IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 31, 2021, accepted March 7, 2021, date of publication March 10, 2021, date of current version March 19, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3065391*

# Optimal Decision Tree for Cycle Time Prediction and Allowance Determination

**CHIH-HUA HSU**
Department of Information Management, Chang Jung Christian University, Tainan City 711301, Taiwan
e-mail: chhsu@mail.cjcu.edu.tw

This work was supported in part by the Ministry of Science and Technology of Taiwan under Contract MOST 109-2218-E-992-005, and in part by the Intelligent Manufacturing Research Center from the Featured Areas Research Center Program within the Framework of the Higher Education Sprout Project by the Ministry of Education in Taiwan.

**ABSTRACT** The due-date quotation is a key performance indicator for managing customer orders which would influence customer acceptance and/or the future potential lateness penalty. The production cycle time and allowance time are added and used as the due date of order. The objective is to maximize the hit rate which is the percentage of the orders fulfilled within the time limit of quoted due date. Under the framework of supervised machine learning, we explore the new developments in feature selection and the optimal decision tree to predict cycle time by using mixed-integer optimization. Cycle time allowance could be added to the predicted cycle time or incorporated in an optimization problem as a managerial decision variable. Case studies are used to demonstrate the effectiveness of this approach, and their performances are comparable to the other popular ensemble tree approaches, such as random forests and gradient boosting.

**INDEX TERMS** Allowance determination, cycle time prediction, gradient boosting, hit rate, local search in a decision tree, mixed-integer optimization, optimal decision tree, random forests.

## I. INTRODUCTION

The due-date quotation is a key performance index in competitive manufacturing and service environments [21]. Those companies with on-time deliveries of the orders have a competitive edge. For due date management (DDM) policy, one important performance index is the hit rate which is the percentage of the orders that are fulfilled within the time limit of quoted due date.

There are intensive researches in cycle time prediction due to its importance and practicality. Tree-based methods are popular due to their interpretability [7], [8]. Dunn [14] formulated the search of the optimal decision tree as a mixed-integer optimization (MIO) problem and circumvented the suboptimal solution of the traditional greedy approaches. In [2], Bertsimas and Dunn reported recent amazing results in optimized-based approaches to solving machine learning problems. The sparse regression methods therein could be used to select key features that will reduce the dimensionalities of the MIO in the optimal tree search.

Our objective in this research is to maximize the hit rate of a DDM policy. An allowance time is added to the cycle time which is used to negotiate with the customer as the due date of order. We utilize the newly developed methodology in feature selection and the optimal decision tree to predict cycle time or maximize the hit rate based on MIO and explain in detail in Section II.

The associate editor coordinating the review of this manuscript and approving it for publication was Diego Oliva.

### A. LITERATURE REVIEW

In this section, we provide a literature review on different approaches for cycle time management.

The first line of research is to predict the cycle time. In [32], Sha and Hsu used a simulation package to generate the training data with 92 factors based on a wafer fabrication factory in Taiwan which consisted of 53 workstations and 301 machines. In nine rules out of three categories, the neural-network-based models have lower tardiness than regression-based and conventional rules. Based on data of 118 predictor variables collected from a real factory, Baskus *et al.* [1] compared the performance of clustering, K-nearest neighbors, and CART based on four measures. By simulating a reduced-scale fab of flash non-volatile memory using Autosched AP simulation software, Meidan *et al.* [26] reduced 182 features to 50 by conditional mutual information maximization algorithm and then used selective naive Bayesian classifier for prediction model which was comparable to that of a neural network,

C5.0 decision tree, and multinomial logistic regression. Verwer and Zhang [35] collected simulated raw data based on a real wafer fabrication factory in Shanghai, and 108 factors were selected from 774 candidates by a regression-based factor selection model. A hybrid model of adaptive fuzzy c-means algorithm and parallel back propagation network was proposed to predict the cycle time and achieved better performance than backpropagation network and multivariate linear regression in terms of the mean absolute deviation and the cycle time variance. Besides these methods, the other popular approaches include fuzzy logic [11], hybrid approach [10], LSTM [36], production simulation [19], and queueing theory [13].

The second approach is to combine scheduling and the due-date quotation. Keskinocak and Tayur [22] offered an extensive survey on the DDM policy consisting of a due date setting policy and a sequencing policy. They categorized the problems into offline/online models for DDM, DDM with service constraints, and DDM with price and order selection decisions. Gordon *et al.* [16] surveyed the common due date assignment and scheduling problems of the deterministic models about single machine and parallel machines under different performance criteria. Please refer to these two survey articles for further references. Kuo *et al.* [23] used neural networks to predict the WIP levels of individual toolsets and reduced 42.1% of the cycle time for a real fab in Taiwan in about one year by identifying the key features in the predictors and prioritizing them by sensitivity analyses.

The last line of research is to maximize the hit rate of a policy. Wang *et al.* [38] considered production management planning under the theory of constraint framework to improve the hit rate and compared the performance of three experimental scenarios of a job shop. Chen and Wang [12] framed DDM as a supervised machine learning problem, that is, a neural network was trained and predicted on training and testing datasets, respectively, and demonstrated its effectiveness over seven other policies in terms of hit rate.

The rest of the paper is organized as follows: Section II describes the methodology, Section III presents the illustrative examples and results, and Section IV includes the conclusion and further directions.

## II. METHODOLOGY

Given a set of features $x \in R^p$ in a manufacturing system and its corresponding cycle time $y$, the regression machine learning approach is to find a function $\hat{y} = f(x)$ so that the error between $\hat{y}$ and $y$ is small. The popular performance measure for regression is the mean square root (MSE) or the mean absolute error (MAE).

We apply the recent results in feature selection and optimal decision trees to solve the problem of cycle time prediction and allowance determination.

### A. FEATURE SELECTION

For a problem with a small number of samples or a large number of features, it is beneficial to find the important features to speed up the training time and enhance the interpretability of the model.

For a linear regression problem $f(x) = \beta x$, it is NP-complete to find the sparsest number of important features in the 0-pseudo norm, i.e., the number of non-zero coefficients in $\beta$ [28]. Instead, a convex $l$-1 norm penalty on $\beta$, namely Lasso, is added and solved [18], [33]. Wang *et al.* [38] proved that Lasso regression is equivalent to a robust regression problem with a feature-wise uncoupled $l^2$-norm uncertainty set.

Bertsimas and Van Parys [5] developed a cutting-plane algorithm for the exact sparse regression problem and solved the problems with 100-thousand sample sizes and features within 10 minutes. In [4], the authors compared the out-of-sample accuracy and false detection rate for five methods under six noise/correlation scenarios. Based on extensive experiments, they demonstrated that Lasso performs poorly in low noise settings and is comparable with other methods as noise increases which explains the robustness aspect of Lasso.

### B. OPTIMAL DECISION TREE

Decision trees are commonly used for regression and classification problems due to their nice interpretability [7], [8]. In CART [7], the stage-wise downward procedures are used to find the splitting feature of one branch node which maximizes its information gain of the Gini index instead of the direct performance of prediction error for the regression problem. The greedy approach would produce a tree that might not be optimal.

Due to the tremendous improvement of hardware speedup and software algorithms, commercial optimization solvers could solve large problems within an appropriate timeframe. Bertsimas and Dunn [2], [14] formulated the search of the optimal decision tree as a mixed-integer optimization (MIO) problem. Under this framework, they consider the possibility of hyperplane splits, i.e., many features are used in the splitting consideration, and a linear prediction function for each leaf when it is a regression problem. The optimal decision tree has different decision variables and constraints for parallel/hyperplane splits and constant/linear predictions. For optimal classification trees with hyperplane splits and optimal regression trees with linear predictions [14], they have 20 and 13 sets of constraints, respectively. To save the space and presentation, we will only explain the objective function, key and new decision variables, and the main constraints, and refer to the dissertation [14] for the full problem formulation.

#### 1) MAXIMIZE THE HIT RATE INDIRECTLY

The first approach is to minimize the MAE of cycle time prediction and use the optimal tree to compute the hit rate afterward.

Taking Figure 1 as an example, the depth of the tree is 2, and there are 3 branch nodes {A, B, C} and 4 leaf nodes {1, 2, 3, 4}. The set of branch nodes and leaf nodes are denoted by $T_B$ and $T_L$, respectively, for any tree. The numbers of sample size and features are $n$ and $p$, respectively. We use

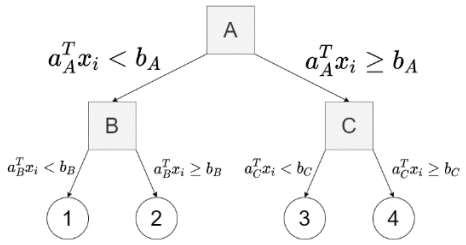


**FIGURE 1.** A decision tree for illustration.

the notation $[n]$ to denote the set $\{1, 2, \ldots, n\}$. The symbol $x_i$ represents the $i$th sample point, for all $i \in$ [n].

At each branch node, the decision variables are the coefficients $\mathbf{a} \in R^p$, and its corresponding threshold $b$. For parallel and hyperplane splits, the decision variables $a_{jt}s$ of $\boldsymbol{a}$ in the branch nodes of Figure 1 are binary and in the range of $[-1, 1]$, respectively, where $j \in [p], t \in T_B$. To simplify the discussion, we will limit ourselves to the case of parallel splits in the remainder of this article. By adding an appropriate small number derived from the dataset [14], the strict inequality in each branch node could be transformed into a non-strict inequality so that the new optimization problem does not violate the feasibility of the original problem and is solvable by an optimization solver.

For the branch nodes, we consider the binary decision variables $\{d_t\}$ and impose the following constraints:

$$d_t = \sum_{j=1}^{p} a_{jt}, \quad \forall t \in \mathrm{T}_B \tag{1}$$

At each branch, if $d_t$ is 1, then one of the coefficients $\{a_{jt}\}$ is 1. Otherwise, the branch node becomes a leaf. That is, the variables $\{d_t\}$ are used to control the complexity of the tree.

At the leaves, a set of binary decision variables $\{z_{it}\}$ is used to decide which data sample belongs to which leaf. The binary decision variable $\{z_{it}\}$ is 1 if the $i$th sample is assigned to the $t$th leaf for $i \in [n], t \in \mathrm{T}_L$. Each data sample is assigned to only one leaf which is imposed as another constraint. Given a hyper-parameter $N_l$, we could impose the constraint that the minimum number of samples in each leaf is greater or equal to $N_l$.

We use a linear prediction function at each leaf which takes the form

$$y(x) = \beta_t^T x + \beta_{0t} \tag{2}$$

In this case, the prediction function for the $i$th sample is

$$f_i = \sum_{t \in \mathrm{T}_L} \left(\beta_t^T x_i + \beta_{0t}\right) z_{it}, \forall i \in [n] \tag{3}$$

The equalities (3) are nonlinear due to the products of the variables $\beta_t$, $\beta_{0t}$, and $z_{it}$ which could be linearized by using the popular big-M argument as follows:

$$\begin{aligned} -M_f(1 - z_{it}) &\leq f_i - (\beta_t^T x + \beta_{0t}) \\ &\leq M_f(1 - z_{it}), \forall i \in [n], t \in \mathrm{T}_L. \end{aligned} \tag{4}$$

If $z_{it} = 1$, then the inequalities (4) become $f_i = \beta_t^T x + \beta_{0t}$. If $z_{it} = 0$, both inequalities hold and become unconstrained because $M_f$ is a big number. The number $M_f$ could be determined beforehand based on the data [14].

The binary decision variables $\{r_{jt}\}$ in the inequalities (5) control the complexity of the prediction function by imposing the constraints with another big-M constant $M_r$. The interpretation is the same as those in the inequalities (3). For data with $p$ features and the tree depth of $D$, the number of binary decision variables $\{r_{jt}\}$ is $p \times 2^D$.

$$-M_r r_{jt} \leq \beta_{jt} \leq M_r r_{jt}, \quad \forall j \in [p], \forall t \in T_L \tag{5}$$

The objective function (6) is to minimize the summation of the prediction error $\sum_{i=1}^{n} L_i$ divided by the baseline error $\hat{L}$, the complexity of the decision tree in terms of the number of branch decision nodes $\sum_{t \in \mathrm{T}_B} d_t$, and the weighted number of the linear prediction functions of all leaves $\sum_{t \in \mathrm{T}_L} \sum_{j=1}^{p} r_{jt}$ in which the last two terms are used to regulate the model from overfitting. The hyper-parameters $\alpha$ and $\lambda$ are tuned in the validation stage. The best parameters are then used in the testing stage to compare the out-of-sample performance of its policy.

$$\min \frac{1}{\hat{L}} \sum_{i=1}^{n} L_i + \alpha \sum_{t \in \mathrm{T}_B} d_t + \lambda \sum_{t \in \mathrm{T}_L} \sum_{j=1}^{p} r_{jt} \tag{6}$$

If the objective is to minimize MSE, then the prediction error $L_i$ is $(f_i - y_i)^2$ for each data sample, and the baseline error $\hat{L}$ is the average number $\sum_i y_i/n$. If the performance criterion is the MAE, then the prediction error $L_i$ is $|f_i - y_i|$. Its baseline MAE error $\hat{L}$ does not have an analytical expression and could be obtained by solving another optimization problem.

Once the optimal tree is determined, the hit rate is evaluated by adding an allowance to each order, that is, the objective of maximizing the hit rate is obtained afterward and indirectly.

### 2) MAXIMIZE THE HIT RATE DIRECTLY

In this subsection, the hit rate is maximized directly.

For the $i$th sample pair of real and predicted cycle time $(y_i, \hat{y}_i)$ in the training set, respectively, if $\hat{y}_i + al \geq y_i$ for a given allowance parameter $al$, that is, the quoted due date $\hat{y}_i + al$ is greater than or equal to the real completion time $y_i$, then the target is achieved. If $y_i > \hat{y}_i + al$, then the target is missed. It makes the problem amenable for an optimization solver to solve, a small positive constant $\varepsilon$ based on the dataset is added to the above strict inequality and it becomes $y_i \geq \hat{y}_i + al + \varepsilon$. Next, a binary decision variable $H_i$ and a big number M are utilized to transform the above if/then constraints into the usual constraints as follows.

$$\begin{aligned} M \times H_i &\geq \hat{y}_i + al - y_i + \varepsilon(1 - H_i) \\ &\geq -M \times (1 - H_i) \end{aligned} \tag{7}$$

If $H_i = 1$, then $\hat{y}_i + al \geq y_i$, and the order is on time. Otherwise, $y_i \geq \hat{y}_i + al + \varepsilon$.

For the $i$th sample pair, the hours of earliness $Early_i$ and lateness $Late_i$ of an order are $\max(\hat{y}_i + al - y_i, 0)$ and $\max(y_i - \hat{y}_i - al, 0)$, respectively.

$$\min 1 - \sum_{i=1}^{n} \frac{H_i}{n} + ef \sum_{i=1}^{n} Early_i + lf \sum_{i=1}^{n} Late_i \qquad (8)$$

The new objective function (8) is to maximize the hit rate ($\sum H_i/n$) or equivalently minimize the miss rate ($1-\sum H_i/n$) where the number $n$ is the sample size. Even though the objective function is to minimize the miss rate, the weighted prediction error and its corresponding constraints still need to be included in the problem formulation. Otherwise, the solution would produce a zero miss rate by assigning large numbers of predicted cycle times which is not appropriate or even unfavorable for the company to negotiate with the customers.

In the objective function (8), two weighting factors $ef$ and $lf$ for earliness and lateness hours, respectively, are used to balance the miss rate from overfitting. For this reason, the weighted prediction error in the grid search for the optimal parameter combination could not start from zero in which the only objective would be the miss rate. Another interpretation is to minimize the deviation errors from the real cycle time in a just-in-time manner [21]. Please refer to the book [30] for further discussions and implications of different performance indexes.

The branch and leaf constraints in Section II-B.1 still need to be imposed for this new objective function.

### C. LOCAL SEARCH

For a sample size of $n$ and the tree depth of $D$, the number of binary decision variables $\{z_{it}\}$ for assigning the samples to all leaves is $n \times 2^D$. The computational complexity of the optimal decision tree grows rapidly with the number of samples and the depth of the tree.

Ensemble methods, such as random forests [9] and gradient boosting [15], achieve state-of-the-art performance, but ensembles of trees lack interpretability [8].

Bertsimas and Dunn [2], [14] utilize the benefits of random forests and generate the initial trees by the training dataset. For each tree in the random forests, a local optimal tree is found by using similar local search methods as those algorithms in [7], [27]. The algorithms search all possible split locations which have polynomial time complexity [14]. The best single local search tree out of the validation set is used and then tested for out-of-sample prediction performance. The performance results of this approach with one decision tree are comparable to those of random forests and gradient boosting in 26 real-world datasets for regression problems [14, Table 5.1].

## III. CASE STUDY

In [12], one of the case studies with 120 data samples with detailed values was collected from a wafer fab located in the Central Taiwan Science Park in Taichung, Taiwan. Six features out of 12 candidates are chosen by backward elimination

**TABLE 1. Parameter and performance for the training data.**

| Average allowance | Compared Methods (CM) | HR of CM | HR of LS(2) | HR of LS(2, 4) |
|---|---|---|---|---|
| 55 | Constant due date | 55% | 72% | 66% |
| 55 | Slack policy | 99% | 72% | 66% |
| 55 | Total work content | 100% | 72% | 66% |
| 50 | Confidence interval | 99% | 70% | 63% |
| 55 | Selective allowance | 77% | 72% | 66% |
| 55 | Random allowance | 78% | 72% | 66% |
| 63 | Inclusion interval | 100% | 74% | 69% |
| 55 | Neural network | 100% | 72% | 66% |

**TABLE 2. Parameter and performance for the testing data.**

| Average allowance | Compared Methods (CM) | HR of CM | HR of LS(2) | HR of LS(2, 4) |
|---|---|---|---|---|
| 55 | Constant due date | 36% | 73% | 67% |
| 55 | Slack policy | 36% | 73% | 67% |
| 55 | Total work content | 36% | 73% | 67% |
| 50 | Confidence interval | 37% | 70% | 67% |
| 43 | Selective allowance | 38% | 70% | 63% |
| 55 | Random allowance | 35% | 73% | 67% |
| 57 | Inclusion interval | 41% | 73% | 70% |
| 74 | Neural network | 70% | 77% | 80% |

regression analyses. The features $\{x_1, x_2, \ldots, x_6\}$ are job size, work-in-process (WIP), queue length before the bottleneck, queue length on the route, average waiting time, and factory utilization, respectively. These predictors are used to estimate its cycle time $y$.

In a stationary system, the cycle time is proportional to its WIP by Little's law [25]. For the above dataset, the rounded correlation coefficient of cycle time and WIP is 0.66. The minimum, maximum, mean, and standard deviation of the ratio (cycle time / WIP) are 0.72, 1.31, 0.96, and 0.13, respectively.

In the following, the simulation programs are written in Python and Julia on a Windows 10 notebook with Intel Core i7-10510U 2.3 GHz processor and 16 GB RAM.

### A. COMPARISON WITH DIFFERENT METHODS

To keep consistent with the results in the paper [12], the raw data are normalized into [0.1, 0.9] by linear min-max normalization, and the first 90 and the last 30 data samples are used for training and testing stages, respectively. In Tables 1 and 2, eight methods in the paper [12] are listed for comparison in the first three columns of each table. For example, the hit rate (HR) of constant due date policy with an allowance of 55 hours is 55% in Table 1.

By using feature selection software by Interpretable AI [20] under programming language Julia 1.5.2 [6], the rounded coefficients of the most important variables $x_2, x_4$, and $x_6$ are 0.4866, 0.4234, and 0.0890, respectively. The summation of these coefficients should be 1 instead of 0.9990 due to a round-off error. Next, we use a local search (LS) decision tree with parallel splits in each branch node and a linear prediction function of important features in each leaf. The numbers in the parentheses denote the used features in which LS(2) and LS(2, 4) represent $x_2$ and $x_2, x_4$, respectively. A grid search over the maximum depth

**TABLE 3.** Parameters and performance results for the optimal decision tree.

| Method | Features | Depth | Splitting rule | Computational budget (s) | Mean HR (standard deviation) | Mean hours of earliness (standard deviation) | Mean hours of lateness (standard deviation) | Training time (s) |
|---|---|---|---|---|---|---|---|---|
| I1 | 2 | 2 | parallel | 3600 | 0.73 (0.06) | 111 (17) | 31 (6) | 4 |
| I2 | 2 | 3 | parallel | 3600 | 0.00 (0.00) | 127 (23) | 37 (8) | 3600 |
| I3 | 2 | 2 | hyperplane | 600 | 0.75 (0.08) | 96 (15) | 24 (10) | 600 |
| I4 | 2 | 2 | hyperplane | 3600 | 0.75 (0.07) | 96 (18) | 24 (9) | 3600 |
| I5 | 6 | 2 | hyperplane | 600 | 0.72 (0.06) | 122 (42) | 31 (8) | 600 |
| I6 | 6 | 2 | hyperplane | 3600 | 0.76 (0.07) | 127 (27) | 28 (13) | 3600 |
| D1 (3) | 2 | 2 | parallel | 3600 | 0.75 (0.26) | 168 (62) | 12 (8) | 3 |
| D2 | 2 | 3 | parallel | 3600 | 0.00 (0.00) | 102 (70) | 18 (16) | 3600 |
| D3 | 2 | 2 | hyperplane | 600 | 0.85 (0.09) | 160 (26) | 11 (9) | 600 |
| D4 | 2 | 2 | hyperplane | 3600 | 0.75 (0.26) | 137 (51) | 12 (7) | 3600 |
| D5 | 6 | 2 | hyperplane | 600 | 0.67 (0.25) | 138 (59) | 26 (16) | 600 |
| D6 | 6 | 2 | hyperplane | 3600 | 0.63 (0.23) | 121 (53) | 33 (17) | 3600 |
| D7 (1) | 2 | 2 | parallel | 3600 | 0.75 (0.26) | 168 (62) | 12 (8) | 5 |
| D8 (2) | 2 | 2 | parallel | 3600 | 0.75 (0.26) | 168 (61) | 12 (7) | 4 |
| D9 (4) | 2 | 2 | parallel | 3600 | 0.75 (0.26) | 171 (62) | 12 (7) | 5 |

(1) $lf = 0.001 \times i$ (2) $lf = 0.002 \times i$ (3) $lf = 0.003 \times i$ (4) $lf = 0.004 \times i$

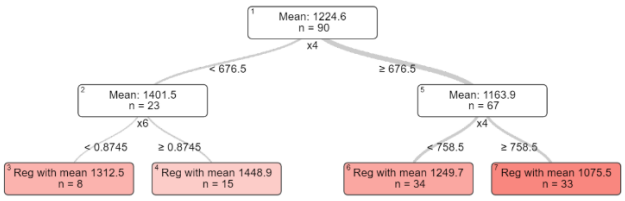


**FIGURE 2.** The decision tree for LS(2) in Tables 1 and 2.

of {2, 3, 4} and λ of {0.0005, 0.001, 0.005} is used. The hyper-parameter λ when fitting the linear prediction equations in each leaf is used to regularize the regression from overfitting.

Figure 2 shows the final local search tree for the experiment LS(2). For root node 1, the sample size ($n$) is 90 with a mean value of 1224.6 hours and the split variable is $x_4$ with a threshold of 676.5 units. If $x_4 < 676.5$ and $x_6 < 0.8745$, then it follows the left most decision path and reaches the leaf node 3. In the leaf nodes {3, 4, 6, 7}, their corresponding prediction functions are $0.7852x_2+306.9$, $3.288x_2-2785.9$, $1.747x_2-987.1$, and $1.961x_2-1413.2$, respectively. For example, if $(x_2, x_4, x_6) = (1, 223, 670, 0.7)$, then its decision path is through the node sequence $1 \rightarrow 2 \rightarrow 3$ and its predicted cycle time is $0.7852 \times 1,223+306.9 \cong 1,267$ hours.

For the local search trees to make a fair comparison, we use the same allowance hours for training and testing data as those in each first column in Tables 1 and 2. Once a decision tree is obtained, their predicted cycle times and the corresponding hit rate are computed as shown in the last 2 columns of Tables 1 and 2. For allowance hours in [43, 50] and [55, 57], the hit rates of LS(2) are 70% and 73%, respectively, in Table 2. That is, the hit rate is a non-decreasing function of the allowances. We will discuss another example in Section III-B.3.

The local search methods perform better for the out-of-sample testing in this simulation and show a smaller gap between the training and testing data, while the other compared methods are overfitting. For a moderate amount of data points, it demonstrates that the decision tree methods are more favorable than data-intensive methods like neural networks. Another merit of a decision tree is its interpretability and its decision paths are easily visualized as shown in Figure 2.

### B. OPTIMAL DECISION TREE

In this section, we construct the optimal regression tree based on mixed-integer optimization (MIO). Ten randomized datasets from the original dataset are drawn. For each experiment, 50%, 25%, and 25% of the randomized dataset are partitioned into training, validation, and testing sets, respectively, and finally the statistical values of ten simulations are computed. To utilize the approaches of [14], the raw data of each feature are normalized into [0, 1] by linear min-max normalization.

We use Python version 3.7.6 [34] and Gurobi version 9.0.3 [17] to develop the simulation programs in this section.

The results in this section are provided in Table 3.

### 1) MAXIMIZE THE HIT RATE INDIRECTLY

Once the optimal tree is determined by solving the MIO problems in Section II-B.1, the hit rate is evaluated after the allowance is added. We use 74 hours for training and testing stages as in Section III-A.

We run many simulations and provide some of them in Table 3. Each row in Table 3 represents an experiment. The first five parameters in each simulation are the sequence number beginning with the letter I for indirect approach, the number of features used, the depth of the decision tree, the splitting rule at each branch node, and the upper limit of computation time in seconds for each simulation. The next three performance indexes are the statistics of 10 simulation runs for each policy, that is, the mean hit rate, the mean hours of earliness and lateness, and their corresponding standard deviation (std) in the parenthesis. The last one is the average training time in seconds.

We use a linear prediction function for every leaf for all simulations. To find the best-validated hyper-parameters of $\alpha$ and $\lambda$ by a grid search, the parameters are varying from [0, 0.02] with an increment of 0.01, that is, each randomized dataset is run $(3 \times 3 =)$ 9 times for the training stage and the one with the optimal objective for the validation set is used for out-of-sample prediction.

One could utilize warm starts to improve MIO performance, e.g., CART for parallel splits and constant prediction. It is unclear what constitutes a good warm start for hyperplane splits and a linear prediction function. For the simulations in Table 3, we solve all MIO problems without warm starts.

For experiment I1 in Table 3, it takes only an average of 4 seconds to obtain the optimal decision tree. Its hit rate is 73% which is 3% lower than the optimal experiment I6 with a 1-hour computation time. Its standard deviation of HR is 6%.

For experiment I2, the depth of the tree is 3 and that is the only difference from experiment I1, and its MIO solution is far away from the optimal one so that its hit rate is zero even after 1 hour of computation. This simulation illustrates the complexity of a larger tree due to the exponential growth of binary decision variables $\{z_{it}\}$.

For the parameter settings of I3 and I4, the only difference is their computational budget. Their performance results are quite close to each other and their rounded average hit rates have the same value of 0.75. Their worst-case prediction accuracy of 10 decision trees happens to be the fifth data set and has the same value of 0.633. The duality gaps of these two optimization problems are 70% and 50%, respectively, when the time limits are reached. In Figures 3 and 4, we only draw the decision functions for the outer decision paths for easier visualization. As seen from the figures, the sign of the branching features and the features of the prediction function in each leaf node might change. For example, the left path of node B in Figures 3 and 4 are $-0.114x_2 + 0.313x_4 < 0$ and $0.059x_2 - 0.094x_4 < 0.016$, respectively. The prediction function of node 1 in Figures 3 and 4 are $50x_2 + 1410$ and $50x_4 + 1229$, respectively. In [24], Li and Belfold provided an example of the instability of the decision tree, proved the cause of the instability problem, and proposed an improved algorithm for classification problems.

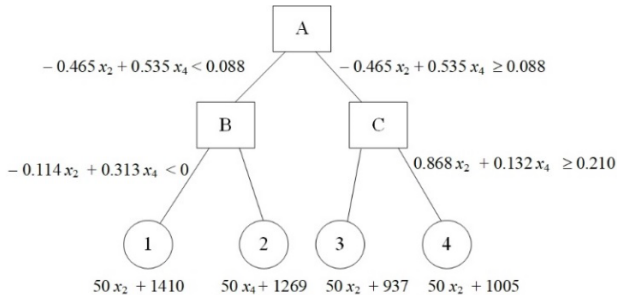


**FIGURE 3.** The worst-case decision tree of I3.

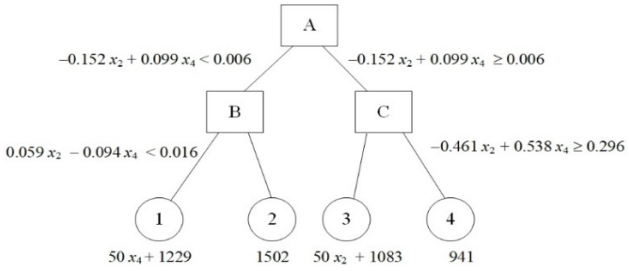


**FIGURE 4.** The worst-case decision tree of I4.

For I5 and I6, we use 6 features instead of 2 in I3 and I4. In this case, they have larger numbers of decision variables and the duality gaps are larger than those of I3 and I4. Their hit rates are comparable with those of I3 and I4, but their mean hours and standard deviations of earliness and lateness are larger than those of I3 and I4.

### 2) MAXIMIZE THE HIT RATE INDIRECTLY

In this subsection, the hit rate is maximized directly.

For the data set under investigation, the range of the real cycle time is within the range of [849, 1810] hours, so $\varepsilon = 10^{-3}$ is chosen in the inequalities (7) and it does not change the feasibility of the original problem.

We use 74 hours for training and testing stages as in Section III-A. The MIO problems in Section II-B.2 are solved, and the simulation results are shown in Table 3.

The first parameter in each simulation is the sequence number beginning with the letter D for the direct approach. For D1 to D6, the simulation setups are the same as those in I1 to I6, correspondingly. For the hyper-parameters in the objective function (8), we set $ef = 0.001 \times i$ and $lf = 0.003 \times i$ for $i \in \{0, 1, 2\}$. That is, the late order gets a larger penalty which would keep it from happening and reach a hit of the target due date.

The experiments D1 and D3 have higher hit rates and lower mean hours of lateness than those of I1 and I3, but it comes with a cost of higher mean hours of earliness. In the

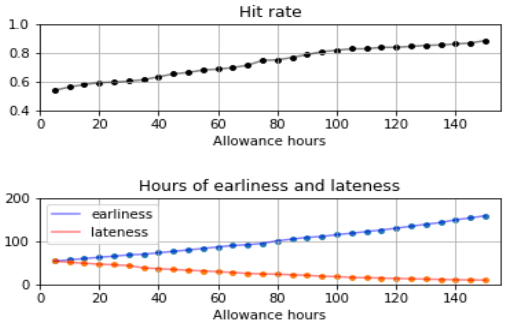


**FIGURE 5.** Allowance hours as a decision variable and their corresponding performance indexes.

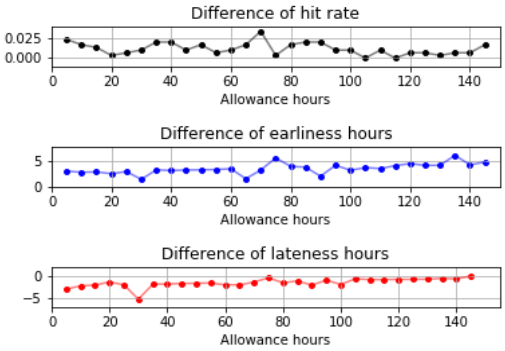


**FIGURE 6.** Performance differences under two consecutive allowance hours.

next subsection III-B.3, we will discuss another aspect of this trade-off when the allowance is a decision variable.

The experiment D2 has the same problem as experiment I2. For the experiments D5 and D6, the hit rates deteriorate as compared with those of I5 and I6.

To understand the influence of the hyper-parameter, we only vary the coefficient of *lf* from 0.003 in D1 to 0.001, 0.002, and 0.004 in D7 to D9, respectively, while keeping the other parameters unchanged. Their performance results are similar to each other.

#### 3) THE ALLOWANCE AS A DECISION VARIABLE

We will treat the allowance as a managerial decision parameter, and explore its impact and tradeoff versus key performance indexes. There are two main approaches based on the methods in the previous two subsections. The first direction is to find the best decision tree, and then use it to find the performance measures for each varying number of allowances. The second one is to solve the corresponding optimization problem for each parameter change. We will focus on the first method because it is computationally easier.

In terms of the relative better performances, we will take experiment I3 as an example. Figure 5 shows the resulting hit rate and hours of earliness and lateness when the range of the allowance hours is changing within the interval [5, 150] with an increment of 5 hours. The dot points represent the simulation results, and the interpolation line is used for easier visualization. In the upper picture, the hit rate is a non-decreasing function of the allowance hours. In the lower picture, the earliness and lateness hours are an increasing and a decreasing function of the allowance hours, respectively.

The manager could decide the allowance hour and its corresponding impact on three performance indexes for each order. For example, when the hit rate is 85%, its corresponding earliness and lateness are 139 and 13 hours in Figure 5, respectively. For the best HR policy D3 in Table 3, its hit rate, hours of earliness, and hours of lateness are 0.85%, 160, and 11, respectively. It is worth further investigation to understand the problem formulation of the objective function (8) and the tuning of its hyper-parameters.

To understand the behavior of each performance index, we consider the difference between them under two consecutive allowance hours. For all pictures in Figure 6, each coordinate of the $x$-axes in [5], [145] represents the left point of every two consecutive allowance hours. Taking the top picture as an example, the first point of the $y$-axis is 0.023 hours, that is, HR(10) – HR(5) = 0.023 where HR(10) is the hit rate when the allowance hours are 10. For the sets of {105, 110} and {115, 120} hours, they have the same hit rates of 0.83 and 0.84, respectively, and hence zero difference. The lateness is a decreasing function of allowance hours, so the differences are all negative in the bottom picture.

The performance indexes themselves are the first-order differences of allowance hours and their derivatives are not monotone as shown in Figure 6, so these three performance indexes are neither concave nor convex functions in terms of the allowance hours. In this case, one could not find the global extremum.

### C. LOCAL SEARCH

For larger problems, it takes a prohibitively long time to find the optimal decision tree. If the training time is limited and a decision needs to be made promptly, e.g., the changing manufacturing environment, then its usage is limited. In this section, we explore computationally efficient local search approaches and compare their performances.

In this section, ten randomized datasets from the original dataset are drawn. For each experiment, 75% and 25% of the randomized dataset are partitioned into training and testing sets, respectively, and finally, the statistical values of ten simulations are computed. We run simulations with tuning hyper-parameters with a validation dataset, but they do not improve the performance. So we use the default values in the software packages and there is no validation dataset in the following results.

The allowance of 74 hours is used for the training and testing stages in this section. We use Python for ensemble learning and Julia for the local search tree. To make the comparisons consistent, we randomly shuffle the data 10 times under Python and then partition the data into training and testing datasets for each method accordingly.

We run many simulations in this section and some results are provided in Table 4. We add a corresponding sequence

**TABLE 4.** Parameters and performance results for ensemble methods and local search decision tree.

| Method | Features | Depth | Splitting rules | Prediction function | Mean training HR (std) | Mean testing HR (std) | Mean testing earliness hours (std) | Mean testing lateness hours (std) | Training time (s) |
|---|---|---|---|---|---|---|---|---|---|
| RF (1) | 6 | u | parallel | constant | 0.92 (0.02) | 0.73 (0.07) | 99 (15) | 19 (4) | 0.99 |
| RF (2) | 6 | 4 | parallel | constant | 0.82 (0.02) | 0.74 (0.06) | 98 (14) | 20 (4) | 0.98 |
| GB (1) | 6 | u | parallel | constant | 0.97 (0.02) | 0.71 (0.09) | 96 (22) | 23 (9) | 0.10 |
| GB (2) | 6 | 4 | parallel | constant | 0.99 (0.01) | 0.71 (0.08) | 100 (18) | 23 (7) | 0.12 |
| LS (1) | 6 | 6 | parallel | constant | 0.76 (0.02) | 0.72 (0.08) | 103 (15) | 26 (9) | 0.86 |
| LS (2) | 6 | 5 | parallel | all | 0.74 (0.05) | 0.71 (0.09) | 95 (15) | 21 (14) | 123.68 |
| LS (3) | 6 | 5 | parallel | (2) | 0.79 (0.04) | 0.77 (0.07) | 105 (20) | 18 (7) | 13.80 |
| RF (3) | 2 | u | parallel | constant | 0.93 (0.02) | 0.72 (0.08) | 103 (17) | 20 (6) | 0.90 |
| RF (4) | 2 | 3 | parallel | constant | 0.77 (0.02) | 0.74(0.07) | 99 (15) | 20 (5) | 0.89 |
| GB (3) | 2 | u | parallel | constant | 0.91 (0.03) | 0.73 (0.08) | 98 (21) | 23 (7) | 0.07 |
| LS (4) | 2 | 6 | parallel | constant | 0.77 (0.03) | 0.73 (0.07) | 107 (17) | 30 (11) | 1.13 |
| LS (5) | 2 | 5 | parallel | (4) | 0.77 (0.03) | 0.71 (0.08) | 100 (23) | 25 (10) | 5.77 |
| LS (6) | 2 | 5 | parallel | (2) | 0.77 (0.02) | 0.72 (0.08) | 98 (21) | 23 (8) | 5.53 |
| LS (7) | 2 | 5 | parallel | (2, 4) | 0.75 (0.04) | 0.74 (0.09) | 96 (19) | 18 (7) | 8.86 |
| LS (8) | 2 | 4 | (2, 4) | (2, 4) | 0.74 (0.03) | 0.73 (0.08) | 97 (18) | 17 (5) | 215.49 |
| LS (9) | 2 | 4 | (2, 4) | (4) | 0.76 (0.03) | 0.72 (0.07) | 101 (17) | 21 (8) | 115.12 |
| LS (10) | 2 | 4 | (2, 4) | (2) | 0.77 (0.03) | 0.72 (0.08) | 102 (24) | 22 (6) | 151.11 |
| LS (11) | 2 | 4 | (4) | (2) | 0.77 (0.02) | 0.71(0.07) | 98 (20) | 23 (7) | 45.25 |

number in the first parameter after each method for easier reference. To understand the overfitting problem, we provide a column on the mean hit rate of the training stage and its corresponding standard deviation.

### 1) LOCAL SEARCH WITHOUT FEATURE SELECTION

In this subsection, we use all six features to train the machine learning models. The results are in the top 7 rows in Table 4.

We use ensemble methods of random forests (RF) and gradient boosting (GB) under Python, and the version of Scikit-learn is 0.22 [29]. The default values are used except the criterion is the MAE and a random_state value is assigned for reproducible results.

For RF(1), it is random forests with 6 features and the depth is unconstrained, denoted by *u*. The larger gap between the training and testing stages demonstrates the overfitting of this (default) parameter setting and dataset. For its first experiment out of 10 randomized datasets, the minimum, the maximum, and the mean depths of 100 trees for random forests are 9, 18, and 11.66, respectively.

For GB(1), the default value in Scikit-learn will produce 100 trees for each experiment. In this case study, all 1,000 trees in 10 randomized datasets have a depth of 3. But some trees are degenerate and have less than 8 leaf nodes as

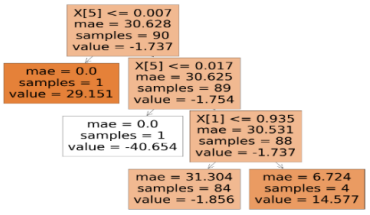


**FIGURE 7.** The 57th tree in the first experiment of gradient boosting.

shown in Figure 7. The Python language starts from the index of 0, so X[5] is the 6th feature in the first branch node. Its left branch is a leaf and it has only one sample with a value of 29.151.

For the local search (LS), the default values are used except a random_seed value is assigned for reproducible results. For the experiment LS(1), a grid search for the allowable depth of a tree is from 2 to 6 for each experiment. The minimum, maximum, and mean depths of 10 locally optimal trees are 2, 6, and 3.5, respectively.

After the mean depth of the local search trees is obtained, we rerun the simulations of RF and GB and limit the maximum depth to be 4. It does increase the hit rate of random forests from 0.73 to 0.74 in RF(2) and reduce the

**TABLE 5. Important features and their statistics.**

| feature | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| frequency | 237 | 1000 | 298 | 999 | 278 | 489 |
| Average weighting | 0.0424 | 0.4699 | 0.0536 | 0.4548 | 0.0478 | 0.0745 |

overfitting problem. For gradient boosting, the original depth is 3 and the overfitting problem becomes severe in GB(2) when we force the depth to be 4.

For LS(2) and LS(3), we use 6 features and the second feature in the prediction function, respectively. The experiment LS(3) achieves the highest hit rate in Table 4.

As shown in Table 4, the hit rates of the local search trees are comparable with random forests and gradient boosting. The merit of the local search method is a single tree that is more interpretable.

### 2) LOCAL SEARCH WITH FEATURE SELECTION

To understand the behavior of important features, we randomly shuffle the dataset 1,000 times and find its statistics as shown in Table 5. The first feature appears 237 times with an average weighting of 0.0424. The sum of the weightings in each experiment is 1, so the sum of the products of the frequency and its corresponding average weighting of 6 features is 1,000. The most important features are 2 and 4. We will use features 2 and 4 for this subsection and their results start from the eighth row in Table 4.

For the local search (LS), a grid search over the maximum depth and the hyper-parameter $\lambda$ of $\{0.0005, 0.001, 0.005\}$ is used to find the optimal parameters. The mean depth of 10 locally optimal trees in LS(4) is 3.

The mean depths of unconstrained RF and GB are 11 and 3, respectively, so we only rerun the simulation for RF. The hit rate of RF(4) is 0.74 and the gap between the training and testing hit rate goes down from 0.21 in RF(3) to 0.03 in RF(4). The feature selection procedure improves the HR of GB from 0.71 in GB(2) to 0.73 in GB(3), while the HR of RF stays the same.

In LS(8) to LS(10), we use the second and fourth features as the splitting hyperplanes. Since a hyperplane has greater flexibility to partition the space, we reduce its maximum search depth of the tree to be 4. For more complex structures, they take longer to compute the tree. For these experiments, their performance indexes do not improve.

In Table 4, the experiment LS(3) has the highest hit rate of 0.77 and its hours of earliness are 9 hours higher than that of the second-best LS(7). The second-best policy is LS(7) because it has lower earliness and lateness hours as compared with those of RF(2) and RF(4) at a cost of larger training time.

## IV. CONCLUSION

We use the optimal decision tree based on mixed-integer optimization and the local search trees to predict the cycle time and investigate the influence of the allowances on the hit rate, hours of earliness, and hours of lateness for the due-date quotation problem. From the results in the examples, our approaches improve the performance of the neural network approach in the cited work, the single decision tree is more interpretable, and their performances are comparable to the other popular ensemble tree approaches, such as random forests and gradient boosting.

For future research, the complexity of the optimal decision tree and the stability of decision trees deserve further investigation.

As mentioned before, the number of binary decision variables $\{z_{it}\}$ depends on its sample size. In [35], Verwer and Zhang provide an interesting integer programming formulation for a classification tree that does not depend on its sample size.

As seen in Figures 3 and 4, the decision trees are unstable. In [31], Rudin viewed it as an advantage, and "the domain expert can pick the one that is the most interpretable." In [3], Bertsimas *et al.* formulated and solved the robust classification problems of support vector machines, logistic regression, and decision trees, and improved the out-of-sample accuracy for 75 data sets.

## ACKNOWLEDGMENT

The author would like to thank Prof. H.-C. Yang and Dr. Y.D. Zhuo for their helpful comments. He would also like to thank Mr. T. Chang for drawing Figure 1.

## REFERENCES

[1] P. Backus, M. Janakiram, S. Mowzoon, G. C. Runger, and A. Bhargava, "Factory cycle-time prediction with a data-mining approach," *IEEE Trans. Semicond. Manuf.*, vol. 19, no. 2, pp. 252–258, May 2006.
[2] D. Bertsimas and J. Dunn, *Machine Learning Under a Modern Optimization Lens*. Charlestown, MA, USA: Dynamic Ideas LLC, 2019.
[3] D. Bertsimas, J. Dunn, C. Pawlowski, and Y. D. Zhuo, "Robust classification," *INFORMS J. Optim.*, vol. 1, no. 1, pp. 2–34, Winter 2019.
[4] D. Bertsimas, J. Pauphilet, and B. Van Parys, "Sparse regression: Scalable algorithms and empirical performance," *Stat. Sci.*, vol. 35, no. 4, pp. 555–578, Nov. 2020.
[5] D. Bertsimas and B. Van Parys, "Sparse high-dimensional regression: Exact scalable algorithms and phase transitions," *Ann. Statist.*, vol. 48, no. 1, pp. 300–323, Feb. 2020.
[6] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Rev.*, vol. 59, no. 1, pp. 65–98, Jan. 2017.
[7] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. New York, NY, USA: Chapman & Hall, 1984.
[8] L. Breiman, "Statistical modeling: The two cultures," *Statist. Sci.*, vol. 16, no. 3, pp. 199–231, 2001.
[9] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001.
[10] T. Chen, A. Jeang, and Y.-C. Wang, "A hybrid neural network and selective allowance approach for internal due date assignment in a wafer fabrication plant," *Int. J. Adv. Manuf. Technol.*, vol. 36, nos. 5–6, pp. 570–581, Mar. 2008.
[11] T. Chen, "An effective fuzzy collaborative forecasting approach for predicting the job cycle time in wafer fabrication," *Comput. Ind. Eng.*, vol. 66, no. 4, pp. 834–848, Dec. 2013.
[12] T. Chen and Y.-C. Wang, "A nonlinearly normalized back propagation network and cloud computing approach for determining cycle time allowance during wafer fabrication," *Robot. Comput.-Integr. Manuf.*, vol. 45, pp. 144–156, Jun. 2017.
[13] S.-H. Chung and H.-W. Huang, "Cycle time estimation for wafer fab with engineering lots," *IIE Trans.*, vol. 34, no. 2, pp. 105–118, Feb. 2002.
[14] J. Dunn, "Optimal trees for prediction and prescription," Ph.D. dissertation, MIT Sloan School Manage., Cambridge, MA, USA, 2018.

[15] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001.
[16] V. Gordon, J.-M. Proth, and C. Chu, "A survey of the state-of-the-art of common due date assignment and scheduling research," *Eur. J. Oper. Res.*, vol. 139, no. 1, pp. 1–25, May 2002.
[17] Gurobi Optimization, LLC. (2020). *Gurobi Optimizer Reference Manual*. [Online]. Available: http://www.gurobi.com
[18] T. Hastie, R. Tibshirani, and M. Wainwright, *Statistical Learning with Sparsity: The Lasso and Generalizations*. New York, NY, USA: Chapman & Hall, 2015.
[19] L. Y. Hsieh, K.-H. Chang, and C.-F. Chien, "Efficient development of cycle time response surfaces using progressive simulation metamodeling," *Int. J. Prod. Res.*, vol. 52, no. 10, pp. 3097–3109, May 2014.
[20] Interpretable AI, LLC. (2020). *Interpretable AI Documentation*. [Online]. Available: https://www.interpretable.ai
[21] F. R. Jacobs and R. Chase, *Operations and Supply Chain Management*, 16th ed. New York, NY, USA: McGraw-Hill, 2020.
[22] P. Keskinocak and S. Tayur, "Due date management policies," in *Handbook of Quantitative Supply Chain Analysis*, D. Simchi-Levi, S. D. Wu, and Z.-J. M. Shen, Eds. New York, NY, USA: Springer, 2004, pp. 485–554.
[23] C.-J. Kuo, C.-F. Chien, and J.-D. Chen, "Manufacturing intelligence to exploit the value of production and tool data to reduce cycle time," *IEEE Trans. Autom. Sci. Eng.*, vol. 8, no. 1, pp. 103–111, Jan. 2011.
[24] R.-H. Li and G. G. Belford, "Instability of decision tree classification algorithms," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Edmonton, AB, Canada, Jul. 2002, pp. 570–575.
[25] J. D. C. Little, "A proof for the queuing formula: $L = \lambda W$," *Oper. Res.*, vol. 9, no. 3, pp. 383–387, 1961.
[26] Y. Meidan, B. Lerner, G. Rabinowitz, and M. Hassoun, "Cycle-time key factor identification and prediction in semiconductor manufacturing using machine learning and data mining," *IEEE Trans. Semicond. Manuf.*, vol. 24, no. 2, pp. 237–248, May 2011.
[27] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees," *J. Artif. Intell. Res.*, vol. 2, pp. 1–32, Aug. 1994.
[28] B. K. Natarajan, "Sparse approximate solutions to linear systems," *SIAM J. Comput.*, vol. 24, no. 2, pp. 227–234, Apr. 1995.
[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, and B. Thirion, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, no. 85, pp. 2825–2830, Oct. 2011.
[30] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 5th ed. New York, NY, USA: Springer, 2016.
[31] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Mach. Intell.*, vol. 1, no. 5, pp. 206–215, May 2019.
[32] D. Y. Sha and S. Y. Hsu, "Due-date assignment in wafer fabrication using artificial neural networks," *Int. J. Adv. Manuf. Technol.*, vol. 23, nos. 9–10, pp. 768–775, May 2004.
[33] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. Roy. Stat. Soc., B, Methodol.*, vol. 58, no. 1, pp. 267–288, Jan. 1996.
[34] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA, USA: CreateSpace, 2009.
[35] S. Verwer and Y. Zhang, "Learning optimal classification trees using a binary linear program formulation," in *Proc. 33rd AAAI Conf. Artif. Intell.*, Honolulu, HI, USA, 2019, pp. 1625–1632.
[36] J. Wang, J. Zhang, and X. Wang, "A data driven cycle time prediction with feature selection in a semiconductor wafer fabrication system," *IEEE Trans. Semicond. Manuf.*, vol. 31, no. 1, pp. 173–182, Feb. 2018.
[37] J. Wang, J. Zhang, and X. Wang, "Bilateral LSTM: A two-dimensional long short-term memory model with multiply memory units for short-term cycle time forecasting in re-entrant manufacturing systems," *IEEE Trans. Ind. Informat.*, vol. 14, no. 2, pp. 748–758, Feb. 2018.
[38] M. Wang, M. Li, C. Chen, H. Chen, and C. Tsai, "An experimental study to improve due-date performance," *Asian J. Qual.*, vol. 10, no. 3, pp. 13–36, Dec. 2009.
[39] H. Xu, C. Caramanis, and S. Mannor, "Robust regression and lasso," *IEEE Trans. Inf. Theory*, vol. 56, no. 7, pp. 3561–3574, Jul. 2010.

**CHIH-HUA HSU** received the B.S. degree in electrical engineering from National Sun Yat-sen University, in 1989, the M.S. degree in control engineering from National Chiao Tung University, Hsinchu, in 1991, and the Ph.D. degree in aerospace engineering from The University of Texas at Austin, Austin, TX, USA, in 1998. Since 2004, he has been an Assistant Professor with the Department of Information Management, Chang Jung Christian University, Tainan City, Taiwan. His research interests include operations research, machine learning, and supply chain management.