Data-based Approach to Predict Feasibility and Computational Requirement for Chemical Production Scheduling

Boeun Kim* Christos T. Maravelias**

**Andlinger Center for Energy and the Environment, Princeton University, Princeton, NJ 08544, USA (e-mail: bk3460@princeton.edu).*
*** Andlinger Center for Energy and the Environment, Princeton University, Princeton, NJ 08544, USA (e-mail: maravelias@princeton.edu).*
*** Department of Chemical and Biological Engineering, Princeton University, Princeton, NJ 08544, USA*

Abstract: Online scheduling requires frequent re-optimization to generate a schedule repeatedly accounting for updated information. However, if the time between re-optimizations is too short, then finding good, and in some cases even feasible, solutions can become challenging. This work proposed an approach, based on supervised learning techniques, to predict whether a given instance is feasible and, given that it is feasible, what is the computational requirement to solve the instance. Towards this goal, we introduce various types of features related to problem size, scheduling horizon, and processing times and costs that can be derived based on domain knowledge. Logistic regression and random forests models are trained as feasibility classifier and computational time regressor, respectively, using the dataset obtained from a wide variety of instances. Both show good predictive performances: F1 score ~0.90 and AUC ~0.98 for the feasibility classification and MSE ~0.5 for the computational time prediction. Finally, we discuss the features that are shown to be significant in the cases of makespan minimization and cost minimization.

Keywords: Batch plant scheduling, mixed integer programming, supervised learning, feasibility classification, algorithm performance prediction

1. INTRODUCTION

Online scheduling, in which a schedule is re-optimized in real-time with updated information, has received considerable attention in recent years (Gupta et al., 2016; Gupta and Maravelias, 2019) due to the increasing demand for smart manufacturing. These studies have revealed that frequent re-optimization is key in achieving good closed-loop performance. Chemical production scheduling problem is often formulated as a mixed-integer programming (MIP) model, however, state-of-the-art algorithms for such combinatorial problems often exhibit high variation in computational time across instances even if their problem size are the same (Hitter et al., 2014). Moreover, some of these instances maybe infeasible if no model modifications are made (e.g., introduction of lost sales or lateness). Therefore, there is a need for developing methods that would allow us to predict, fast, whether an instance will be feasible and the computational requirements needed for its solution.

Although instance characteristics (e.g., horizon length, batch-unit ratio, load, etc.) can make a problem instance infeasible or hard to solve within a given time limit, our understanding of what problem and instance features make a MIP scheduling model computationally expensive is very limited. In the computer science community, empirical hardness refers to the required time to solve a given problem instance through a given algorithm. Several studies have developed empirical hardness models for combinatorial optimization problems such as propositional satisfiability (SAT), constraint programming (CSP), traveling salesperson (TSP), and combinatorial auction winner determination (Smith-Miles and Lopes, 2012; Hutter et al., 2014; Leyton-Brown et al., 2009; Leyton-Brown et al., 2014). Various supervised learning methods (e.g., Ridge regression, Gaussian process regression, regression trees, neural networks) have been used to predict how long an algorithm will take to solve an instance. Recently, (Muñoz and Capón-García, 2019) proposed a multi-label classification strategy for selecting a suitable scheduling model and solution approach to solve a certain production scheduling problem according to its objective function, decisions, and production and resource constraints. However, there are no methods establishing relationships between instance characteristics and model performance.

Accordingly, we use supervised machine learning (ML) techniques to predict the feasibility and computational requirement for chemical production scheduling in this work. In particular, we study short-term scheduling in single-stage multiple-unit environment and consider two objective functions, i.e., makespan minimization and cost minimization. We use a large number of instances having different characteristics (e.g., number of batches and units, processing times and costs, horizon length, etc.). Then, using a specific model, we gather computational cost and solution quality for each instance. To obtain generalized supervised learning models, we introduce various problem-specific instance features, which we then utilize to build a logistic regression model that categorizes the feasibility of an instance, and a

random forest model to predict the CPU time required by the solver to solve a feasible instance.

2. BACKGROUND

2.1 Problem Statement

We study the short-term batch plant scheduling in the single-stage with parallel-unit environment. Given production recipe (e.g., processing time and cost) and batching decisions (e.g., batch number and dates), we aim to find optimal assignment and sequencing/timing of given batches to units in terms of two objective functions: makespan minimization and cost minimization. We also make the following assumptions: (i) all batches can be carried out in any unit, (ii) preemption is not allowed, (iii) all processing parameters are integers, (iv) processing times/costs and release/due times are deterministic, and (v) changeover between batches, utility, and other shared resources are not considered.

We introduce the following indices, sets, and parameters to describe the scheduling model and instance. We define a set of batches $i \in \mathbf{I}$ and a set of units $j \in \mathbf{J}$. The time and cost of batch i processed in unit j are denoted by τ_{ij} and γ_{ij}, respectively. Release/due time for batch i are represented by ρ_i/ε_i and the scheduling horizon of an instance is given by η. For the supervised learning models, we use $x_m = [x_1, .., x_n]^T$ and y_m to denote the input vector including n features and output of instance sample $m \in \mathbf{M}$, respectively.

2.2 Discrete-Time MIP Scheduling Model

We adopt a discrete-time representation, in which η is uniformly discretized with a period width of δ, and use index $t \in \mathbf{T} = \{0, 1, 2, \dots, |\mathbf{T}|\}$ to denote the time points and periods. δ is typically chosen as the greatest common factor of all time-related parameters. The converted parameters in terms of the discrete-time grid are obtained by dividing their original values by δ. In this study, all the processing parameters are integers and $\delta = 1$, so we use the original notations of time-related parameters.

We use the discrete-time MIP scheduling model (Maravelias, C.T., 2021) that consists of batch-unit assignment (1), unit utilization (2), and release/due times restriction (3).

$$\sum_{j\in\mathbf{J}}\sum_{t\in\mathbf{T}} X_{ijt} = 1 \quad \forall i \in \mathbf{I} \tag{1}$$

$$\sum_{i\in\mathbf{I}}\sum_{t'=t-\tau_{ij}+1}^{t'=t} X_{ijt'} \le 1 \quad \forall j \in \mathbf{J}, t \in \mathbf{T} \tag{2}$$

$$X_{ijt} = 0 \quad \forall i \in \mathbf{I}, j \in \mathbf{J}, t < \rho_i, t > \varepsilon_i - \tau_{ij} \tag{3}$$

where binary variable X_{ijt} implies batch i starts in unit j at time point t when $X_{ijt} = 1$. We enforce that each batch is processed exactly once by (1), and the unit can process at most one batch at any time point by (2).

We consider two objective functions: makespan minimization (4) and cost minimization (5).

$$\min MS : MS \ge \sum_{j\in\mathbf{J}}\sum_{t\in\mathbf{T}} \delta(t + \tau_{ij}) X_{ijt} \quad \forall i \in \mathbf{I} \tag{4}$$

$$\min \sum_{i\in\mathbf{I}}\sum_{j\in\mathbf{J}}\left(\gamma_{ij}\sum_{t\in\mathbf{T}} X_{ijt}\right) \tag{5}$$

where $MS \in \mathbb{R}_+$ denotes the makespan which is the time required to finish all given batches. We ensure that the MS is greater than or equal to the finish times of every batch.

2.3 Classification Model

The general aim of supervised ML algorithms is to develop an empirical model from data to predict a given instance's qualitative output (in classification) or quantitative output (in regression). To build the feasibility classification model, we employ logistic regression, one of the widely used statistical methods for the binary classification due to its simplicity and probabilistic interpretability (Komarek, P. 2004). If a given instance has any feasible integer solution, it belongs to the class labeled by 0 (i.e., y_m =0). Otherwise, one belongs to the class labeled by 1 (i.e., $y_m = 1$). We introduce z_m to denote the net input of an instance sample m, calculated as a linear weighted combination of features: $z_m = w^T x_m + w_0$ where w_0 represents the constant term and $w = [w_1, \dots, w_n]^T$ is the vector of the coefficients. The estimated coefficient (except for w_0) indicates the linear relationship between the log-odds and the corresponding feature. In the logistic regression, z_m for each instance passes through the logistic function, i.e., sigmoid function: $\sigma(z_m) = 1/(1 + e^{-z_m})$, that converts the calculated z_m (any real value) into certain constant between $[0, 1]$. The converted value corresponds to the conditional probability that the given instance belongs to the class 1. Note that $\sigma(z_m)$ becomes 1 as z_m increases, and $\sigma(z_m)$ tends towards 0 as z_m decreases. Finally, the threshold function such that

$$\begin{cases} 1 \text{ if } \sigma(z_m) \ge \epsilon \\ 0 \text{ otherwise} \end{cases} \tag{6}$$

decides the class to where a given instance belongs.

We use the penalized logistic regression that finds the optimal w minimizing the following negative log-likelihood with L_1 penalty term (Ravikumar et al., 2010):

$$\min_w \sum_{m\in\mathbf{M}} \{-y_m \log(\sigma(z_m)) - (1 - y_m)\log(1 - \sigma(z_m))\} + c\|w\|_1 \tag{7}$$

where c is the hyperparameter that controls the number of nonzero weights (feature selection) and helps in retaining

meaningful features and reducing overfitting. In this work, the logistic regression is solved by using "saga" solver in scikit-learn Python library (Pedregosa et al., 2011), and $\epsilon = 0.5$.

We can evaluate the binary classification performance by various performance metrics based on the confusion matrix (Aggarwal, C.C., 2014). Such metrics are accuracy, precision, recall, F1 score, and area under receiver operating christianistic curve (AUC). Table 1 shows the confusion matrix for the binary classification, whose elements refer to the number of samples classified according to the actual and predicted classes. All instances correctly classified (i.e., TP and TN) are located in the diagonal of the confusion matrix.

Accuracy, the fraction of the correct predictions, i.e., $(\mathrm{TP} + \mathrm{TN})/(\mathrm{TP} + \mathrm{FN} + \mathrm{FP} + \mathrm{TN})$, is the most intuitive performance measure but can be biased. Recall or precision is the ratio of the TP to the total actual positives $(\mathrm{TP} + \mathrm{FN})$ or to the total predicted positives $(\mathrm{TP} + \mathrm{FP})$, respectively. F1 score, one of the commonly used metrics, is defined as the harmonic mean of the precision and recall and thus accounts for both FP and FN errors. The higher F1 score, the better. The receiver operating christianistic (ROC) curve is generated by plotting the TP rate (i.e., recall) against the FP rate (i.e., $\mathrm{FP}/(\mathrm{TN} + \mathrm{FP})$). AUC is widely used to assess the classifier's ability to discriminate classes. A no skill classifier has an AUC score of 0.5, whereas a perfect one has an AUC score of 1.0. The higher the AUC, the better the model is at distinguishing between the classes.

Table 1. Confusion matrix for binary classification

	Actual class = 1	Actual class = 0
Predicted class = 1	True positive (TP)	False positive (FP)
Predicted class = 0	False negative (FN)	True negative (TN)

2.4 Regression Model

Random forests (RF) yielded the best performance in predicting the required computational costs of SAT, TSP, and MIP among the different supervised ML methods in the previous work (Hutter et al., 2014). Thus, in this work, we employ the RF algorithm to build the model that can predict the computational time required for solving a given scheduling MIP instance. RF, developed by (Breiman, L., 2001), is an ensemble of randomly trained decision trees. A decision tree is a collection of leaf nodes that partition the input space into disjoint regions, $R_1, \ldots, R_K$ and approximates the output in each region as a constant. Starting from the root node, each internal node in a decision tree splits samples into two branches according to a decision rule. For instance, if a sample whose feature selected (split feature) is larger than its split point goes to the right branch of an internal node. Otherwise, a sample goes to the left branch. Consequently, the prediction by the trained regression tree $\hat{y}_m$ is given by

$$\hat{y}_m(x_m) = \sum_{k=1}^{K} \bar{y}_k I_{x_m \in R_k} \tag{8}$$

where $\bar{y}_k$ represents the average output of training samples in R_kAnd I is the indicator function which takes 1 if given x_m belongs to region R_k and 0 otherwise (James et al., 2013). Training of the decision tree is to learn the decision rule of each node, minimizing the sum of squared errors between the actual and predicted outputs. The decision tree is capable of modeling complex interactions with small bias but is likely to have high variance. We can overcome this limitation by combining multiple regression trees into an ensemble. Each tree is trained on subsamples of the training data, and RF provides the final prediction by averaging the predictions across the trees (Breiman, L., 2001). Therefore, RF is less prone to overfitting. In this work, we use CART (Classification and Regression Trees) algorithm in scikit-learn Python library (Pedregosa et al., 2011) to establish the regression trees, and decide the number of trees and tree depth through 5-fold cross validation. The performance of the trained regression model is evaluated in terms of mean squared error (MSE) between the computational time predicted by the model and its true value.

3. PROBLEM INSTANCES AND DATA GENERATION

We account for a wide range of instance attributes (i.e., batch number $|\mathbf{I}|$, unit number $|\mathbf{J}|$, processing times τ_{ij}, processing costs γ_{ij}, and due times ρ_i) to build the reliable classifier and regressor. Their investigated ranges are summarized in Table 2. For each unit number, the maximum batch number tested has the ratio of the batch number to the unit number around (or above) 8. Other processing parameters are drawn from uniform distributions over respective considered ranges. More details are described in the subsequent subsections.

Table 2. Summary of instance attributes for data generation

	Range		Range
$\|\mathbf{J}\|$	{3, ..., 8}	$\|\mathbf{I}\|$	{10, …, 30} if $\|\mathbf{J}\|$= 3 or 4 {10, …, 40} if $\|\mathbf{J}\|$= 5 {10, …, 50} if $\|\mathbf{J}\|$= 6 {10, …, 55} if $\|\mathbf{J}\|$= 7 {10, …, 65} if $\|\mathbf{J}\|$= 8
τ_{ij}	{3, ..., 9}		
γ_{ij}	{10, ..., 16}		
ρ_i	Random		

We use the solver CPLEX 12.8.0 in GAMS 26.1.0 to solve the discrete-time MIP scheduling models with the generated problem instances. We set the absolute and relative optimality criteria as 0.999 and 0 for makespan minimization and cost minimization, respectively. The CPU time resource limit is 3 hours (i.e., 10,800 seconds). All optimizations are executed on a cluster of 24 Intel Xeon machines running CentosOS operating system.

We collect the qualitative and quantitative optimization results of each scheduling MIP instance. Model and solver statuses in GAMS solution report provide information on whether a given

instance has any feasible solution and whether an optimal solution of the feasible instance can be found within the given time limit. We utilize such information to label a problem instance by 1 or 0 for the feasibility classification. Also, we measure CPU time in seconds that the solver takes to solve a given instance for the computational time prediction.

3.1 Makespan Minimization

For given unit and batch numbers ($|\mathbf{J}|$ and $|\mathbf{I}|$), processing times τ_{ij} are sampled from a discrete uniform distribution reported in Table 2. To generate a list of candidates for η that will be tested, we introduce a base scheduling horizon η_{base} as follow:

$$\eta_{base} = \left\lceil \frac{\sum_{i \in \mathbf{I}} \tau_i^{AVG}}{|\mathbf{J}|} \right\rceil \tag{9}$$

where τ_i^{AVG} is the average processing time of batch i, defined as $\sum_{j \in \mathbf{J}} \tau_{ij} / |\mathbf{J}|$. To analyze the effect of η on the instance's feasibility and computational time, we calculate horizon lengths, i.e., $\{\lceil 0.7\eta_{base} \rceil, \lceil 0.75\eta_{base} \rceil, \dots, \lceil 1.3\eta_{base} \rceil\}$ and remove any duplicate values; all processing data except for η remain the same. In makespan minimization, we assume that the release and due times for all batches are 0 and η, respectively. We generate 2,000 sets of the processing data for each combination of unit and batch numbers and explore at most 13 horizon lengths for each set.

3.2 Cost Minimization

For cost minimization, processing times τ_{ij} and costs γ_{ij} are uniformly drawn for given unit and batch numbers as presented in Table 2. In addition to this, $\rho_i = 0$ for all batches whereas due times ε_i are sampled from a discrete uniform distribution, i.e., $[\lceil (0.9 - d)\eta_{base} \rceil, \lceil (0.9 + d)\eta_{base} \rceil]$ where the parameter d controls the variation of due times and $d =$ 0.2, 0.3 or 0.4. Then, η is given by the maximum value of ε_i. We generate 5,000 sets of processing time, cost, and due time data for each combination of unit and batch numbers.

4. PROBLEM-SPECIFIC FEATURES

Feature engineering is the process of creating relevant features from raw data using domain knowledge which help improve the performance of supervised ML models. We introduce novel problem-specific features summarized in Table 3 and categorize them into size-, time-, cost- and due date-related features. Subscripts *AVG* and *STD* stand for the average and standard deviation of the corresponding parameter, respectively. For example, $\tau_{AVG} = \sum_i \sum_j \tau_{ij} / |\mathbf{J}|\,|\mathbf{I}|$ and $\varepsilon_{AVG} = \sum_i \varepsilon_i / |\mathbf{I}|$.

4.1 Makespan Minimization

For makespan minimization, we account for size- and time-related features. Size-related features include the numbers of batch and unit, their product/ratio, the numbers of equations and variables, and sparsity of an instance. In this work, we define sparsity as the ratio of the number of nonzero elements to the total element numbers in the coefficient matrix, i.e., the product of the numbers of variables and constraints.

Table 3. Summary of features

Size-related features	*Time-related features*
Batch number	η
Unit number	η/η_{base}
Size I ($\|\mathbf{I}\| \times \|\mathbf{J}\|$)	Load
Size II ($\|\mathbf{I}\| \times \|\mathbf{J}\| \times \eta$)	$\max_{i,j}\{\tau_{ij}\}/\eta$
Batch-unit ratio	τ_{AVG}
Number of variables	τ_{STD}
Number of equations	$\tau_{AVG}/\|\mathbf{J}\|$
Sparsity	π_{AVG}^{τ}
	π_{STD}^{τ}
	φ_{AVG}^{τ}
	φ_{STD}^{τ}
Cost-related features	*Due date-related features*
γ_{AVG}	ε_{AVG}
γ_{STD}	ε_{STD}
π_{AVG}^{γ}	μ_{AVG}
π_{STD}^{γ}	μ_{STD}
φ_{AVG}^{γ}	λ_{AVG}
φ_{STD}^{γ}	λ_{STD}

Next, time-related features derived from the scheduling horizon are η and η/η_{base}. Based on processing times τ_{ij}, τ_{AVG}, τ_{STD}, $\tau_{AVG}/|\mathbf{J}|$ and load are considered. The load represents how much units would be utilized for processing given batches in terms of time. It is defined as the ratio of the averaged time required for processing all batches and the production capacity with the given scheduling horizon, i.e., $\sum_i \tau_i{}^{AVG} / (|\mathbf{J}| \times \eta)$. Besides, we introduce $\pi_{jj'>j}^{\tau}$ and $\varphi_{ii'>i}^{\tau}$ to denote the inter-unit and inter-batch dissimilarity in processing times τ_{ij}, respectively. They are given by

$$\pi_{jj'>j}^{\tau} = \sum_i \frac{\left|\tau_{ij} - \tau_{ij'}\right| - \min_{i,j,j'>j}\left\{\left|\tau_{ij} - \tau_{ij'}\right|\right\}}{\max_{i,j,j'>j}\left\{\left|\tau_{ij} - \tau_{ij'}\right|\right\} - \min_{i,j,j'>j}\left\{\left|\tau_{ij} - \tau_{ij'}\right|\right\}} \tag{10}$$

$$\varphi_{ii'>i}^{\tau} = \sum_j \frac{\left|\tau_{ij} - \tau_{i'j}\right| - \min_{i,i'>i,j}\left\{\left|\tau_{ij} - \tau_{i'j}\right|\right\}}{\max_{i,i'>i,j}\left\{\left|\tau_{ij} - \tau_{i'j}\right|\right\} - \min_{i,i'>i,j}\left\{\left|\tau_{ij} - \tau_{i'j}\right|\right\}} \tag{11}$$

based on the normalized absolute similarity metric (also known as Manhattan distance). Their respective averages and standard deviations are considered as time-related features: π_{AVG}^{τ}, π_{STD}^{τ}, φ_{AVG}^{τ}, and φ_{STD}^{τ}.

4.2 Cost Minimization

In addition to the previously introduced features, we consider cost- and due date-related features for cost minimization. Cost-related features are based on processing costs γ_{ij}: γ_{AVG} and γ_{STD}. In a similar way to calculating the dissimilarity features in terms of τ_{ij}, we introduce $\pi^{\gamma}_{jj'>j}$ and $\varphi^{\gamma}_{ii'>i}$ to denote the inter-unit and inter-batch dissimilarity in γ_{ij}, respectively, and account for their averages and standard deviations, i.e., π^{γ}_{AVG}, π^{γ}_{STD}, φ^{γ}_{AVG}, and φ^{γ}_{STD}.

As due date-related features, we basically consider ε_{AVG} and ε_{STD}. Others are derived from the time window length of batch i defined as $\varepsilon_i - \rho_i$ indicating the allowable production time period for batch i. Note that $\rho_i = 0$ in this study, but the due date-related features introduced here can be readily extended to the problem case where ρ_i are nonzero. We introduce two parameters, i.e., μ_t and λ_i; μ_t denotes the degree of overlap of the time windows of batches at time period t, implying the potential unit busyness at each period; λ_i is the ratio between the time window length and the average processing time of batch i, indicating the time window tightness for each batch. Smaller λ_i means that batch i has a tighter time window. Finally, we account for their averages and standard deviations as due date-related features as presented in Table 3.

5. RESULTS AND DISCUSSION

We train the classifier and regressor using scikit-learn Python library (Pedregosa et al., 2011) for instances with each objective function, and evaluate their performances on the test data. As the data preprocessing for the classification model, we perform a min-max normalization on feature data to make the model less sensitive to the scale of features and more accurate. Also, we make balanced data by under-sampling, i.e., reducing the size of the abundant class. For the regression model, we use a log-transformation for CPU time data to reduce its large variability and thus improve the model's predictive performance.

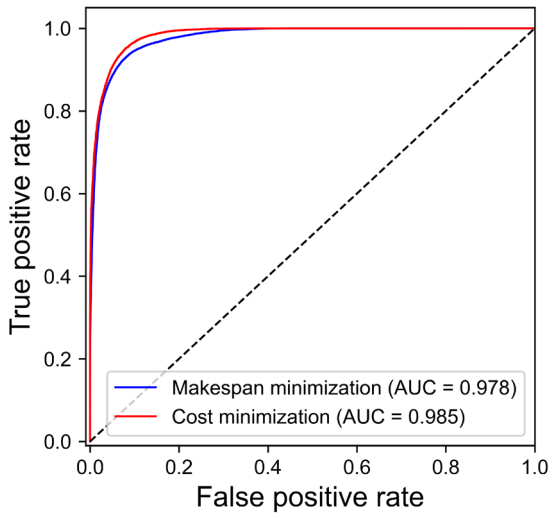

Fig. 1. ROC curves of classification models on test data

5.1 Feasibility Classification

Both trained logistic regression models (with c of 10) for makespan and cost minimization problems show good classification performances (see Fig. 1). In makespan minimization, F1 score and AUC on the test data are 0.90 and 0.978, respectively. They are 0.90 and 0.985, respectively, in cost minimization. The developed classification models can potentially allow online scheduler to avoid online optimization executions that are (likely to be) infeasible.

Table 4 presents 7 top-ranked features selected in the classifier and their estimated coefficients for each scheduling objective. The coefficient indicates the expected change in the log-odds of being in the class 1 (in which an instance has no feasible solution) when the corresponding normalized feature increases from 0 to 1 while fixing others. For makespan minimization, the load is the most significant feature, and an instance with a larger load has a higher possibility of having no feasible solution. Of course, in this case, we can find any feasible solution with a very long horizon, so η has a negative coefficient. Lower τ_{AVG} and higher τ_{STD} increase the possibility that an instance has any feasible solution. Increasing batch-unit ratio and size II negatively impacts the instance's feasibility. The batch number has a negative coefficient since, in our data set, the instance with a very high number of batches has a high unit umber and exhibits less possibility that it belongs to class 1. For cost minimization, the due date-related features are significant. Scheduling MIP instance that has a less tight time window (i.e., larger λ_{AVG}) and a higher ε_{STD} is likely to be feasible. On the other hand, increasing λ_{STD} and ε_{AVG}, and decreasing μ_{STD} make an instance infeasible. Batch-unit ratio and τ_{AVG} have the same effect on the instance feasibility as in makespan minimization.

Table 4. 7 top-ranked features from the classification

Makespan minimization		Cost minimization	
Feature	Coefficient	Feature	Coefficient
Load	25.8	λ_{AVG}	-137
Batch number	-22.4	Batch-unit ratio	73.0
η	-17.7	ε_{STD}	-50.1
τ_{AVG}	16.3	λ_{STD}	20.3
Batch-unit ratio	16.16	ε_{AVG}	17.6
Size II	14.3	μ_{STD}	-13.6
τ_{STD}	-11.3	τ_{AVG}	7.79

5.2 Computational Time Prediction

Fig. 2 and Fig. 3 show visual comparisons of the actual and predicted CPU times for two objective functions. Quantitatively, MSEs for predicting $\log_{10}$ CPU time are 0.482 and 0.535 in makespan minimization and cost minimization, respectively; the average misprediction is less than 5.5. Note that the optimized tree-depth and number of trees are 25 and 30, respectively. For makespan minimization, the batch-unit ratio has the highest feature importance, computed as the

normalized total MSE decrease by the corresponding feature. Other significant features are τ_{AVG}, τ_{STD}, and the inter-batch and inter-unit dissimilarity in τ_{ij}. Without the dissimilarity features, MSE increases by 24 % in the case of makespan minimization. For cost minimization, ε_{AVG} is the most significant feature followed by τ_{AVG}, Size II, and μ_{AVG}. Increasing ε_{AVG} and τ_{AVG} can lead to longer scheduling horizon η while fixing the batch and unit numbers. Thus, without them, η becomes the most important feature and MSE is slightly increased. Other supervised learning algorithms (e.g., a hybrid of linear regression and RF) can be tested to improve the regression model's interpretability and performance, and it is our ongoing research.

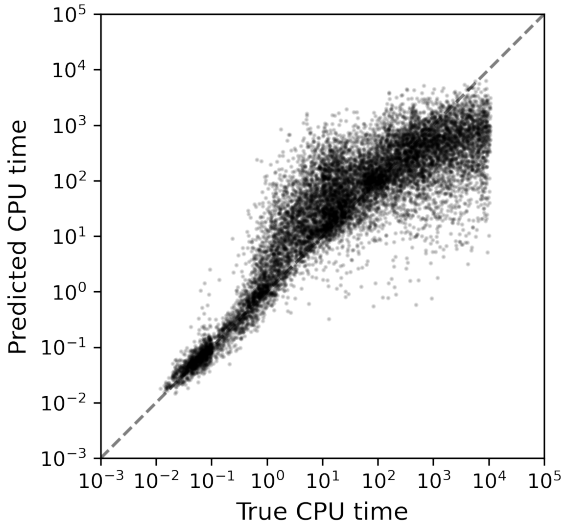

Fig. 2. Predictions on test instances for makespan minimization

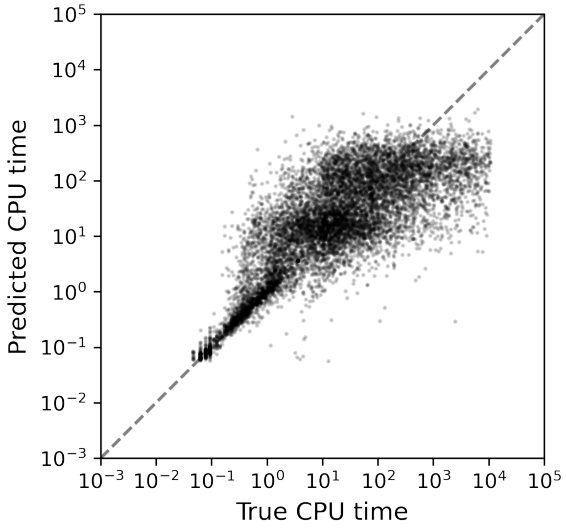

Fig. 3. Predictions on test instances for cost minimization

6. CONCLUSIONS

We employed supervised machine learning techniques to predict the performance of chemical production scheduling MIP models. We trained logistic regression and random forests models to predict model feasibility and computational time, respectively, for two objective functions (i.e., makespan minimization and cost minimization). The classifier and regressor, trained based on the features introduced in this work, yielded good predictive performances on the test instances for both objectives. We focused on single-stage multiple-unit environment, but this data-based approach can be extended to other scheduling objective functions or more complex production environments. Indeed, this work can support the development of an intelligent online scheduler for choosing adequate horizon and re-optimization time-step.

REFERENCES

Aggarwal, C.C. (2014). *Data classification: Algorithms and Applications*, CRC Press, United States.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32.

Gupta, D., Maravelias, C.T., and Wassick, J.M. (2016). From rescheduling to online scheduling, *Chemical Engineering Research and Design*, 116, 83–97.

Gupta, D., and Maravelias, C.T. (2019). On the design of online production scheduling algorithms, *Computers and Chemical Engineering*, 129, 106517.

Hutter, F., Xu, L., Hoos, H.H., and Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206(1), 79–111.

James, G., Witten, D., Hastie, T. and Tibshirani, R. (2013). *An introduction to statistical learning: With applications in R*, New York: Springer.

Komarek, P. (2004). *Logistic regression for data mining and high-dimensional classification*, Carnegie Mellon University.

Leyton-Brown, K., Nudelman, E. and Shoham, Y. (2009). Empirical hardness models: Methodology and a case study on combinatorial auctions. *Journal of the ACM (JACM)*, 56(4), 1–52.

Leyton-Brown, K., Hoos, H.H., Hutter, F., and Xu, L. (2014). Understanding the empirical hardness of NP-complete problems, *Communications of the ACM*, 57(5), 98–107.

Maravelias, C.T. (2021). *Chemical Production Scheduling: Mixed-Integer Programming Models and Methods*, Cambridge University Press.

Muñoz, E. and Capón-García, E. (2019). Systematic approach of multi-label classification for production scheduling, *Computers & Chemical Engineering*, 122, 238–246.

Smith-Miles, K., and Lopes, L. (2012). Measuring instance difficulty for combinatorial optimization problems. *Computers and Operations Research*, 39(5), 875–889.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. (2011). Scikit-learn: machine learning in python, *Journal of Machine Learning Research*, 12, 2825–2830.

Ravikumar, P., Wainwright, M.J. and Lafferty, J.D. (2010). High-dimensional Ising model selection using $\ell 1$ -regularized logistic regression. *The Annals of Statistics*, 38(3), 1287–1319.