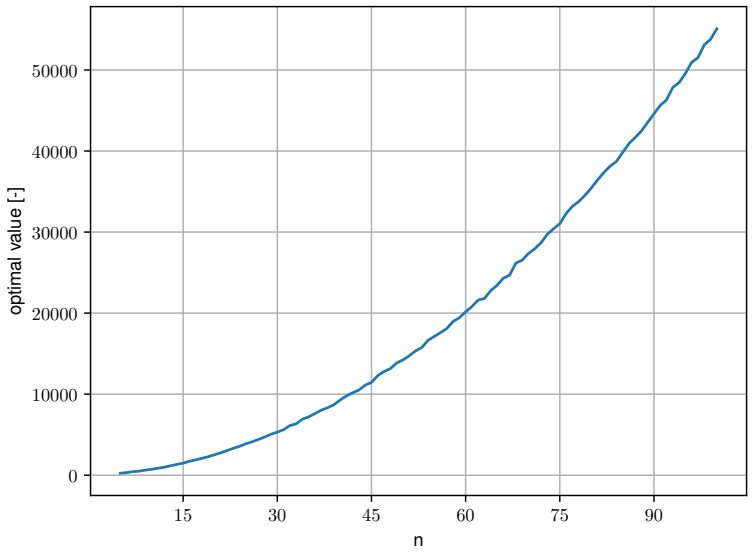


Figure (5.1): Mean value of optimal criterion on instance with respect to size.

5.3 Evaluation of the network

In this section, we will introduce end evaluate the individual experiments with the neural network. Let us define the baseline for experiments as follows. Training set contains instances with n from 5 to 100, with p_{max} equal to 100 and RDD equal to $\{0.2, 0.4, 0.6, 0.8, 1.0\}$, TF equal to $\{0.2, 0.4, 0.6, 0.8, 1.0\}$ generated by method *Generate and evaluate*. For each combination of RDD, TF and n we have 5000 training instances. The total number of samples is 11 875 000. The impact of input normalization described in subsubsection 4.3 by 1PSMD. Also criterion normalization described in subsubsection 4.3 by 1PSMD. Data is sorted in EDD and there is no information about position in instance on input. LSTM layer has a size 256. For training, we use Adam optimizer with learning rate 0.0001. Batch size is 250 instances per batch. We train the neural network with 100 epochs and a select epoch with the smallest error on the test set. All results described below are measured on the validation data set.

We have proposed several experiments to test the abilities of the network, the first experiment is focused on measuring the ability of neural network estimation with different data set size. The second experiment is focused on measuring the influence of input data sorting. The third experiment is focused on measuring the influence of position embedding. The fourth experiment is focused on measuring the influence of criterion normalization. The fifth experiment is focused on measuring the influence of training set generated by the partial generator. The sixth experiment is focused on measuring the ability of the neural network to extrapolate results on testing set generated with different parameters than the training set.

5.3.1 Impact of the Data set size

We expect that with the growing number of instance used for training the mean absolute error of neural network on the test set will decrease. We trained neural network with 200, 500, 1000, 2000, 5000, 10000 training samples for each combination of *RDD*, *TF* and n. All other parameters are the same like in the baseline defined in section 5.3.

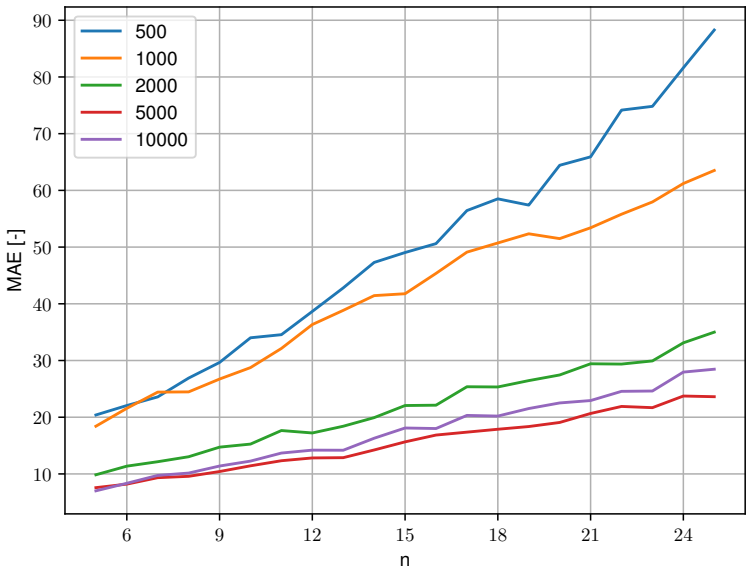


Figure (5.2): Mean absolute error of neural network trained on 500, 1000, 2000 and 10000 samples. For n from 5 to 25.

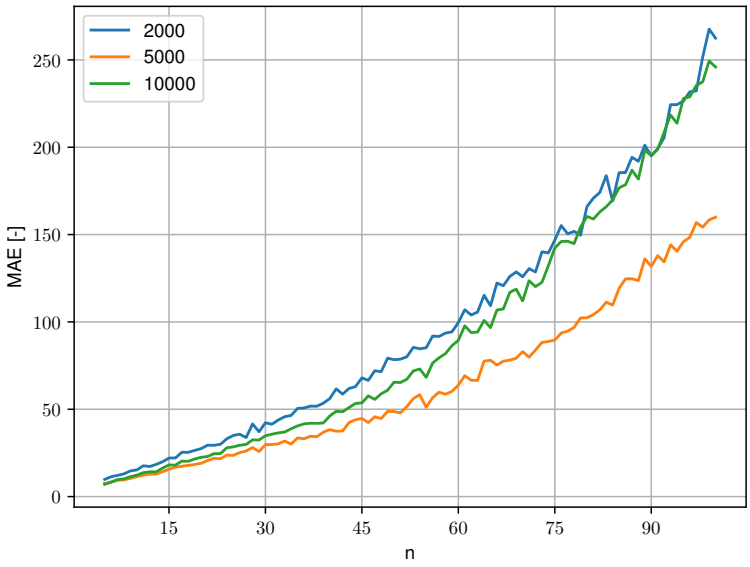


Figure (5.3): Mean absolute error of neural network trained on 2000, 5000 and 10000 samples. For n from 5 to 100.

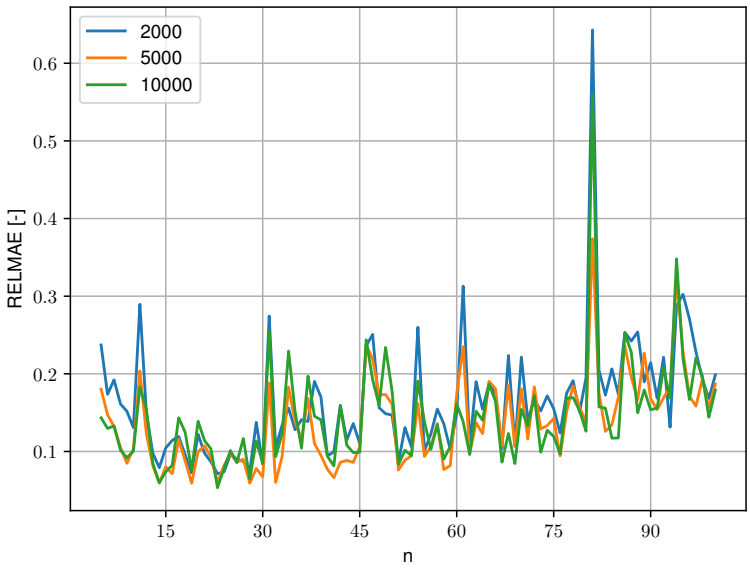


Figure (5.4): Relative mean absolute error of neural network trained on 2000, 5000 and 10000 samples for combination of *RDD*, *TF* and n

Mean absolute error of neural network trained only on 200 samples per combination of *RDD* and *TF* grow rapidly in comparison to the other neural networks. Due to this, we have excludeed it from the comparison. In Figure 5.2 is shown mean absolute error of neural network trained on 500, 1000, 2000, 10000 samples in detail for n on interval from 5 to 25. In Figure 5.3 is shown the mean square error of neural network trained on 500, 1000, 2000, 10000

samples for each combination RDD, TF, n for n on interval from 5 to 100. It is important to note, that optimal value of criterion grow with size of instance (see Figure 5.1), due to this it is expectable that mean absolute error grows with size of instanc as well.

From our experiment it is shown, that mean absolute error of neural network on the validation set decreases to zero with the growing data set size. Without one exception, that neural network trained on 5000 samples is better than neural network trained on 10000 samples.

In Figure 5.4 is shown RELMAE of neural networks. It is important to note that MAE of criterion grows for all neural networks, in contrast to RELMAE which is relatively constant over all n.

5.3.2 Input data sorting

We expect, that EDD sorting of input data is not inferior to another possible sorting of instance. We have proposed several sorting methods for the $1||\sum T_j$ problem in subsubsection 4.3. The results of neural networks trained with different sorting methods are presented in Figure 5.5. All other parameters of the neural network are the same as in the baseline model.

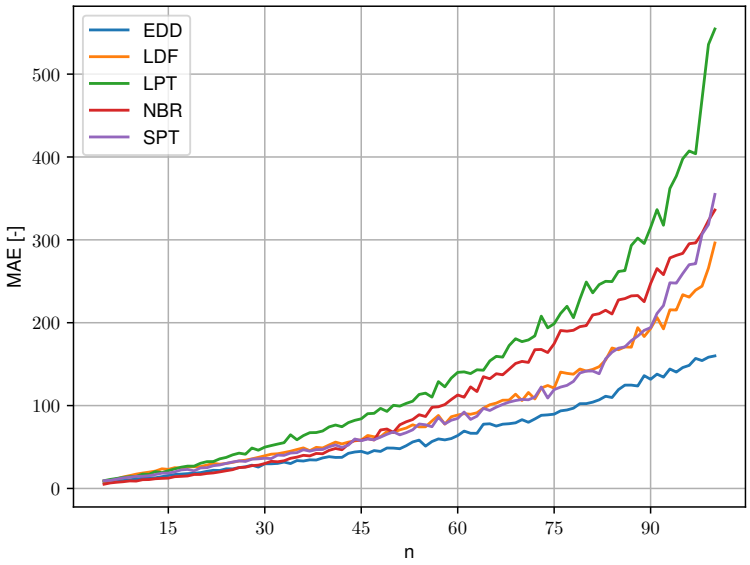


Figure (5.5): Mean absolute error of neural network trained with input sample sorted by EDD, SPT, LPT, LDD, NBR.

Neural network trained with EDD has average MAE of criterion 70, neural networks trained with LDD and SPT sorting have MAE of criterion around 100, NBR has average MAE around 120 and LPT has average MAE around 150. In this experiment, the EDD sorting has the best performance over all sorting methods.

5.3.3 Positional encoding

We proposed several methods to encode position. We train neural network with all method introduced in subsubsection 4.3. All other parameters of the neural network are the same as in the baseline model.

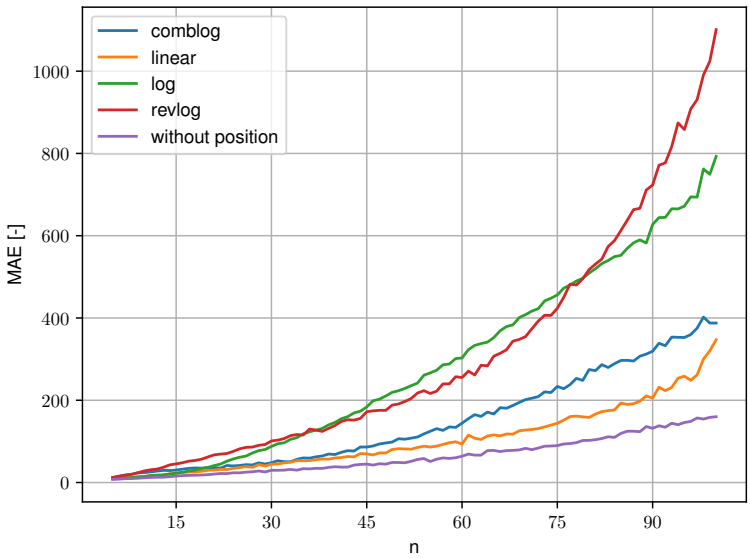


Figure (5.6): Mean absolute error of neural network trained with linear, logarithmic, reversed logarithmic and combined logarithmic position encoding.

In our experiments, the best results were obtained with neural network without position encoding followed by a neural network with linear position encoding and combined logarithmic position encoding. Visualization of MAE aggregated by n is in Figure 5.6.

5.3.4 Criterion normalization

In this experiment we expected, that 1PSMD criterion normalization function is not inferior to other possible criterion normalization function. We proposed two alternative function of $1||\sum T_j$ criterion normalization in subsubsection 4.3. The result of neural network trained with different criterion normalization function is presented in Figure 5.7. All other parameters of neural network are same as in baseline model.

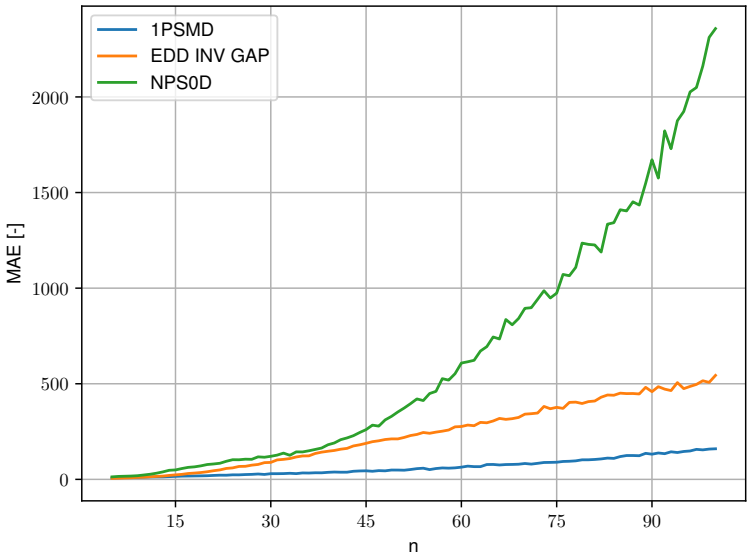


Figure (5.7): Mean absolute error of neural network trained with input sample sorted by 1PSMD, NPS0D, EDD INV GAP.

Neural network trained with 1PSMD has average MAE of criterion 70, neural networks trained with EDD INV GAP has average MAE of criterion around 270 and neural network trained with NPS0D has average MAE of criterion around 700. In this experiment, 1PSMD criterion normalization function has the best performance over all sorting methods.

5.3.5 Partial data generator

We trained the neural network on data set generated by partial generator with span set to 20. We expected that neural network trained on data set generated by partial generator is able to predict criterion on validation data set generated by generate and evaluate approach. In Figure 5.8 there is evaluation of model trained on partial and generate and evaluate data set.

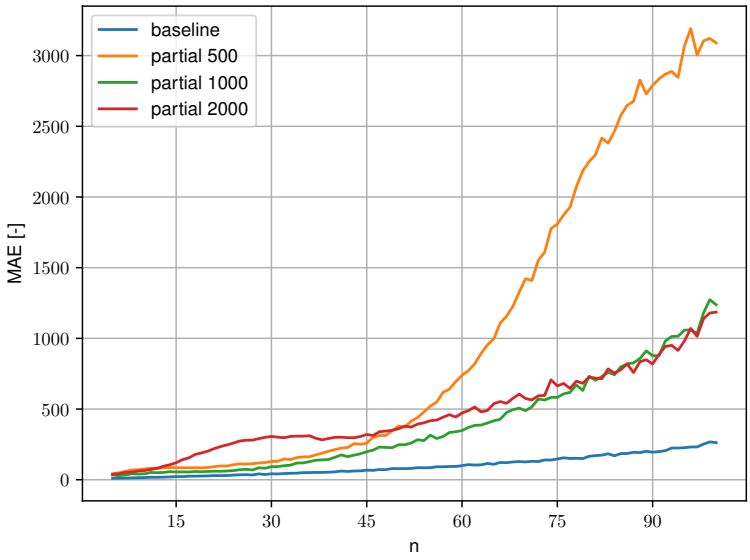


Figure (5.8): Mean absolute error of neural network trained on data set generated by generate and evaluate method and on data set generated by partial generator.

Neural networks trained on partial data set have worse ability to estimate criterion than neural network trained on generate and evaluate data set. Neural networks trained on 1000 and 2000 samples have a better ability than a neural network trained on 500 samples. However, there are no improvements between 1000 and 2000 samples.

5.3.6 Extrapolation

Motivation for extrapolation is easy, we want to use neural network on different instance than from training data set. Usually it is too hard to collect data set for some part of instance space. In our case time to compute instance is grow with growing n. Also time to compute instance is grow with p_{max}. There are two way how to save time and use neural network for extrapolation. First way is use neural network for estimation on bigger n, than maximal n from training data set. Second way is to use neural network for estimation on instances with different p_{max}, than p_{max} from training data set. Let us recall, that p_{max} is set to 100 in training data set.

Best result for the extrapolation over n have the baseline model and model with EDD inverse gap criterion normalization. Extrapolation ability of neural network to estimate criterion on n bigger than in training data set is shown in Figure 5.9.

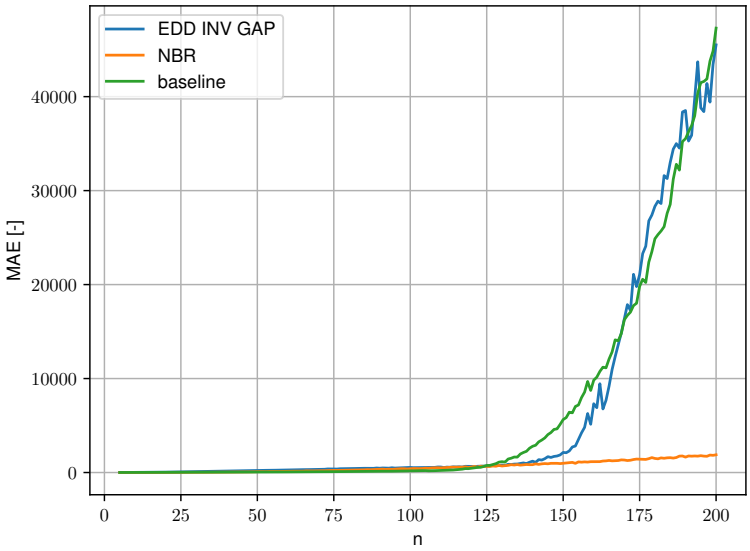


Figure (5.9): Mean absolute error of neural network for n from 5 to 200.

The neural network trained with EDD inverse gap provides better estimation of the optimal value in interval from 125 to 160, although that the baseline mode has better prediction for $n \leq 100$, i.e. the range of the training set. NBR heuristic outperforms both neural networks on n greater than 125. From our later experiment conclude that prediction to n equal 150 is good enough for use in Lawler heuristic search.

The ability of neural network to estimate optimal value for instances with $p_{max} = 2000$ is shown in Figure 5.10 and Figure 5.11. Let us recall that in the training set the p_{max} equals to 100.

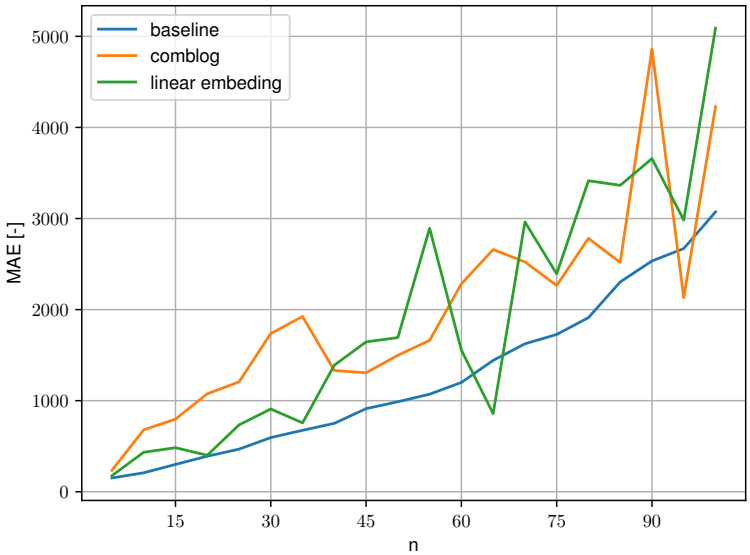


Figure (5.10): Mean absolute error of neural network for n from 5 to 100.

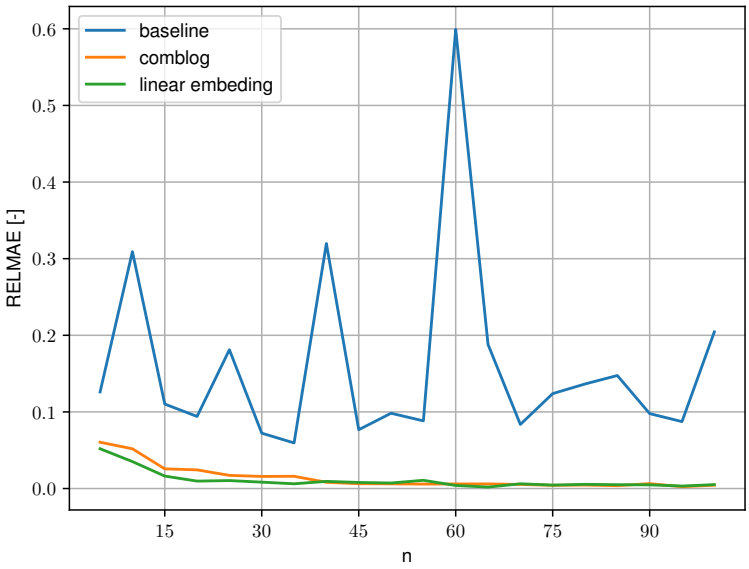


Figure (5.11): Relative mean absolute error of neural network for n from 5 to 100.

It is important to note, that the average optimal value of instances with $p_{max} = 2000$ is bigger than for standard instance. Instance with $p_{max} = 200$ and $n = 100$ has average optimal value 1000000, let us recall that standard instance has average optimal value 50000. We judge results of the neural network on instances with $p_{max} = 2000$ as successful, MAE grows around 15 times in comparison with $p_{max} = 100$ instances. However, the average optimal value grows around 20 times. It means that the relative mean absolute error decrease.

5.4 Evaluation of the data-driven algorithm

In this section, we evaluate our approach and its ability to solve the problem. First, we analyze a weak point of TTBR optimal solver considered as the state-of-the-art exact method. Then, we compare the results of this optimal solver with ours approach. As the second, we analyze state-of-the-art classical heuristic and compare optimality gaps between the heuristic and our approach. As the third, we compare the integration of classical heuristic into Lawler heuristic search with the integration of neural network into Lawler heuristic search. As the fourth, we evaluate Lawler heuristic search enhanced with limited discrepancy search.

5.4.1 TTBR

First, let us demonstrate the runtime of TTBR. In Figure 5.12 is shown average solving time for TTBR, with respect to growing n.

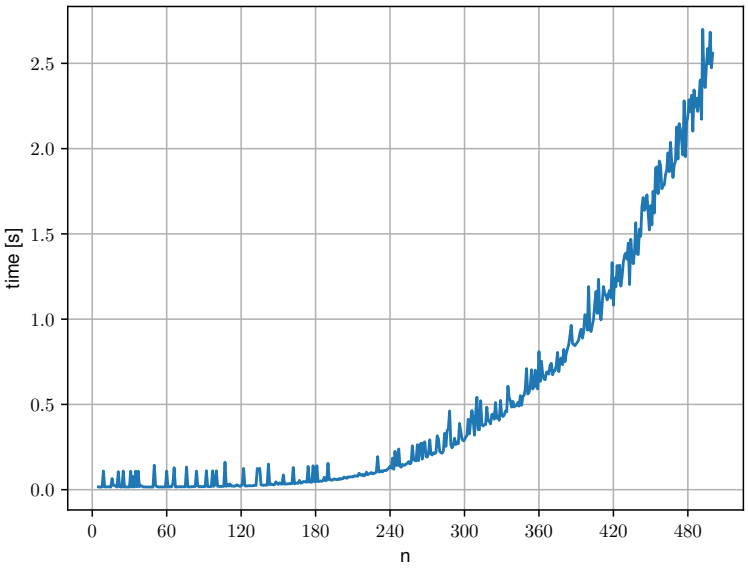


Figure (5.12): Runtime of the TTBR.

It is known, that TTBR algorithm needs different time to solve the instance over parameters RDD and TF. In Table 5.1 there are average times for all RDD and TF combination for n from 5 to 500.

RDD/TF	0.2	0.4	0.6	0.8	1.0
0.2	0.075	2.155	**5.160**	1.643	0.035
0.4	0.036	0.362	1.635	0.051	0.036
0.6	0.036	0.056	0.473	0.035	0.036
0.8	0.036	0.038	0.068	0.035	0.037
1.0	0.039	0.035	0.036	0.035	0.038

Table (5.1): TTBR runtimes with respect to instance parameters.

Combination of RDD equals to 0.2 and TF equals to 0.6 is the hardest for TTBR. Let us demonstrate in Figure 5.13 the average runtime of TTBR for $RDD = 0.2$ and $TF = 0.6$.

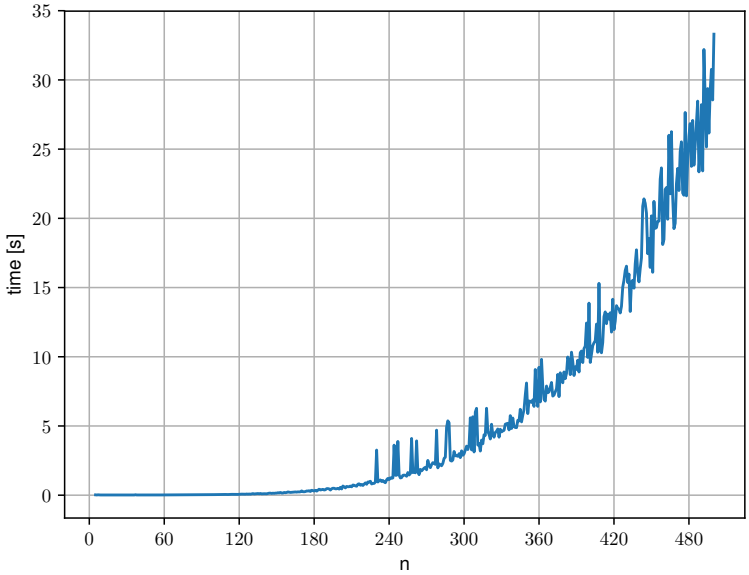


Figure (5.13): Runtime of the TTBR for $RDD = 0.2$ and $TF = 0.6$ with respect to n.

TTBR is pseudo-polynomial algorithm in terms of p_{max} parameters. Let us demonstrate it in Figure 5.14, where the average runtime of the TTBR for $RDD = 0.2$ and $TF = 0.6$ with p_{max} grows from 100 to 6400. The average runtime of the TTBR for $RDD = 0.2$ and $TF = 0.6$ and p_{max} changed from 100 to 2000 is shown in Figure 5.15.

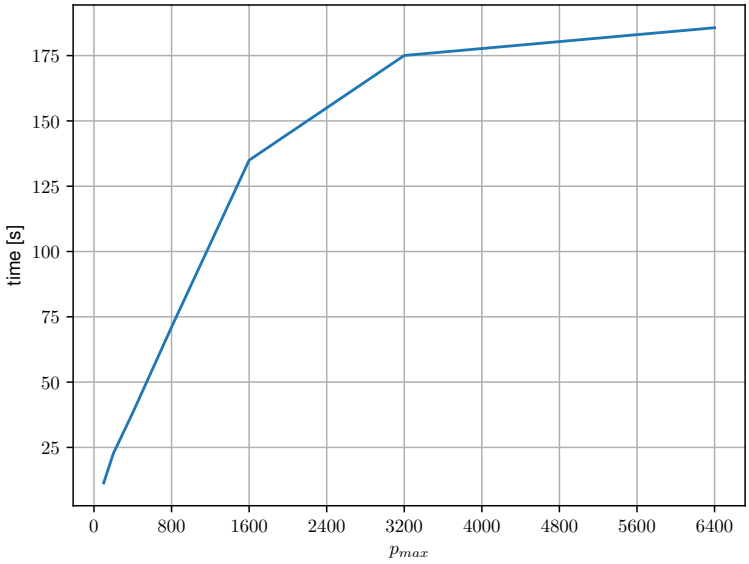


Figure (5.14): Runtime of the TTBR for $RDD = 0.2$ and $TF = 0.6$ with respect to p_{max}.

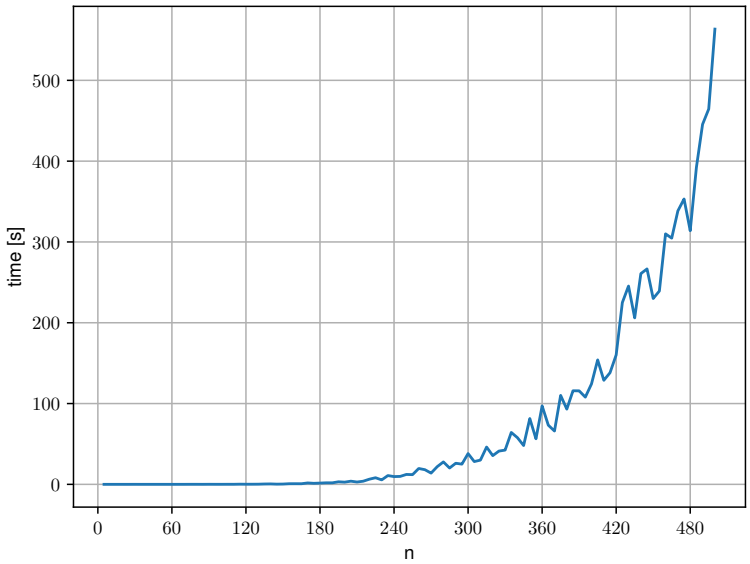


Figure (5.15): Runtime of the TTBR for $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 2000$ with respect to n.

The execution time of the TTBR grows rapidly with the p_{max} until 2000. Also, the execution time of the TTBR is more than 10 times greater with $p_{max} = 2000$ than with $p_{max} = 100$.

Let us demonstrate the execution time of TTBR and Lawler decomposition search with the neural network as a regressor. As first, we demonstrate execution times on instances with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 100$ in Figure 5.16. As second, we demonstrate execution times on instances with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 100$ in Figure 5.17.

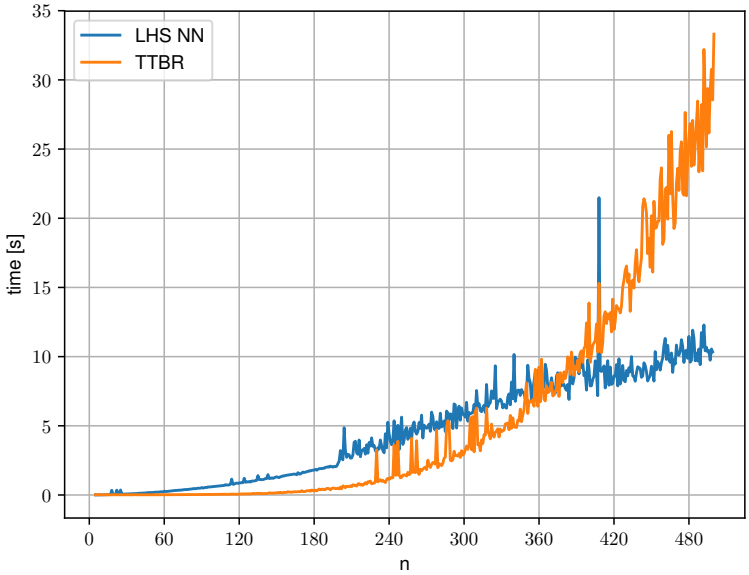


Figure (5.16): Runtime of the TTBR and Lawler heuristic search with neural network as regressor for $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 100$ with respect to n.

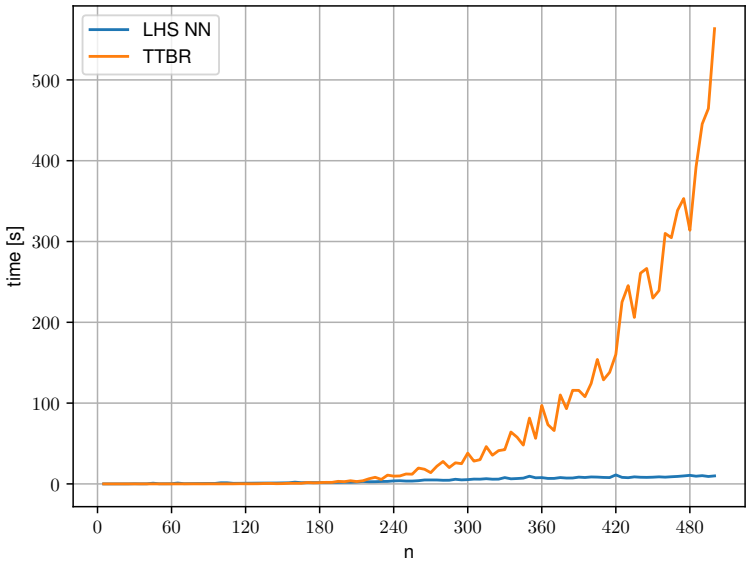


Figure (5.17): Runtime of the TTBR and Lawler heuristic search with neural network as regressor for $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 2000$ with respect to n.

Lawler heuristic search is faster from n 380 on instances with parameters $RDD = 0.2$ and $TF = 0.6$ and $p_{max} = 100$. And Lawler heuristic search is faster from n 180 on instances with parameters $RDD = 0.2$ and $TF = 0.6$ and $p_{max} = 2000$.

5.4.2 Comparison to classic heuristics

In the following experiment, we compare the gap from the optimal solution of EDD, NBR, and Lawler heuristic search with the neural network as a regressor. In Figure 5.18 we present result on parameters $RDD = 0.2$ and $TF = 0.6$ and $p_{max} = 100$ and n to 200. In Figure 5.19 we present result on parameters $RDD = 0.2$ and $TF = 0.6$ and $p_{max} = 2000$ and n to 200.

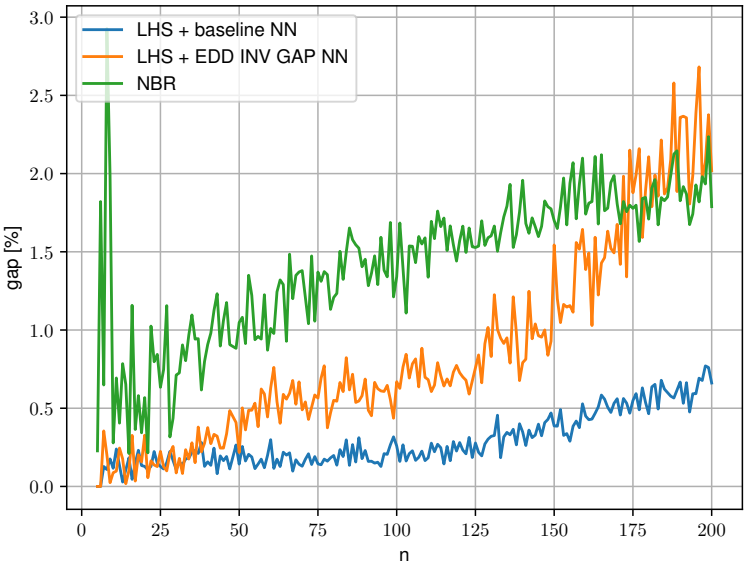


Figure (5.18): Optimality gap of estimators on instance with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 100$.

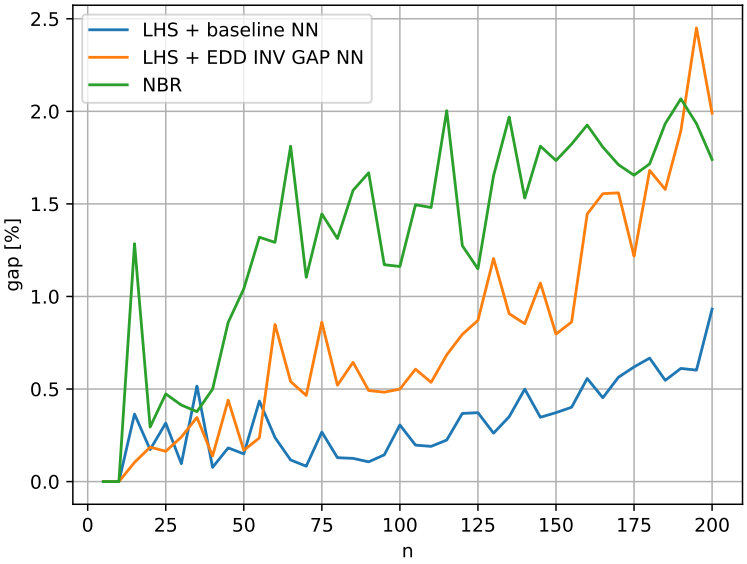


Figure (5.19): Optimality gap of estimators on instance with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 2000$.

Lawler heuristic search with baseline neural network outperforms NBR and Lawler heuristic search with the neural network with edd inverse gap criterion normalization for either instance with $pmax = 100$ and $pmax = 2000$.

5.4.3 Lawler heuristic search with classic regressor

In the following experiment, we compare the gap from the optimal solution of Lawler heuristic search with the EDD and NBR as a regressor and Lawler heuristic search with the neural network as a regressor. In Figure 5.22 we present result on parameters $RDD = 0.2$ and $TF = 0.6$ and $p_{max} = 100$ and n to 200. In Figure 5.23 we present result on parameters $RDD = 0.2$ and $TF = 0.6$ and $p_{max} = 2000$ and n to 200.

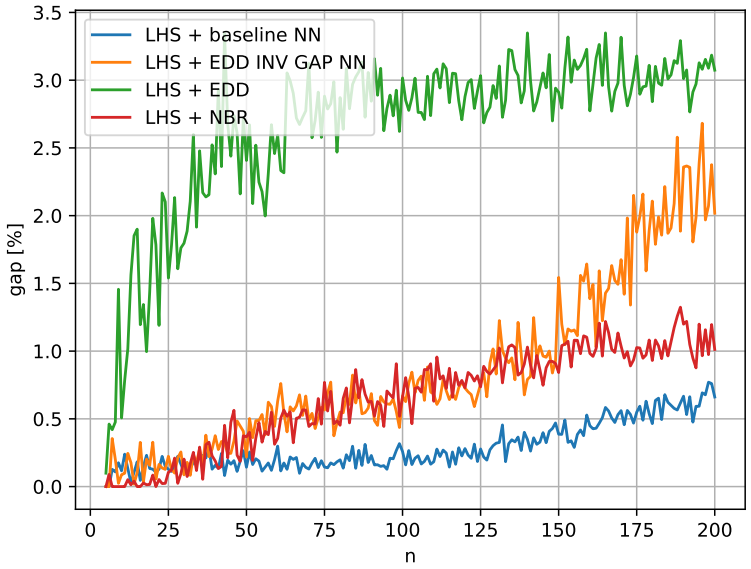


Figure (5.20): Optimality gap of estimators on instance with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 100$.

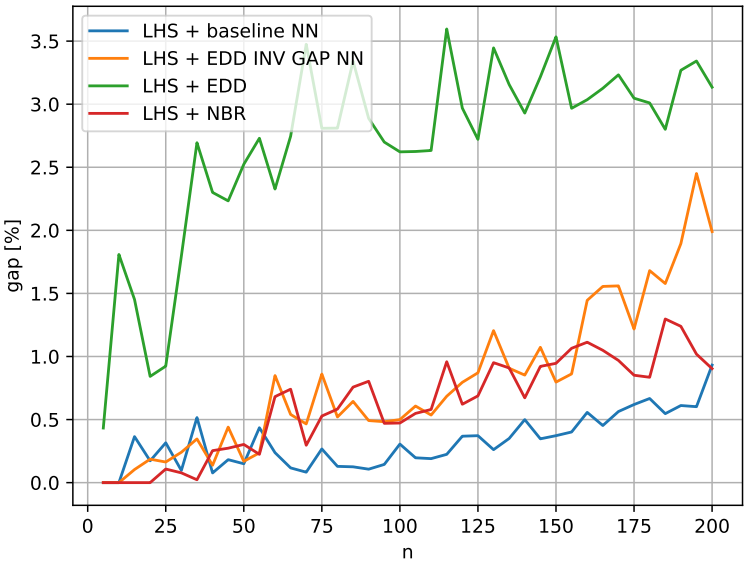


Figure (5.21): Optimality gap of estimators on instance with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 2000$.

Lawler heuristic search with baseline neural network outperform NBR and Lawler heuristic search with neural network with EDD inverse gap criterion normalisation for either instances with $p_{max} = 100$ and $p_{max} = 2000$.

In Table 5.2, there is the comparison of NBR, Lawler heuristic search with NBR as a regressor, Lawler heuristic search with our baseline neural network model as regressor and TTBR. In the table, there is a comparison of the optimality gap, optimality gap standard deviation, and runtime. Results in tabs are reported for instances with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 2000$. Lawler heuristic search with the neural network as regressor outperform all other heuristics in quality of solution and outperform TTBR on instances bigger than 180.

5.4.4 Dependency between MAE and GAP

In the following experiment, we compare the gap from the optimal solution of Lawler heuristic search with the neural networks trained with different size of training sets. We expect that the gap of Lawler heuristic search depends on the MAE of the used neural network as a regressor. In Figure 5.22 and Figure 5.23, there are evaluations of Lawler heuristic search with neural networks trained on 200, 500, 1000, 2000, and 5000 samples per each combination of RDD, TF, and n. MAE of neural network is shown in subsection 5.3.1.

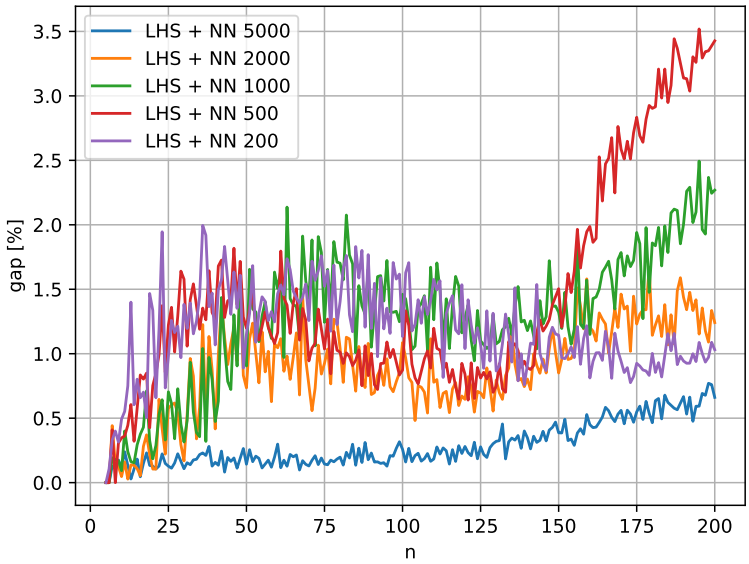


Figure (5.22): Gap of estimators on instance with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 100$.

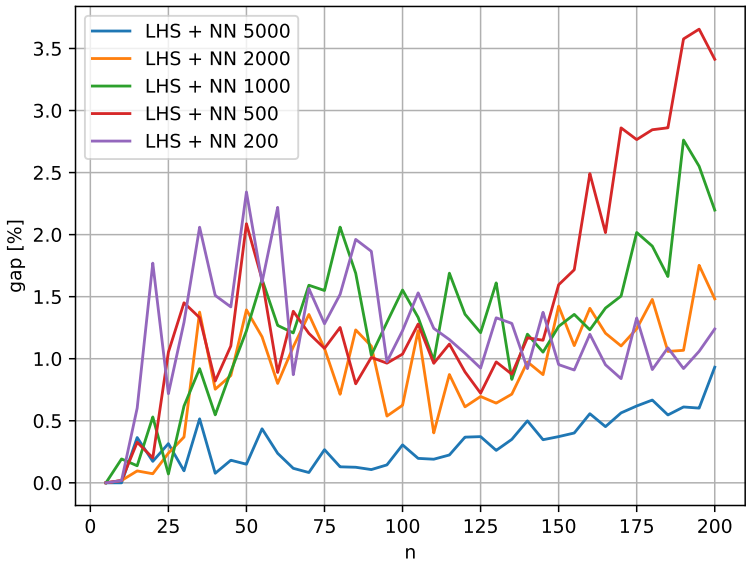


Figure (5.23): Gap of estimators on instance with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 2000$.

On either instances with $p_{max} = 100$ and $p_{max} = 2000$ has best result model with 5000 training samples, this model also has the smallest MAE of all models. What is surprising is that the neural network trained on 200 outperform neural network trained on 500, 1000, and 2000 samples for extrapolation on n greater than 150.

5.4.5 Lawler search space alternatives

In the following experiment, we show the influence of limited discrepancy search in Lawler heuristic search. We compare gap to the optimal solution and runtime of algorithms. In the first experiment, we show the influence of limited discrepancy search on Lawler heuristic search with EDD as a regressor. In the second experiment, we show the influence of limited discrepancy search on Lawler heuristic search with baseline neural network as a regressor.

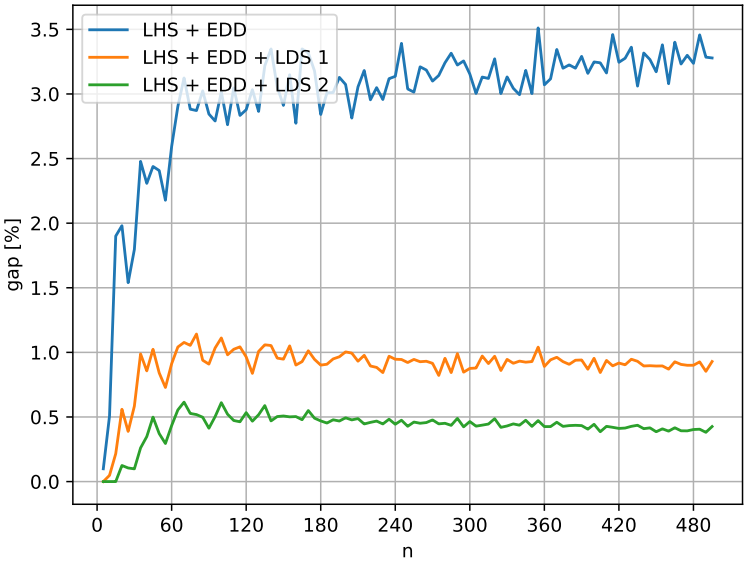


Figure (5.24): Optimality gap of estimators on instance with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 100$ with respect to allowed discrepancy.

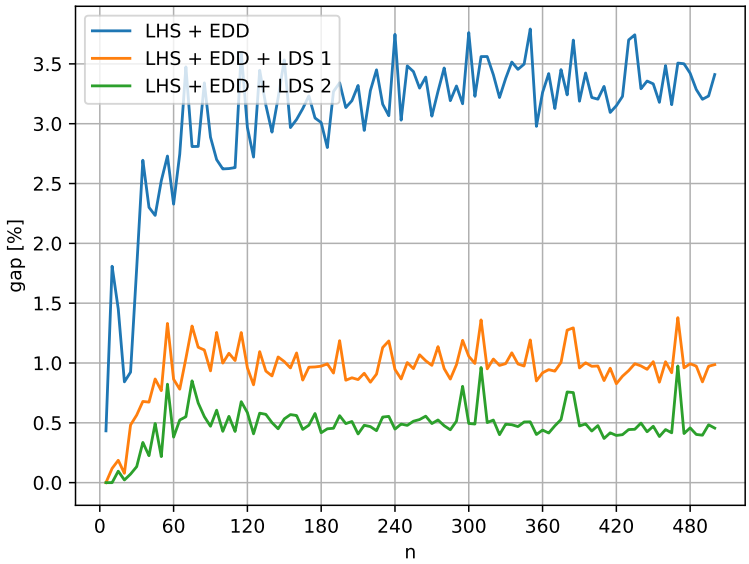


Figure (5.25): Optimality gap of estimators on instance with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 2000$ with respect to allowed discrepancy.

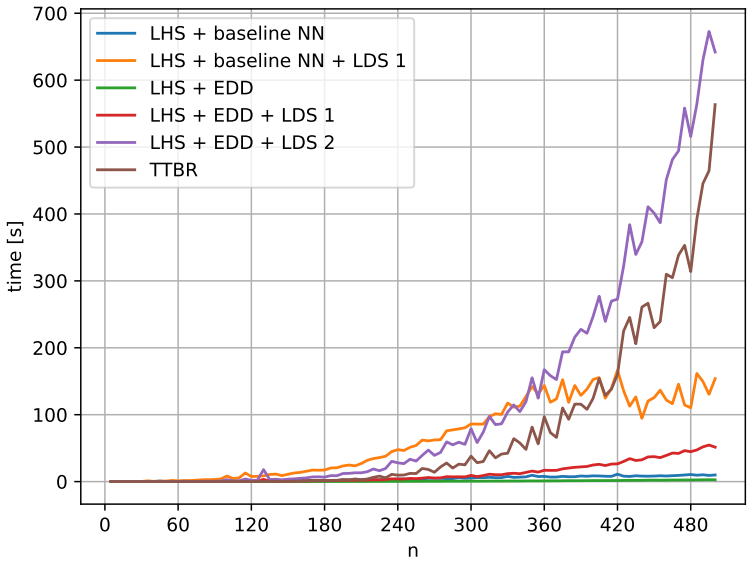


Figure (5.26): Runtime of estimators on instance with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 2000$ with respect to allowed discrepancy.

Discrepancy equals to 1 decrease gap of Lawler heuristic search with EDD regressor from around 3% to around 1%. Also, this estimator has still better runtime than TTBR on the instance with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 2000$. Discrepancy equals to 2 decrease gap to around 0.5%, but runtime of this estimator is longer than the runtime of TTBR.

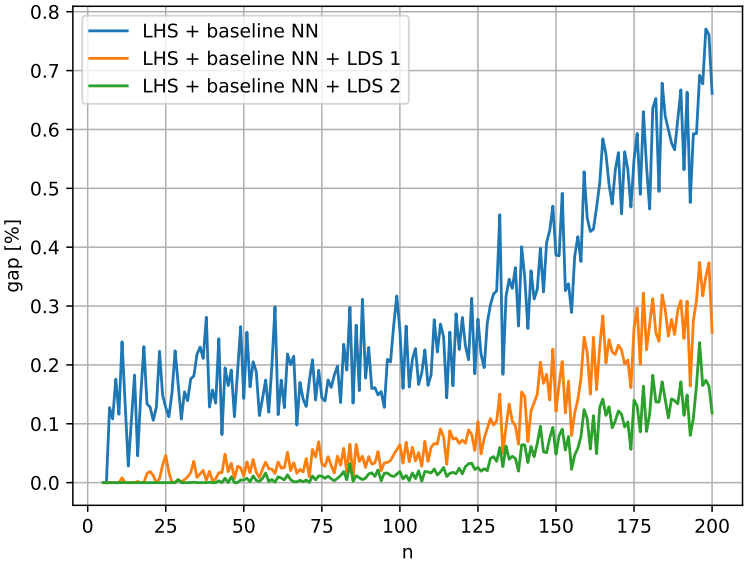


Figure (5.27): Optimality gap of estimators on instance with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 100$ with respect to allowed discrepancy.

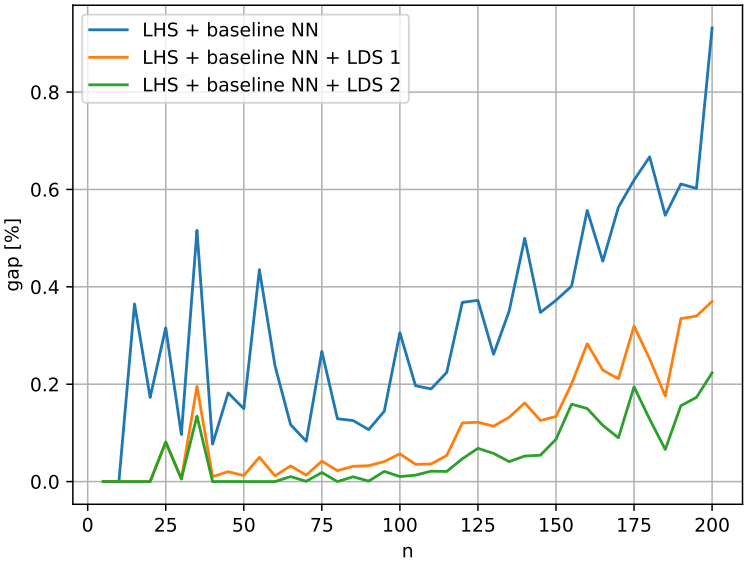


Figure (5.28): Optimality gap of estimators on instance with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 2000$ with respect to allowed discrepancy.

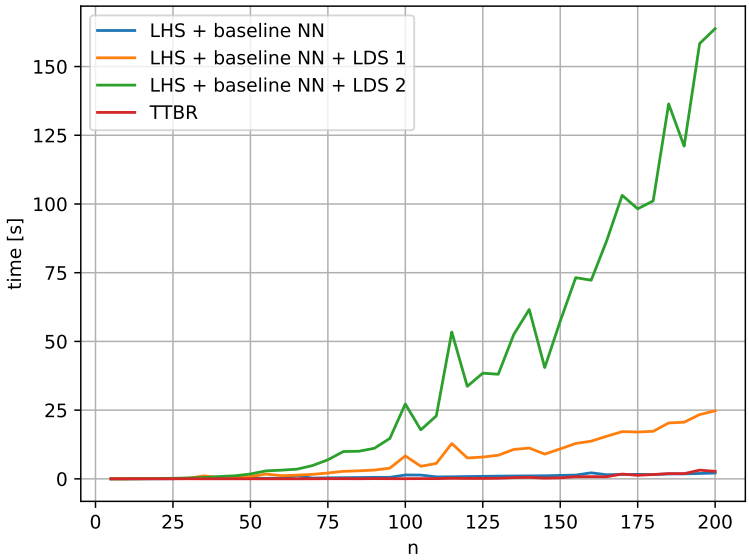


Figure (5.29): Runtime of estimators on instance with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 2000$ with respect to allowed discrepancy.

Discrepancy equals to 1 decrease gap of Lawler heuristic search with EDD regressor to maximally 0.4% on n up to 200. But runtime of this estimator is longer than the runtime of TTBR. Discrepancy equals to 2 decrease gap of Lawler heuristic search with EDD regressor to maximally 0.25% on n up to 200.

5.5 Experiment summary

The experiment results show the following:

- MAE of the proposed neural network decreases with growing size of the training data set.
- The proposed neural network has ability to generalize its predictions for instances up to 25% larger than used during the training.
- Lawler heuristic search is faster than TTBR for instances with $p_{max} = 2000$ and for n larger than 180.
- The optimality gap of Lawler heuristic search decreases with decreasing MAE of the regressor.
- Lawler heuristic search with the proposed neural network as regressor is better both than NBR and Lawler heuristic search with EDD or NBR.
- Lawler heuristic search with the proposed neural network as regressor has average optimality gap 1% on instances for n to 200 and $p_{max} = 2000$.

n	**TTBR** time [s]	**NBR** gap [%]	**NBR** time [s]	**LHS NBR** gap [%]	**LHS NBR** time [s]	**LHS NN** gap [%]	**LHS NN** time [s]
5	0.02	0.00 (±0.00)	0.00	0.00 (±0.00)	0.00	0.00 (±0.00)	0.00
10	0.02	0.00 (±0.00)	0.00	0.00 (±0.00)	0.00	0.00 (±0.00)	0.01
15	0.02	1.28 (±1.74)	0.00	0.00 (±0.00)	0.01	0.36 (±0.45)	0.02
20	0.02	0.29 (±0.51)	0.00	0.00 (±0.00)	0.01	0.17 (±0.32)	0.03
25	0.02	0.47 (±0.81)	0.00	0.11 (±0.21)	0.01	0.32 (±0.41)	0.05
30	0.04	0.41 (±0.41)	0.00	0.08 (±0.16)	0.02	0.10 (±0.13)	0.18
35	0.02	0.38 (±0.32)	0.00	0.02 (±0.05)	0.02	0.52 (±0.87)	0.09
40	0.02	0.50 (±0.85)	0.00	0.25 (±0.69)	0.03	0.08 (±0.10)	0.12
45	0.03	0.86 (±0.66)	0.00	0.27 (±0.47)	0.04	0.18 (±0.29)	0.89
50	0.03	1.04 (±0.98)	0.00	0.30 (±0.53)	0.05	0.15 (±0.24)	0.17
55	0.02	1.32 (±1.41)	0.01	0.22 (±0.37)	0.07	0.44 (±0.60)	0.19
60	0.02	1.29 (±0.99)	0.01	0.68 (±0.66)	0.07	0.24 (±0.35)	0.23
65	0.02	1.81 (±1.29)	0.01	0.74 (±0.70)	0.08	0.12 (±0.13)	1.08
70	0.03	1.10 (±0.63)	0.01	0.30 (±0.42)	0.11	0.08 (±0.08)	0.29
75	0.06	1.45 (±0.95)	0.01	0.53 (±0.49)	0.13	0.27 (±0.30)	0.36
80	0.06	1.31 (±0.74)	0.01	0.58 (±0.37)	0.16	0.13 (±0.11)	0.41
85	0.04	1.57 (±0.72)	0.01	0.76 (±0.51)	0.17	0.13 (±0.13)	0.47
90	0.09	1.67 (±0.75)	0.01	0.80 (±0.71)	0.17	0.11 (±0.14)	0.55
95	0.06	1.17 (±0.64)	0.01	0.47 (±0.41)	0.24	0.14 (±0.15)	0.56
100	0.06	1.16 (±0.59)	0.05	0.47 (±0.35)	0.25	0.31 (±0.29)	1.41
105	0.09	1.50 (±0.58)	0.02	0.55 (±0.28)	0.27	0.20 (±0.14)	1.38
110	0.08	1.48 (±0.75)	0.02	0.58 (±0.38)	0.32	0.19 (±0.11)	0.71
115	0.22	2.00 (±0.62)	0.05	0.96 (±0.57)	0.35	0.22 (±0.27)	0.75
120	0.15	1.27 (±0.42)	0.02	0.62 (±0.35)	0.43	0.37 (±0.24)	0.84
125	0.16	1.15 (±0.51)	0.02	0.69 (±0.34)	0.39	0.37 (±0.34)	0.90
130	0.21	1.65 (±0.81)	0.14	0.95 (±0.83)	0.54	0.26 (±0.22)	0.98
135	0.44	1.97 (±0.66)	0.03	0.91 (±0.42)	0.54	0.35 (±0.25)	1.04
140	0.53	1.53 (±0.61)	0.03	0.67 (±0.40)	0.63	0.50 (±0.20)	1.07
145	0.26	1.81 (±0.92)	0.03	0.92 (±0.64)	0.62	0.35 (±0.24)	1.11
150	0.37	1.73 (±0.54)	0.03	0.95 (±0.45)	0.61	0.37 (±0.34)	1.25
155	0.78	1.82 (±0.42)	0.03	1.06 (±0.27)	0.77	0.40 (±0.26)	1.37
160	0.80	1.93 (±0.41)	0.03	1.11 (±0.51)	0.78	0.56 (±0.39)	2.20
165	0.77	1.81 (±0.43)	0.04	1.05 (±0.43)	0.83	0.45 (±0.34)	1.49
170	1.75	1.71 (±0.61)	0.04	0.97 (±0.52)	0.99	0.56 (±0.36)	1.61
175	1.28	1.65 (±0.51)	0.04	0.85 (±0.39)	1.07	0.62 (±0.38)	1.62
180	1.57	1.72 (±0.71)	0.04	0.84 (±0.40)	1.02	0.67 (±0.47)	1.57
185	1.91	1.93 (±0.59)	0.04	1.30 (±0.68)	1.21	0.55 (±0.29)	1.78
190	1.89	2.07 (±0.56)	0.05	1.24 (±0.57)	1.22	0.61 (±0.28)	1.82
195	3.17	1.93 (±0.77)	0.05	1.02 (±0.37)	1.50	0.60 (±0.34)	1.95
200	2.73	1.74 (±0.75)	0.05	0.90 (±0.45)	1.35	0.93 (±0.48)	2.13

Table (5.2): Comparison of TTBR, NBR, Lawler heuristic search with NBR and Lawler heuristic search with baseline neural network on instances with $RDD = 0.2$, $TF = 0.6$ and $p_{max} = 2000$.

Chapter 6

Conclusion

Classical approaches of solving $\mathcal{NP}$-hard problems in the literature can be divided into optimal and heuristic approaches. The optimal methods consume unfruitful time to find an optimal solution. Heuristic algorithms often rely on hand-crafted rules that are developed by humans which gain experience by looking at the solutions. The aim is to automatize this process. We introduced a data-driven approach able to find a utilizing the knowledge gained from based on previously solved instances.

We proposed experiments for evaluating of the possibility of using a neural network to estimate the optimal objective value of instances. In these experiments, we evaluated several configurations of neural network and selected suitable candidates. We proposed Lawler heuristic search with the neural network as a regressor. We evaluated state-of-the-art optimal solver and identify hard instances for this solver. Lawler heuristic search is faster than the optimal solver on these instances, and the difference in runtime grows rapidly with the growing size of the instance. Lawler heuristic search with a neural network is better than state-of-the-art heuristic algorithm NBR in the quality of the solution. Moreover, Lawler heuristic search with a neural network is better than Lawler heuristic search with EDD or NBR in quality of the solution. We enhanced Lawler heuristic search by limited discrepancy search and evaluated it with EDD and neural network as a regressor.

From our point of view, there are several ways to improve the results. The first possibility is to train the neural network on instances with a larger size. Unfortunately, this procedure consumes a huge amount of time. We expect from this step improved ability of the neural network to provide reasonable estimations of optimal values on instances with larger size. Due to this, we expect improved results of Lawler heuristic search on instances with the bigger size. The second possibility is to improve the partial generator. We want to introduce partial generator with flowing slack value. We expect from this step faster generation of data set with parameters similar to the standard data set. The third possibility is to introduce a heuristic search based on SDD.

Bibliography

[ABCC06] David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook, *Concorde TSP solver*, 2006.

[AK96] Solomon R Antony and Christos Koulamas, *Simulated annealing applied to the total tardiness problem*, Control and Cybernetics **25** (1996), 121–130.

[BBHS99] Andreas Bauer, Bernd Bullnheimer, Richard F Hartl, and Christine Strauss, *An ant colony optimization approach for the single machine total tardiness problem*, Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), vol. 2, IEEE, 1999, pp. 1445–1450.

[BDAF96] M Ben-Daya and M Al-Fawzan, *A simulated annealing approach for the one-machine mean tardiness scheduling problem*, European Journal of Operational Research **93** (1996), no. 1, 61–67.

[CLG09] TC Edwin Cheng, Alexander A Lazarev, and Evgeny R Gafarov, *A hybrid algorithm for the single-machine total tardiness problem*, Computers & Operations Research **36** (2009), no. 2, 308–315.

[DCGP04] Federico Della Croce, Andrea Grosso, and Vangelis Th Paschos, *Lower bounds on the approximation ratios of leading heuristics for the single-machine total tardiness problem*, Journal of Scheduling **7** (2004), no. 1, 85–91.

[DCL+18] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau, *Learning heuristics for the tsp by policy gradient*, International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer, 2018, pp. 170–181.

[DCTBG98] Federico Della Croce, R Tadei, P Baracco, and Andrea Grosso, *A new decomposition approach for the single machine total tardiness scheduling problem*, Journal of the Operational Research Society **49** (1998), no. 10, 1101–1106.

[DDC99] Marco Dorigo and Gianni Di Caro, *Ant colony optimization: a new meta-heuristic*, Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406), vol. 2, IEEE, 1999, pp. 1470–1477.

[DL90] Jianzhong Du and Joseph Y. T. Leung, *Minimizing total tardiness on one machine is NP-Hard*, Math. Oper. Res. **15** (1990), no. 3, 483–495.

[DZ99] Christos Dimopoulos and AMS Zalzala, *A genetic programming heuristic for the one-machine total tardiness problem*, Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), vol. 3, IEEE, 1999, pp. 2207–2214.

[Emm69] Hamilton Emmons, *One-machine sequencing to minimize certain functions of job tardiness*, Operations Research **17** (1969), no. 4, 701–715.

[GSDCT18] Michele Garraffa, Lei Shang, Federico Della Croce, and Vincent T'kindt, *An exact exponential branch-and-merge algorithm for the single machine total tardiness problem*, Theoretical Computer Science **745** (2018), 133–149.

[HG95] William D Harvey and Matthew L Ginsberg, *Limited discrepancy search*, IJCAI (1), 1995, pp. 607–615.

[HR92] J Edward Holsenback and Randolph M Russell, *A heuristic algorithm for sequencing on one machine to minimize total tardiness*, Journal of the Operational Research Society **43** (1992), no. 1, 53–62.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber, *Long short-term memory*, Neural Comput. **9** (1997), no. 8, 1735–1780.

[JM98] A. S. Jain and S. Meeran, *Job-shop scheduling using neural networks*, International Journal of Production Research **36** (1998), no. 5, 1249–1272.

[KDZ+17] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song, *Learning combinatorial optimization algorithms over graphs*, Advances in Neural Information Processing Systems, 2017, pp. 6348–6358.

[KGV83] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi, *Optimization by simulated annealing*, science **220** (1983), no. 4598, 671–680.

[Kou10] Christos Koulamas, *The single-machine total tardiness scheduling problem: Review and extensions*, European Journal of Operational Research **202** (2010), no. 1, 1 – 7.

[KW18] WWM Kool and M Welling, *Attention solves your TSP*, arXiv preprint arXiv:1803.08475 (2018).

[Law77] Eugene L. Lawler, *A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness*, Studies in Integer Programming (P.L. Hammer, E.L. Johnson, B.H. Korte, and G.L. Nemhauser, eds.), vol. 1, Annals of Discrete Mathematics, no. Supplement C, Elsevier, 1977, pp. 331 – 342.

[LBQ12] Jingpeng Li, Edmund K. Burke, and Rong Qu, *A pattern recognition based intelligent search method and two assignment problem case studies*, Applied Intelligence **36** (2012), no. 2, 442–453.

[MRG+17] Anton Milan, Seyed Hamid Rezatofighi, Ravi Garg, Anthony R Dick, and Ian D Reid, *Data-driven approximations to NP-hard problems.*, AAAI, 2017, pp. 1453–1459.

[OSL17] Junhyuk Oh, Satinder Singh, and Honglak Lee, *Value prediction network*, Advances in Neural Information Processing Systems, 2017, pp. 6118–6128.

[PSK93] SS Panwalkar, ML Smith, and CP Koulamas, *A heuristic for the single machine tardiness problem*, European Journal of Operational Research **70** (1993), no. 3, 304–310.

[PVW91] CN Potts and Luk N Van Wassenhove, *Single machine tardiness sequencing heuristics*, IIE transactions **23** (1991), no. 4, 346–354.

[PW82] C.N Potts and L.N Van Wassenhove, *A decomposition algorithm for the single machine total tardiness problem*, Operations Research Letters **1** (1982), no. 5, 177 – 181.

[RH97] RM Russell and JE Holsenback, *Evaluation of greedy, myopic and less-greedy heuristics for the single machine, total tardiness problem*, Journal of the Operational Research Society **48** (1997), no. 6, 640–646.

[Sch15] Jürgen Schmidhuber, *Deep learning in neural networks: An overview*, Neural Networks **61** (2015), no. Supplement C, 85 – 117.

[SDCG99] Wlodzimierz Szwarc, Federico Della Croce, and Andrea Grosso, *Solution of the single machine total tardiness problem*, Journal of Scheduling **2** (1999), no. 2, 55–71.

[SM96] Wlodzimierz Szwarc and Samar K Mukhopadhyay, *Decomposition of the single machine total tardiness problem*, Operations Research Letters **19** (1996), no. 5, 243–250.

[SYA+12] Gürsel A Süer, Xiaozhe Yang, Omar I Alhawari, Joel Santos, and Ramon Vazquez, *A genetic algorithm approach for minimizing total tardiness in single machine scheduling*, International Journal of Industrial Engineering and Management (IJIEM) **3** (2012), no. 3, 163–171.

[VFJ15] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly, *Pointer networks*, Advances in Neural Information Processing Systems, 2015, pp. 2692–2700.

[VNŠH18] Roman Václavík, Antonin Novak, Přemysl Šůcha, and Zdeněk Hanzálek, *Accelerating the branch-and-price algorithm using machine learning*, European Journal of Operational Research **271** (2018), no. 3, 1055 – 1069.

[VŠH16] Roman Václavík, Přemysl Šůcha, and Zdeněk Hanzálek, *Roster evaluation based on classifiers for the nurse rostering problem*, Journal of Heuristics **22** (2016), no. 5, 667–697.

[ZCBO91] D. N. Zhou, V. Cherkassky, T. R. Baldwin, and D. E. Olson, *A neural network approach to job-shop scheduling*, IEEE Transactions on Neural Networks **2** (1991), no. 1, 175–179.

Chapter 7

Attachements

Data_driven_algorithm_for_single_machine_scheduling_problem.pdf	This document.
src	source codes for master thesis.