Predicting the Optimal Period for Cyclic Hoist Scheduling Problems

Nikolaos Efthymiou and Neil Yorke-Smith(✉)

STAR Lab, Delft University of Technology, Delft, The Netherlands
nikolaos.efthymiou@epfl.ch, n.yorke-smith@tudelft.nl

Abstract. Since combinatorial scheduling problems are usually NP-hard, this paper investigates whether machine learning (ML) can accelerate exact solving of a problem instance. We adopt supervised learning on a corpus of problem instances, to acquire a function that predicts the optimal makespan for a given instance. The learned predictor is invariant to the instance size as it uses statistics of instance attributes. We provide this prediction to a solving algorithm in the form of bounds on the objective function. Specifically, this approach is applied to the well-studied Cyclic Hoist Scheduling Problem (CHSP). The goal for a CHSP instance is to find a feasible schedule for a hoist which moves objects between tanks with minimal cyclic period. Taking an existing Constraint Programming (CP) model for this problem, and an exact CP-SAT solver, we implement a Deep Neural Network, a Random Forest and a Gradient Boosting Tree in order to predict the optimal period p. Experimental results find that, first, ML models (in particular DNNs), can be good predictors of the optimal p; and, second, providing tight bounds for p around the predicted value to an exact solver significantly reduces the solving time without compromising the optimality of the solutions.

Keywords: combinatorial optimisation · supervised learning · cyclic hoist scheduling · constraint programming

1 Introduction

Computationally-challenging scheduling problems are common in industrial practice [8]. The hardness often arises from a combinatorial core in the problem, coupled by large size. Examples of such problems are satellite downlink scheduling [7], staff rostering [19], and hoist scheduling [11].

Contemporary machine learning (ML) methods are being exploited in the solving of large-scale combinatorial optimisation problems, including challenging scheduling problems [9]. Among the various approaches in the recent literature are learning problem-class-specific heuristics, end-to-end production of solutions, and warm-starting an optimisation solver. Bengio et al. [1], Kotary et al. [10] provide surveys. This paper explores the effectiveness of a loose coupling of the learning and optimisation. Thus we adopt the last of the above approaches:

A. A. Cire (Ed.): CPAIOR 2023, LNCS 13884, pp. 238–253, 2023.
https://doi.org/10.1007/978-3-031-33271-5_16

warm-starting a solver using information provided for a given problem instance by a pre-trained ML model. Specifically, taking inspiration from Wang et al. [24], we acquire and leverage an oracle for the makespan of the scheduling problem.

In more detail, we perform supervised learning on a corpus of problem instances to acquire a function that predicts the optimal makespan for a given instance. We provide this prediction to a solving algorithm in the form of bounds on the objective function. In this way we study the effect on the solver's computation time and the solution quality.

In the current paper this approach is developed for the *Cyclic Hoist Scheduling Problem* (CHSP), an optimization problem of practical and theoretical importance [11,12]. The aim is to find a schedule for one or multiple industrial hoists on track(s) that move objects between tanks. Process constraints impose bounds for the processing time in each tank, while the time that a hoist needs to travel between different tanks depends on the tanks and whether the hoist is empty or loaded. An important characteristic of CHSP is that the fixed series of moves is repeatedly performed by the hoist(s). This repetitive – cyclic – hoist schedule introduces the notion of the *cycle period* p, which is defined as the difference between the start time of two consecutive objects. Note that minimising the period is the analogue of minimising the makespan in this cyclic setting.

The literature boasts a host of techniques for solving CHSP instances [6], including custom branch-and-bound, mixed integer linear programming (MIP), constraint programming (CP), and meta-heuristics and evolutionary algorithms. We study the idea of providing predicted bounds to a competitive CP model of the CHSP [23], using an exact CP-SAT solver as the backend. Results show that supervised learning can acquire a function that predicts the optimal CHSP period p, and that providing bounds of 10% of this predicted p value leads to a mean time decrease of 90% in finding a feasible solution on unseen industrial problem instances, and a mean time decrease in solving to optimality of 44%.

This accelerated solving is important for industrial practice, where hoist lines can be much longer than tackled in the bulk of the academic literature [21]. Further, when an event occurs that causes a deviation from the planned schedule, a rapid rescheduling is necessary. In addition, from an academic perspective, the CHSP is relatively simple in its essential form [23] – while remaining challenging – which suggests that the same kind of methodology used in this paper can benefit other combinatorial scheduling problems.

Summarising: 1) we show that supervised learning can effectively acquire a function that predicts the optimal CHSP makespan; 2) we provide evidence that feeding a solver with predicted bounds of the objective function can accelerate the solving process; and 3) through extensive computational experiments we provide the first demonstration of the value of ML in finding CHSP solutions.

The remainder of the paper is structured as follows. Section 2 introduces the CHSP. Section 3 explains our approach. Section 4 studies the approach empirically. Section 5 situates our work in the literature. Section 6 concludes with future directions.

2 Hoist Scheduling Problem

The hoist scheduling problem is to operate one or multiple hoists which move along a linear track above a set of tanks (Fig. 1). Among the many problem variants [2], the cyclic hoist scheduling problem assumes a fixed sequence of items to be processed. From a scheduling perspective, the challenge is to allow the processing of successive items to overlap, so that (different) tasks on different items may be carried out at the same time. The schedule of tasks for one item is repeated for subsequent items: the length of a cycle is the time between the start of processing for an item and that of its successor. The objective is to maximise throughput, i.e., to minimise the cycle period, p; this is equivalent to minimising the makespan. An efficient CP model for the generic CHSP problem is formulated by Wallace and Yorke-Smith [23]. At the centre is a three-variable disjunctive constraint, from which arises the hard combinatorial core of the CHSP. The CP approach is interesting because a single model can solve a set of CHSP problem variants, whereas solving is performed by any of a range of state-of-the-art backend solvers which can ingest the model.

Optimal p Predictor. The CP model uses static calculated lower and upper bounds of p, denoted B_{calc}, to specify the space of feasible solutions. Given that such computation reflects the theoretical maximum range of p, B_{calc} tend to be quite loose. This leads to the central hypothesis of the paper: predicting the optimal value of p – without solving the CSHP instance – and then restricting the range in which the solver is trying to find a solution could result in lower solving times (T). This is a form of predict-then-optimise in which the prediction is not *necessary* for the optimisation, but can serve as a catalyst in the solver's inference and search process for an optimal solution.

We would like the learned predictor to be invariant to the instance size. For a CHSP instance with n tanks, we have the following exhaustive set of possible raw features: number of hoists, number of tanks, minimum/maximum processing times of each tank, $(n+1)$-dimensional vector of loaded move times, $(n+1) \times (n+1)$ matrix with empty move times, and capacity of each tank. Considering all these features leads to a dimensionality of $(n+1)^2+3n+4$. This is a large number of features, and also of varying dimension depending on the

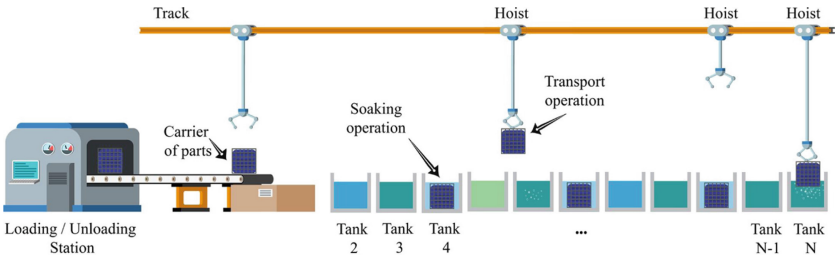

Fig. 1. Typical hoist scheduling line (from Laajili et al. [11]).

number of tanks. One approach would be to train a different ML model for each value of n [24]. Instead, we will study the possibility of having a fixed number of independent variables for the ML models, irrespective of the amount of tanks. To this end we will replace instances' attributes per tank with their descriptive statistics.

Hypothesis 1. *A universal regressor for the optimal value of p can be implemented that uses a fixed number of selected CHSP descriptive statistics.*

3 Methodology

In order to study accelerated solving of CHSP instances, we first perform supervised learning on a corpus of instances, to acquire a predictor of the optimal period for a given instance. We provide this prediction to a CP-SAT solver in the form of bounds on the objective function. This section explains the approach.

3.1 Data

Industry Instances. Seven sets of industry instances (PU [18], BO1, BO2, Zinc, Copper [13], Ligne1 and Ligne2 [15]) were used to: 1) analyse the various patterns that the values of instances' features follow, in order to implement the random generator (described below); 2) test the ML models; and 3) assess the performance of the CP solver when predicted bounds B_{pred} are used. Each instance has the following attributes: number of tanks n; minimum processing time for each tank $tmin$; maximum processing time for each tank $tmax$; vector of loaded move times from tank i to tank $i+1$, f; and 2D matrix of empty move times from tank i to tank j, e. Further, in order to extend the original instances to larger sizes, we considered values in $\{1, 2, 3, 4, 5\}$ for the tank capacity and the number of hoists (compare the 'multiplier' factor of Wallace and Yorke-Smith [23]). This yields a total of $7 \cdot 5 \cdot 5 = 175$ instances. We will denote this set with I_{ind}. We note that the values of 4 and 5 for the capacity and the number of hoists are not present in random instances used to train ML models, and thus we test the generalizability of the ML models. For comparison reasons, a subset of I_{ind} for which the number of hoists and the capacity take values in $\{1, 2, 3\}$ is also used (we will refer to this subset as $I_{ind[3]}$).

Random Instances. Since the number of industrial instances is limited, we generated random CHSP instances for training and testing the ML models. We implemented a random generator by making assumptions about the loading-unloading stations, the relative position of the tanks, and the times defining each instance. To this aim, we examined the patterns of the industry instances with respect to the following features and then uniform random values were chosen from a fixed range of each parameter: 1) Time window of treatments (minimum time, variability in minimum time, maximum to minimum time ratio) 2) Empty move times from tank i to tank j and their variability 3) Loaded move times from tank

i to tank $i+1$ and their variability 4) Number of tanks: $\{3, 4, \ldots, 24\}$ 5) Number of hoists: $\{1, 2, 3\}$ and 6) Tank capacity: $\{1, 2, 3\}$. These features are the ones mostly used in the literature and especially in the CP model that we utilise as a baseline. We assume the following: 1) treatment i occurs in tank i; 2) the loading and the unloading stations are the same; 3) there is one track; 4) the time to load/unload a job into/from a tank is 0; and 5) all tanks of an instance have the same capacity. These assumptions are made in most published works that include single-track industry hoist scheduling data. Regarding Hypothesis 1, we flattened each instance to a 19-dimensional vector (corresponding to the features of the ML models) consisting of: 1) the number of tanks; 2) the number of hoists; 3) the capacity of the tanks; and 4) the minimum, maximum, average and standard deviation of e, f, $tmin$ and $tmax$.

Four generators were implemented, each of which corresponds to a different topology. In particular, the following four spatial arrangements of the tanks were examined: 1) *linear* topology, where tank n is the farthest from the loading/unloading station (similar to Ligne2 [15]); 2) *reversed linear* topology with tank 1 being the farthest from the loading/unloading station (similar to BO1, BO2, Ligne1 [13,15]); 3) *ring* topology with tank $\frac{n}{2}$ being the farthest from the loading/unloading station (similar to Copper, Zinc, PU [13,18]); and 4) a transformed version of the linear topology with increased time for the loaded move from tank n to the loading/unloading station. In total we generated 166,320 random instances: 66,528 for the linear topology and 33,264 for each of the others. The observed variety in the descriptive statistics of the instances' e, f, $tmin$ and $tmax$ suggests the dataset captures many possible real life instances.

Solving Instances with Calculated Bounds. Of the generated instances, 98.6% (164,032) were solved with at least a feasible solution, within 6 min using the Google OR-Tools CP-SAT solver [17] ($\{$*processes* $= 8$, *free search* $=$ True, *optimisation level* $= 1$, *timeout* $= 360$ s$\}$). We denote this set with I_{gen}; it was used to train and test ML models. For hypothesis testing purposes, a random sample of 4,000 instances was drawn from I_{gen} (1,000 for each topology). In addition, as we consider to be 'difficult' those instances that led to a Satisfied solution when B_{calc} were used, and in order to make our sample more demanding, we include the remaining 173 such instances to the sample. We denote this set of 4,173 instances with I_{sample}.

3.2 ML Model Training

We experimented with three ML models. The (optimal) cycle period p_{calc} found by using B_{calc} was used as a target value of the ML models to be trained. We split the data set I_{gen} into a train set I_{train} and a test set I_{test} (test size $= 0.33$).

First, we used the Keras library for training and testing a Deep Neural Network (DNN); we experimented with 3–8 hidden, non-linear, dense layers with various activation functions (ReLu, Leaky ReLu, PReLu, ELU). We normalized the input features and used a linear dense single-output layer for predicting the optimal p. Model compilation was performed with the MAPE loss function

and the Adam optimizer (learning rate = 0.001). Fine-tuning was done with a validation split of 0.2. Six DNN variants were tested on I_{test}, I_{ind} and $I_{ind[3]}$.

Second, we also trained and tested a *Random Forest* (RF) regressor model using the Scikit-learn Python library. During the fine-tuning phase we tuned various values of the parameters *max depth, max features, min samples leaf, min samples split and n-estimators*. Third, we trained and tested a *Histogram-based Gradient Boosting Regression* (HGBR) model using Scikit-learn by tuning the following parameters: *l2 regularization*, *learning rate*, *loss*, *max iterations*, *max leaf nodes* and *min samples leaf*.

Lastly, to obtain a simpler and easier to interpret model, we performed feature extraction (by removing irrelevant features) on the flattened instances. For this we considered the *mean decrease in impurity* and the *permutation feature importance* [3], using the structure of the RF model. We give details below.

Solving Instances with Predicted Bounds. The best-performing ML model was applied to predict the optimal p value for the instances of I_{sample} and I_{ind}. This predicted value (p_{pred}) and its deviation from p_{calc} provide the basis for calculating the predicted bounds B_{pred}. We denote with $\hat{p}_l$ and $\hat{p}_u$ the lower and upper bounds derived accordingly. Note there is no guarantee that the true optimum is within these derived bounds. We denote with p_l and p_u the static lower and upper bounds from Wallace and Yorke-Smith [23], and write $B_{calc} = [p_l, p_u]$. These calculated bounds are conservative: they guarantee to contain the true optimum.

To solve problem instances from the sample dataset I_{sample} and the industrial dataset I_{ind}, we use the CP model unaltered, except that we give the following bounds on the objective function (note we always keep the tightest bounds):

$$p_l^* = \begin{cases} \hat{p}_l, & \text{if } \hat{p}_l \in (p_l, p_u) \\ p_l, & \text{otherwise} \end{cases} \qquad p_u^* = \begin{cases} \hat{p}_u, & \text{if } \hat{p}_u \in (p_l, p_u) \\ p_u, & \text{otherwise} \end{cases} \tag{1}$$

Given that in almost all cases the $\hat{p}_l, \hat{p}_u$ values were used, for simplicity we define the predicted bounds $B_{pred} = [p_l^*, p_u^*]$.

Throughout, the OR-Tools CP-SAT solver was used with the same machine configuration for experiments on each data set, to make the results comparable. We compare the solutions found by the solver when using B_{calc} and using B_{pred}, in terms of the following standard metrics: number of Optimal, Satisfied and Unsatisfiable cases; best found p value; number of solutions (N) found; and total solving time (T) to find and prove the optimal solution.

4 Experimental Results

With the pipeline described, this section reports an empirical study. Firstly in Sect. 4.1 we compare the various ML models to select the best performing model. Then in Sect. 4.2 we study the tightness of the predicted bounds versus their inclusion of the true optimal period. Thirdly in Sect. 4.3 we assess the performance of the whole pipeline in terms of solving time and optimal solution.

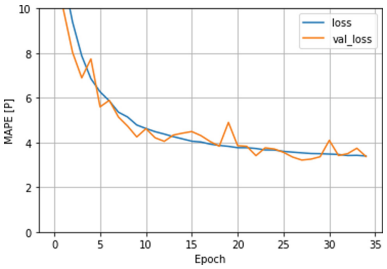

Fig. 2. DNN loss on training and validation sets

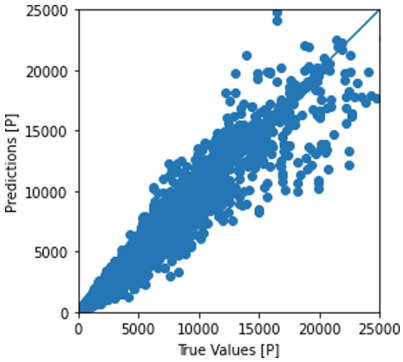

Fig. 3. Predicted vs. true values of p on dataset I_{test}

4.1 Experiment 1: ML Predictive Power and Model Selection

The DNN model that led to the best results (f^*) has 4 hidden dense layers with 92 neurons. The *Exponential Linear Units* activation function was used and the model was run for 35 epochs. Plots for the performance of this model are shown in Figs. 2 and 3. The graph of the training–validation loss reveals a rather normal learning progress over the number of epochs, and a good fit to the training data. The scatterplot shows a high correlation between predicted and true p values. The parameter values of the best RF model were the default values of the *Scikit-learn* library. As for the HGBR model, the strongest performance was obtained with the following parameter configuration: {*l2 regularization* = 0.22, *learning rate* = 0.065, *loss* = Poisson, *max iterations* = 1500, *max leaf nodes* = 100, *min samples leaf* = 160}.

Table 1 shows the seven most important features, as calculated by the RF regressor, with respect to two importance measures: *mean decrease in impurity* (MDI) and *feature permutation* (FP). These features were used to train and test a simpler DNN model f^- that receives 7-dimensional examples as input. Tables 2 and 3 report the MAPE values obtained with the best-performing models and the difference in performance between using all features and using only the important features. The best DNN model slightly outperforms the other ML methods used, on MAPE values on the test set I_{test}. The full model f^* achieved a MAPE of 3.38 on the test set I_{test}, while the value of 4.73 was reached using f^- on the same set. Further, the DNN model has significantly higher predictive power when testing on the industry sets I_{ind} and $I_{ind[3]}$. f^* performed adequately, even in the case of I_{ind} that contains instances with unseen attribute values. We note that the Ligne1 and PU industry sets have higher MAPE values compared to the other industry instances. One explanation might be that only these sets have tanks with infinite processing time (that is somehow treated by the CP solver) and there is no such case in the training dataset.

Table 1. Features sorted by importance

Feature	MDI	FP
$max(tmin)$	0.51	1.32
tank capacity	0.19	0.47
# hoists	0.09	0.31
$avg(f)$	0.05	0.23
# tanks	0.05	0.13
$max(f)$	0.04	0.07
$avg(tmax)$	0.02	0.04

Table 2. MAPE values (%) of f^* and f^- on the industry instances

	f^*		f^-	
	$I_{ind[3]}$	I_{ind}	$I_{ind[3]}$	I_{ind}
BO1	9	9	11	14
BO2	7	9	17	18
Copper	1	3	1	10
Ligne1	48	34	73	75
Ligne2	8	12	6	22
PU	13	19	22	23
Zinc	5	8	2	11
All	13	13	19	25

Table 3. MAPE values (%) of the ML models

Features	DNN			HGBR			Random Forest		
	$I_{ind[3]}$	I_{ind}	I_{test}	$I_{ind[3]}$	I_{ind}	I_{test}	$I_{ind[3]}$	I_{ind}	I_{test}
All	12.95	13.34	3.38	18.01	30.81	4.50	22.60	36.20	3.81
Important	18.85	24.74	4.73	27.61	37.18	4.89	26.61	37.16	3.98

Table 4. Cumulative relative frequency of instances per bound deviation d

d	f^*			f^-		
	$I_{ind[3]}$	I_{ind}	I_{test}	$I_{ind[3]}$	I_{ind}	I_{test}
5%	42.9	37.7	81.5	44.4	32.6	76.7
10%	66.7	61.7	90.2	57.1	42.3	86.7
15%	77.8	72.0	94.5	66.7	55.4	91.8
20%	84.13	77.7	96.6	71.4	60.6	94.7
>	100.0	100.0	100.0	100.0	100.0	100.0

Calculation of B_{pred} In order to minimise the possibility of p_{calc} being outside of the new predicted bounds of p, while keeping the bounds as narrow as possible, we calculated the cumulative relative frequency of instances per interval class of the *predicted to actual p deviation* ($d = |\frac{p_{pred} - p_{calc}}{p_{calc}}|\%$). Table 4 presents the percentage of instances with p_{pred} being at most $x\%$ away from p_{calc} (for $x \in \{5, 10, 15, 20\}$). The relative difference between p_{pred} and p_{calc} did not exceed 20% for 96.6% of random instances and 5% for 81.5% of instances. Thus we selected these *margins* for the further experiments. In case of I_{ind} these cumulative frequencies are lower and so we selected a margin of $\pm 10\%$ instead of $\pm 5\%$.

4.2 Experiment 2: Bounds and Solutions

Solutions on Random Instances. We examine first the results of the CP-SAT solver on the 4,173 instances of I_{sample}, when the predicted bounds B_{pred} (as predicted by f^*) were used. In most cases (98.3% for the ±5% margin and 90.2% for the ±20% margin), both the lower and upper predicted bounds of p (B_{pred}) are tighter than the calculated bounds (B_{calc}), and they were used by the solver. Further, in most cases (85.9% for the ±5% margin and 97.1% for the ±20% margin), B_{pred} contain the original p_{calc}. In the other 14.1% of cases, either the solver (incorrectly) found the instance unsatisfiable (e.g., $N = 421$ in case of ±5%), or the solver found a sub-optimal p (e.g., $N = 168$ in case of ±5%): usually only slightly sub-optimal.

A feasible solution was found for 89.9% of instances, and an optimal solution for 86.1% of instances, in the case of ±5% margin (respectively, 97.3% and 92.2%, in the case of ±20% margin). Hence only a relatively small number of instances have no solution ('unsatisfiable': there is no solution with p within B_{pred} or 'unknown'). As expected, this number reduces as the B_{pred} margin increases from 5% to 20%. The solver found the original p_{calc} in most cases: 85.0% and 96.1% of all instances or 94.6% and 98.8% of solved instances, per margin respectively. The percentage of satisfied cases is similar in all three scenarios: 5.4% for the original solver, 4.2% for the ±5% margin and 5.3% for the ±20% margin.

As the bound margin decreases, a very small number of instances have an optimal p greater (i.e., worse) than the original solver: $N = 147$ (plus $N = 18$ previously satisfied) for $B_{\text{pred\_5\%}}$ vs. B_{calc}, and $N = 9$ for $B_{\text{pred\_20\%}}$ vs. B_{calc}. This is so because the original optimal p is out of the predicted bounds used. Further, in case of satisfied solutions, there are only 13 and 20 cases respectively with p greater than the original solver and p_{calc} in B_{pred}. Table 5 reports in detail the solutions found with B_{pred} (in rows) in comparison with those found with B_{calc} (in columns) for each scenario.

Solutions on Industry Instances. Table 6 reports that, for most of the industry instances of I_{ind}, an (optimal) solution was found (163 out of 175, in case of ±10% margin and 168 in case of ±20%). However, especially in the case of ±10% margins, the B_{pred} of some instances do not contain the original p_{calc}. Thus the optimal p_{pred} found using B_{pred} is higher than p_{calc}. Table 6 presents the solutions per industry setting. We observe a variation in the performance of the solver that corresponds to the MAPE values presented in Table 2.

Table 5. Solutions for the set I_{sample}

	±5% *any p*		±5% p_{calc}		±20% *any p*		±20% p_{calc}	
	Opt	Sat	Opt	Sat	Opt	Sat	Opt	Sat
Optimal	3,546	46	3,399	23	3,834	14	3,825	14
Satisfied		159		126	1	213	1	172
Unknown		11						
Unsatisfiable	400	11			111			
Total	3,946	227	3,399	149	3,946	227	3,826	186

Table 6. Solutions per industry set

Industry instances	Unsatisfiable		p_{calc} found		$p \neq p_{calc}$ found	
	±10%	±20%	±10%	±20%	±10%	±20%
BO1	5		14	25	6	
BO2			17	24	8	1
Copper			25	25		
Ligne1	2	2	8	12	15	11
Ligne2	2	2	16	20	7	3
PU	3	3	10	16	12	6
Zinc			22	22	3	3
Total	12	7	112	144	51	24

Comparison with a Baseline Approach. We compare the above B_{pred}-based method with a simple bound estimation. First we compute the average r of $p_{calc}/\overline{B_{calc}}$ across all instances; then for each instance compute estimated bounds: $B_{est} = (lb_{est}, ub_{est}) = (r \cdot \overline{B_{calc}} \cdot 0.95, r \cdot \overline{B_{calc}} \cdot 1.05)$. B_{est} contain the original p_{calc} for 5% of the solved instances of I_{sample} and for 7% of the solved instances of I_{ind}. Hence if we apply these new estimated bounds to the CP solving process, for most instances either no solution will be found or a solution with a greater p.

4.3 Experiment 3: Solver Performance with Predicted Bounds

Our final experiment studies the impact of using B_{pred} in the CP-SAT solver. For this we used a subset of I_{sample} to only include cases for which an optimal solution was found, and we formulate three hypotheses:

Hypothesis 2. *A (CP) solver that uses B_{pred} instead of B_{calc} requires less time (T) to find an optimal solution.*

Hypothesis 3. *As B_{pred} become tighter, the solving time (T) decreases further.*

Table 7. Random Instances: Predicted vs. Calculated bounds

		Optimal solution				Satisfied			
		Inst.	Δp	ΔT	ΔN	Inst	Δp	ΔT	ΔN
Margin ±5%	Any p	3,546	0.3%	−70.7%	−56.6%	159	0.0%	−16.6%	−77.4%
	p_{calc}	3,399	—	−68.1%	−53.4%	126	—	−29.5%	−78.0%
Margin ±20%	Any p	3,834	0.1%	−33.1%	−27.1%	213	−0.1%	−9.5%	−36.3%
	p_{calc}	3,825	—	−33.0%	−26.9%	172	—	-22.9%	−40.2%

Table 8. Solving time difference (margin ±5%) – Negative vs. positive deviation

	T_{calc}	$T_{pred\_5\%}$	ΔT
$T_{pred\_5\%} - T_{calc} > 0$	0.18	0.22	0.04
$T_{pred\_5\%} - T_{calc} < 0$	2.31	0.66	−1.65
Total	1.91	0.58	−1.33

Hypothesis 4. *Using B_{pred} instead of B_{calc} does not increase the value of p of the optimal solution found.*

To test these hypotheses, we follow a repeated-measurements experimental design and thus only cases with a feasible solution in both conditions (calculated bounds vs. predicted bounds) are included. Note that cases with no solution are irrelevant here, for their failure relates to prediction error, i.e., Hypothesis 1. Comparisons are made independently for instances with an optimal solution in both conditions, and instances with a satisfied solution in both conditions.

Impact on Solving Time. Regarding Hypotheses 2 and 3, we can accept that the solving time is significantly lower when the predicted bounds of p (B_{pred}) are used, instead of the calculated bounds B_{calc} ($T_{pred\_5\%}$: $\overline{X} = 0.58$, $s = 4.93$; $T_{pred\_20\%}$: $\overline{X} = 1.27$, $s = 11.01$; T_{calc}: $\overline{X} = 1.91$, $s = 14.09$). Specifically, there is a decrease in time ($\Delta T = -1.3$) when $B_{pred\_5\%}$ is used instead of B_{calc}. There is a more modest decrease ($\Delta T = -0.6$) when $B_{pred\_20\%}$ is used instead of B_{calc}. Accordingly, there is a decrease in solving time when B_{pred} become narrower from ±20% to ±5% ($\Delta T = -0.7$).

As Table 7 shows, the decrease in the solving time for cases in which an optimal solution was found (−70.7%) is larger compared to cases with a satisfied solution (−16.6%). Figure 4 shows there was a decrease in the solving time for 81% of all solved instances. We also see that for a minority of some 60 instances, much more time was required (relative change higher than 90%). However, in most cases with $\Delta T > 0$, the solving time and its increase are very small in absolute terms. The corresponding means of ΔT and T are shown in Table 8.

Impact on p Value. Regarding Hypothesis 4, the impact of B_{pred} on the value of p of the optimal solution is negligible (p_{calc}: $\overline{X} = 3007$, $s = 3099$; $p_{pred\_5\%}$: $\overline{X} = 3017$, $s = 3117$; $p_{pred\_20\%}$: $\overline{X} = 3010$, $s = 3107$). We found a very small

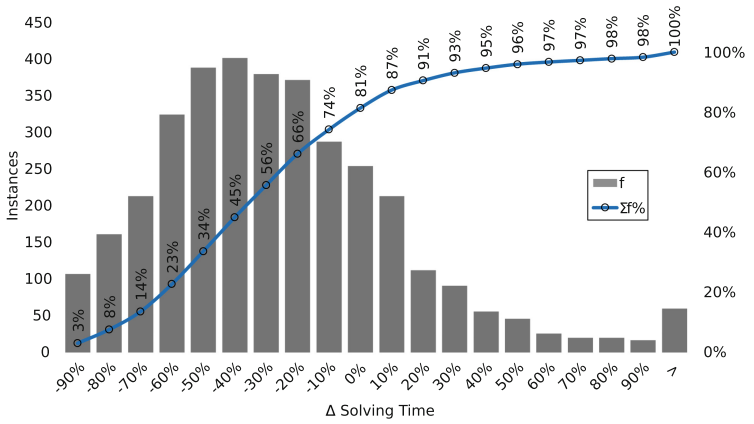

Fig. 4. Solving time difference with 5% margin on I_{sample} (optimal solution found)

Table 9. Industry Instances: Predicted vs. Calculated bounds

		Inst.	ΔP	ΔT	ΔN
Margin ±10%	Any p	163	1.2%	−90.3%	−32.6%
	p_{calc}	112	—	−43.5%	−18.8%
Margin ±20%	Any p	168	0.5%	−78.8%	−3.7%
	p_{calc}	144	—	−33.2%	2.0%

increase ($\Delta p = 9.4$) between $p_{pred\_5\%}$ and p_{calc}, an even smaller increase ($\Delta p = 2.2$) between $p_{pred\_20\%}$ and p_{calc}, and an analogous increase ($\Delta p = 7.2$) between $p_{pred\_5\%}$ and $p_{pred\_20\%}$. Table 7 reports the relative change in p.

Results on Industry Instances. Similar promising results were observed on the industry data set I_{ind} (Table 9). Using B_{pred} resulted in a large decrease in solving time with practically no increase in the value of p. Further, Fig. 5 shows that in 85% of all solved instances, solving time was reduced.

5 Related Work

The interplay between learning, search and inference for solving combinatorial optimisation problems was surveyed by Bengio et al. [1]. In CP, Cappart et al. [4] tightly hybridise reinforcement learning with a dynamic programming representation. Most works tend to emphasise one side or the other.

On the one hand, some authors emphasise the ML side followed by some search. In early work, Deudon et al. [5] for instance study the travelling salesperson problem, and improve a learned schedule with local search. In more recent work, Kool et al. [9] study vehicle routing problems with time windows and derive a schedule using neural-network guided dynamic programming.

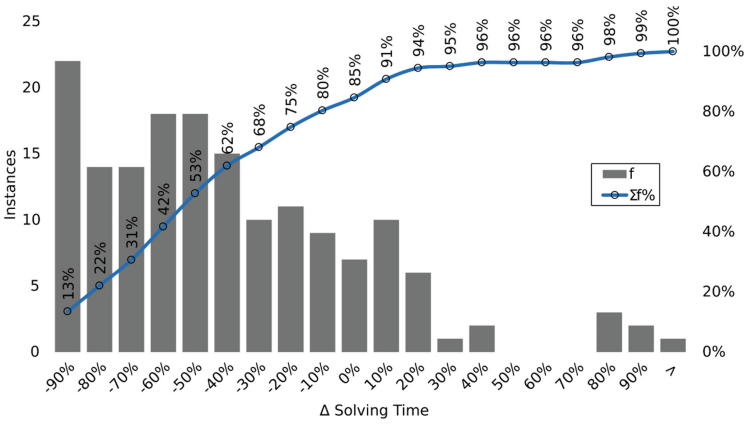

Fig. 5. Solving time difference with 10% margin on I_{ind} (all solved instances)

On the other hand – and closer to this paper – is the idea of pre-computing using ML and informing a 'standard' optimisation solver with this information. Osanlou et al. [16], for instance, study constrained path planning and use a graph convolution network to predict an optimal vertex order. They feed the resulting cost into a branch-and-bound search as an upper bound. Xu et al. [25] predict with supervised learning the satisfaction of a general (binary) CSP. Wang et al. [24] study job shop scheduling, and predict the optimal makespan of an instance. We follow the same approach, but on a more complex problem; we use aggregation functions in order to obtain a regressor independent of the problem size (number of tanks); we further simplify the ML models by performing feature extraction; and we use prediction bounding rather than the binary objective search. There are many recent similar works (Václavík et al. [26], Zhang et al. [22]); none to our knowledge consider the hoist scheduling problem.

Lastly we note the approach of decision-focussed learning (e.g., [14]), which offers a tighter integration between learning and optimisation than either the learning-to-search or the predict-then-optimise approaches.

6 Conclusion and Future Work

The growing research field at the intersection of machine learning and combinatorial optimisation leads in this paper to the question: how can a predict-then-optimise approach lead to faster solving of hard scheduling problems, without sacrificing solution quality?

This approach – in the form of offline supervised learning coupled with online exact solving – was applied to the CHSP, a hard combinatorial optimisation problem that involves search for a feasible hoist schedule that minimises the cycle period p. We tested various ML models for predicting the optimal p, in order to improve the effectiveness of a CP-SAT solver by providing tighter objective function bounds. Using a size-invariant representation, coupled with feature ablation,

we found that a relatively simple dense neural network was effective in acquiring a predictor. Experiments on synthetic and industrial benchmarks showed that using tighter bounds derived from $\hat{p}$ leads to markedly lower solving times (mean 44% decrease on industrial instances), without increasing the value of p in the majority of cases. Only for a small fraction of instances is the solving time using the predicted bounds longer. This minority increase most commonly happens when the initial solving time is negligible (i.e., easy instances): thus the delta is small in absolute terms.

Our approach has intentionally emphasised simplicity and a loose coupling of the learning and optimisation. The highly-promising results suggest several future directions. First, it will be interesting to choose the relaxed bounds around $\hat{p}$ based not on static percentages, but according to the confidence of the ML model. Second, to further study the generalisation ability of the predictor: for instance, testing on instances obtained from a different generator, or exploring other backend solvers such as a MIP. Third, since the ML models implemented in this paper do not consider some CSHP instance attributes (e.g., number of tracks and loading/unloading times), their inclusion could be investigated in future work. Fourth, the ML model could be informed by a regret-based loss function, in the style of decision-focussed learning [20]. Besides these directions, we are exploring whether a more end-to-end ML approach could be promising. To this aim we currently study the use of a graph neural network embedding to try and predict the optimal solution (not just the optimal objective function value). Initial experiments indicate that predicting the whole solution is challenging; it seems more promising to predict only the removal time of the first item from its first tank, and we find this already is useful guidance for the solver.

Acknowledgements. We thank the anonymous reviewers. Thanks to K. van den Houten, S. van der Laan, M. Wallace and M. de Weerdt. Partially supported by TAILOR, funded by the EU Horizon 2020 programme under grant 952215.

References

1. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: a methodological tour d'horizon. Eur. J. Oper. Res. **290**(2), 405–421 (2021). https://doi.org/10.1016/j.ejor.2020.07.063
2. Boysen, N., Briskorn, D., Meisel, F.: A generalized classification scheme for crane scheduling with interference. Eur. J. Oper. Res. **258**(1), 343–357 (2017). https://doi.org/10.1016/j.ejor.2016.08.041
3. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001). https://doi.org/10.1023/A:1010933404324
4. Cappart, Q., Moisan, T., Rousseau, L.M., Prémont-Schwarz, I., Cire, A.A.: Combining reinforcement learning and constraint programming for combinatorial optimization. In: AAAI 2021, pp. 3677–3687 (2021). https://doi.org/10.1609/aaai.v35i5.16484
5. Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., Rousseau, L.: Learning heuristics for the TSP by policy gradient. In: CPAIOR 2018, pp. 170–181 (2018). https://doi.org/10.1007/978-3-319-93031-2_12

6. Feng, J., Chu, C., Che, A.: Cyclic jobshop hoist scheduling with multi-capacity reentrant tanks and time-window constraints. Comput. Ind. Eng. **120**, 382–391 (2018). https://doi.org/10.1016/j.cie.2018.04.046
7. He, L., Guijt, A., de Weerdt, M., Xing, L., Yorke-Smith, N.: Order acceptance and scheduling with sequence-dependent setup times. Comput. Ind. Eng. **138**, 106102 (2019). https://doi.org/10.1016/j.cie.2019.106102
8. Kacem, I., Kellerer, H.: Foreword: combinatorial optimization for industrial engineering. Comput. Ind. Eng. **61**(2), 239–241 (2011). https://doi.org/10.1016/j.cie.2011.07.016
9. Kool, W., van Hoof, H., Gromicho, J.A.S., Welling, M.: Deep policy dynamic programming for vehicle routing problems. In: CPAIOR 2022, pp. 190–213 (2022). https://doi.org/10.1007/978-3-031-08011-1_14
10. Kotary, J., Fioretto, F., Hentenryck, P.V., Wilder, B.: End-to-end constrained optimization learning: a survey. In: IJCAI 2021, pp. 4475–4482 (2021). https://doi.org/10.24963/ijcai.2021/610
11. Laajili, E., Lamrous, S., Manier, M., Nicod, J.: An adapted variable neighborhood search based algorithm for the cyclic multi-hoist design and scheduling problem. Comput. Ind. Eng. **157**, 107225 (2021). https://doi.org/10.1016/j.cie.2021.107225
12. Lee, C.Y., Lei, L., Pinedo, M.: Current trends in deterministic scheduling. Ann. Oper. Res. **70**, 1–41 (1997). https://doi.org/10.1023/A:1018909801944
13. Leung, J.M., Zhang, G., Yang, X., Mak, R., Lam, K.: Optimal cyclic multi-hoist scheduling: a mixed integer programming approach. Oper. Res. **52**(6), 965–976 (2004). https://doi.org/10.1287/opre.1040.0144
14. Mandi, J., Bucarey, V., Mulamba Ke Tchomba, M., Guns, T.: Decision-focused learning: through the lens of learning to rank. In: ICML 2022, pp. 14935–14947 (2022). https://proceedings.mlr.press/v162/mandi22a.html
15. Manier, M.A., Lamrous, S.: An evolutionary approach for the design and scheduling of electroplating facilities. J. Math. Model. Algorithms **7**(2), 197–215 (2008). https://doi.org/10.1007/s10852-008-9083-z
16. Osanlou, K., Bursuc, A., Guettier, C., Cazenave, T., Jacopin, E.: Optimal solving of constrained path-planning problems with graph convolutional networks and optimized tree search. In: IROS 2019, pp. 3519–3525 (2019). https://doi.org/10.1109/IROS40897.2019.8968113
17. Perron, L., Furnon, V.: OR-Tools version 9.3 (2022). https://developers.google.com/optimization/
18. Phillips, L.W., Unger, P.S.: Mathematical programming solution of a hoist scheduling program. AIIE Trans. **8**(2), 219–225 (1976). https://doi.org/10.1080/05695557608975070
19. Quesnel, F., Wu, A., Desaulniers, G., Soumis, F.: Deep-learning-based partial pricing in a branch-and-price algorithm for personalized crew rostering. Comput. Oper. Res. **138**, 105554 (2022). https://doi.org/10.1016/j.cor.2021.105554
20. Teso, S., et al.: Machine learning for combinatorial optimisation of partially-specified problems. CoRR abs/2205.10157 (2022). https://doi.org/10.48550/arXiv.2205.10157
21. UTIKAL Automation: Private correspondence (2022). https://utikal-automation.com
22. Václavík, R., Novák, A., Sucha, P., Hanzálek, Z.: Accelerating the branch-and-price algorithm using machine learning. Eur. J. Oper. Res. **271**(3), 1055–1069 (2018). https://doi.org/10.1016/j.ejor.2018.05.046

23. Wallace, M., Yorke-Smith, N.: A new constraint programming model and solving for the cyclic hoist scheduling problem. Constraints **25**(3), 319–337 (2020). https://doi.org/10.1007/s10601-020-09316-z
24. Wang, T., Payberah, A.H., Vlassov, V.: CONVJSSP: convolutional learning for job-shop scheduling problems. In: ICMLA 2022, pp. 1483–1490 (2020). https://doi.org/10.1109/ICMLA51294.2020.00229
25. Xu, H., Koenig, S., Kumar, T.K.S.: Towards effective deep learning for constraint satisfaction problems. In: CP 2018, pp. 588–597 (2018). https://doi.org/10.1007/978-3-319-98334-9_38
26. Zhang, W., et al.: NLocalSAT: boosting local search with solution prediction. In: IJCAI 2020, pp. 1177–1183 (2020). https://doi.org/10.24963/ijcai.2020/164