Hybrid machine learning/constraint programming method for proactive and reactive scheduling applied to PCB assembly

Youssef Karouma[1,2,3], Christian Artigues[2], Le Toan Duong[1], Houssem-eddine Gharbi[1] and Romain Guillaume[3]

***Abstract*— Printed Circuit Board (PCB) assembly is a demanding production environment where due-date compliance must be maintained under variable operating conditions. This paper addresses scheduling under uncertainty for uniform parallel machines and introduces a proactive-reactive rescheduling policy learned as a decision tree. Starting from an initial schedule, the policy evaluates the ongoing scenario at data-driven checkpoints—whose timing is learned by a classifier—and classifies it as Good or Bad with respect to due-date performance. If Good, the schedule is continued; if Bad, a rescheduling action is triggered. The decision tree is trained from simulation labels and complemented by a post-processing step that encourages early decisions to support timely reactions. The evaluation protocol examines whether the learned policy increases the proportion of on-time jobs on scenarios predicted as adverse for the initial schedule. Experiments are conducted with stochastically generated speed scenarios to emulate variability; future work will extend the study with realistic data from a Schaeffler manufacturing plant. Results highlight the feasibility of learning proactive rescheduling rules that balance responsiveness and planning stability in PCB assembly.**

I. INTRODUCTION

We study a PCB assembly application at a Schaeffler plant. Production flows through Front-End, ICT, Back-End and Packaging. We focus on Front-end with SMT placement lines operating continuously. Each PCB has two jobs: face 1 then face 2, with a precedence constraint. Sequence-dependent setups apply on a line when two jobs are executed consecutively. Lines are modeled as uniform parallel machines; speeds vary over time in uncertain way, and affect processing durations. Our objective is to schedule jobs so as to meet due dates (minimize total tardiness) while coping with these speed variations.

Under uncertainty, three categories of scheduling approaches are commonly used. First, discrete scenario models: either equiprobable sets used to evaluate candidate schedules, or probabilistic/two-stage formulations where a first-stage baseline is decided before uncertainty is revealed and a recourse policy adapts after scenario realization [1]. Variants further impose an ordering on scenario likelihood by using fuzzy/possibilistic degrees, which encode graded plausibility rather than precise probabilities and guide decisions toward robust performance with respect to more plausible scenarios [2], [3]. Second, uncertainty-set models: interval or continuous sets handled by robust optimization, where feasibility and performance are guaranteed against all realizations inside a budgeted or interval set; this limits conservatism while providing worst-case protection [4]. Third, proactive reactive policies: adaptive two-stage rescheduling that updates a baseline at predefined events or periods [5], [6], decision-tree policies with checkpoints fixed in advance [7], and our variant in which both the checkpoint times and the classification rules are learned from data, aiming to trigger earlier yet reliable reactions under speed variability.

Contribution

Building on the above overview of scheduling under uncertainty, we target the PCB uniform parallel machines setting with time-varying line speeds and sequence-dependent setups. Rather than fixing rescheduling times a priori or committing to a specific scenario structure, we aim for a data-driven policy that decides both when to react and which schedule to apply. This naturally motivates the decision-tree framework detailed next, where checkpoints and rules are learned to balance responsiveness and planning stability.

Indeed, a scheduling decision tree with 2 nodes is proposed. At 1st node, an initial schedule is given along with a learned condition, and 2 branches. This condition classifies the scenarios into *good* or *bad* towards this schedule. Being *good* means that the schedule reaches a certain level of performance in the objective function, which is the opposite in the case of *bad*. The condition also allows pursuing the same schedule on the *good* scenarios (1st branch), and updates the schedule for *bad* ones (2nd branch). The updated schedule is considered for the 2nd node. Then for a real ongoing scenario, we can dynamically move from a schedule to a schedule-update, based on the path of conditions that the scenario satisfies until now. Through our last experimentations, this approach improved the performance on a set of scenarios considered initially as *bad* for the 1st schedule. It allowed also anticipating early reaction moments compared to pure reactive methods.

The paper is structured as follows. Section 2 presents the

[1]Y. Karouma, L. Duong, H.-E. Gharbi are with Schaeffler group, Toulouse, France {youssef.karouma, le.toan.duong, houssem-eddine.gharbi}@vitesco.com

[2]Y. Karouma, C. Artigues are with LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France {youssef.karouma, christian.artigues}@laas.fr

[3]Y. Karouma, R. Guillaume are with IRIT, Université de Toulouse, Toulouse, France {youssef.karouma, romain.guillaume}@irit.fr

This work is supported by ANITI (Artificial Intelligence Toulouse Institute), Toulouse, France

deterministic form of the problem, as well as the constraint programming model used to solve it. It also explains uncertainty model. In section 3, we explain our scheduling decision tree policy to handle uncertainties, and how we use it in online phase for an ongoing scenario. Numerous experiments are described in section 4, in order to analyze the effectiveness of our approach in detecting bad situations and improving the scheduling outcome. We also assess its ability to anticipate reactions the earliest possible.

II. PROBLEM STATEMENT

A. Deterministic form

The uniform parallel machines scheduling problem consists of assigning jobs to machines that have different speed performances. We consider that each job corresponds to a PCB face (face 1 or face 2) which should be assigned and scheduled on a production line. We have a set of n jobs $\mathscr{J} = \{J_i | i = 1, \dots n\}$. The "face 2" jobs are denoted $\mathscr{J}_2$, while the "face 1" jobs are in set $\mathscr{J} \setminus \mathscr{J}_2$. Every job J_i has a nominal duration p_i. It is described as a quantity of job that would be executed slower or faster depending on the line speed, which, in the real use case, evolves over time. We have a set of m lines. The dependence between the duration of a job J_i and the speed of a line k is illustrated in the following equation:

$$\int_{t=S_i}^{t=S_i+p_{i,k}} v_k(t)dt = p_i \tag{1}$$

where S_i is the starting time of job J_i, $p_{i,k}$ is the real duration of J_i if assigned to line k, and $v_k(t)$ describes the evolution of line k speed over time. However this speed evolution is uncertain and, for computing an initial schedule an average estimate $\bar{v}_k$ is used.

The schedule has to satisfy 3 main constraints :

- **Precedence constraint**: 1st face of each PCB should be processed before its 2nd face.
- **setup delay**: a setup time is needed if a job J_j is performed directly after a job J_i on a same line k. This setup is expressed as : $s_{i,j,k}$.
- **Non overlap** between jobs assigned to the same line.

The objective is to find a schedule that satisfies the constraints and minimizes the total tardiness : $min \sum_{i \in \mathscr{J}_2} T_i$ where the tardiness of a "face 2" job $j \in \mathscr{J}_2$ is equal to $T_i = max(0, C_i - d_i)$. C_i and d_i are respectively the completion date and due date of a PCB product, after finishing its 2nd face.

B. Constraint programming model

We propose CP model (2–7). We use the following decision variables:

- $M_i \in \{1, \dots, m\}$: line to which $J_i \in \mathscr{J}$ is assigned
- $S_i \geq 0$: start time of job $J_i \in \mathscr{J}$.
- $C_i \geq 0$: completion time of job $J_i \in \mathscr{J}$.
- $T_i \geq 0$: tardiness of job $J_i \in \mathscr{J}_2$.

The schedule must satisfy the following constraints.

- **Precedence constraint**: Each "face 2" job $J_i \in \mathscr{J}_2$ cannot start before the completion of its corresponding "face 1" job $\pi(i) \in \mathscr{J} \setminus \mathscr{J}_2$

$$S_i \geq S_{\pi(i)} + p_{\pi(i),M_{\pi(i)}} \tag{2}$$

- **Duration and Completion time**: For each job $J_i \in \mathscr{J}$,

$$p_{i,M_i} = \frac{p_i}{\bar{v}_{M_i}} \tag{3}$$

$$C_i = S_i + p_{i,M_i} \tag{4}$$

- **No overlap with setups**: Any distinct two jobs $J_i, J_j \in \mathscr{J}$ scheduled on the same line must not overlap and the setup times needed in between must be satisfied.

$$\begin{aligned} M_i = M_j = k \quad \Rightarrow \quad & S_j \geq S_i + p_{i,k} + s_{i,j,k} \quad \vee \\ & S_i \geq S_j + p_{j,k} + s_{j,i,k} \end{aligned} \tag{5}$$

- **Tardiness**: For each "face 2" job $J_i \in \mathscr{J}_2$,

$$T_i = \max(0, C_i - d_i) \tag{6}$$

- **Objective function**:

$$\min \sum_{i \in \mathscr{J}_2} T_i \tag{7}$$

Note this model is implemented in the google CP-SAT solver. In particular constraints (5) are modeled via `AddNoOverlap` and `AddCircuit` constraint, building a Hamiltonian path on each line k.

C. Uncertainty model : scenario definition

As already mentioned, lines speeds $v_k(t)$ are not known in advance. They evolve over time in ways we cannot perfectly predict. Also, since job durations depend directly on these speeds, the durations $p_{i,k}$ themselves are uncertain at the start. For example, in Figure 1, we have two possible scenarios having completely different speed behaviors. This makes the problem harder to solve. Moreover, for the sake of simplification, we suppose that speeds are piecewise functions of time.

III. HANDLING UNCERTAINTIES WITH PROACTIVE REACTIVE APPROACH : SCHEDULING DECISION TREE

A. Mathematical formulation

Since lines speeds can decelerate abruptly, reactive scheduling is too late to be effective. Thus, our approach consists on predicting rules based on speeds events. Every rule allows partitioning scenarios into *Good* or *Bad* towards the current schedule S^0. Being *Good* means we could pursue this schedule until all the tasks are finished, since this scenario is considered good enough in terms of objective function (7). Whereas, being *Bad* means we need to update S^0 and compute a new schedule S^1 at the time when the rule was checked.

Beforehand, here are some key notations and definitions:

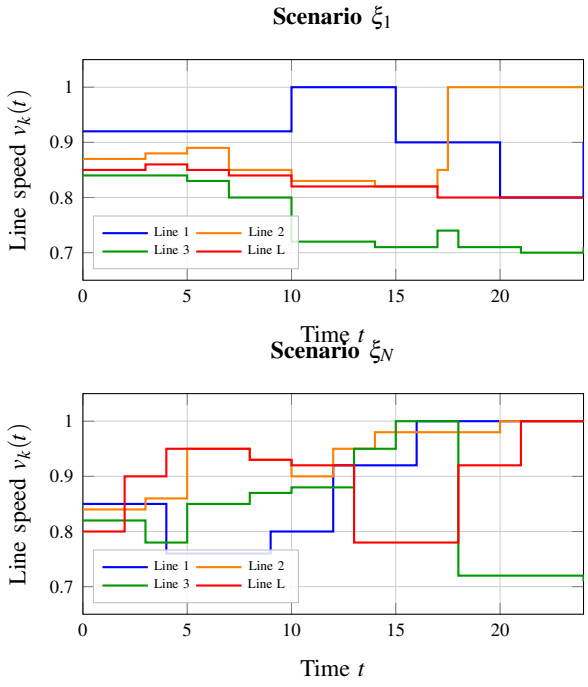


Fig. 1. Example of 2 scenarios for lines speeds evolution over time

- Let $\xi_1[0,t_\theta]$ denote a possible scenario of lines speeds evolution from 0 to t_θ. It's also called a partial scenario.
- $S^0(\xi_1[0,t_\theta])$ means the current status of schedule S^0 on a scenario ξ_1 at time t_θ (finished/ongoing/pending tasks).
- $S^1(t_\theta, S^0(\xi_1[0,t_\theta]), \xi_2[t_\theta,T])$ means that the reschedule/update S^1 was computed at t_θ by considering the current state of the previous schedule S^0 on the scenario ξ_1 at t_θ, and by considering for the future the partial scenario $\xi_2[t_\theta,T]$. T is the total horizon of temporal speeds evolution.
- $\widehat{\Xi}$ means the average scenario on the scenarios of a set Ξ.

The approach is described in the following formula as well as Fig. 2:

$$S^{\pi_\theta}(\xi) = \begin{cases} S^0, & \text{if } \mathrm{Good}_\theta\big(\xi_{[0,t_\theta]}\big) \\ S^1\big(t_\theta, S^0(\xi_{[0,t_\theta]}), \widehat{\Xi}_{\text{bad\_for\_}S^0}[t_\theta,T]\big), & \text{if } \mathrm{Bad}_\theta\big(\xi_{[0,t_\theta]}\big) \end{cases} \tag{8}$$

At the beginning, we compute an initial schedule S^0 based on the average scenario. Then at checkpoint time t_θ, we observe what is actually happening; i.e the partial scenario $\xi_1[0,t_\theta]$; and we classify it as either *Good* or *Bad* for S^0. If it's *Good*, we keep S^0. If it's *Bad*, we update S^0 to S^1. S^1 is computed by considering the current state of S^0 execution at t_θ for the past (i.e : $S^0(\xi_1[0,t_\theta])$), and the average of *bad* scenarios for the future. (i.e :$\widehat{\Xi}_{\text{bad\_for\_}S^0}[t_\theta,T]$).

While θ parameterizes the classifier that learns checkpoint time and rescheduling condition $\pi_\theta = (t_\theta, condition_\theta)$, our minimization problem is as follows :

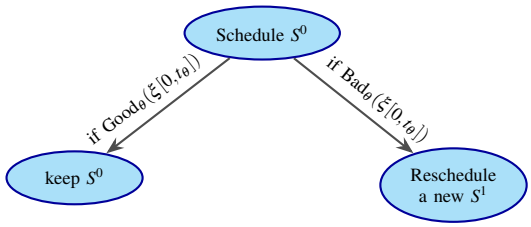


Fig. 2. Small simple tree, with rescheduling condition from S^0 to S^1

$$\min_{\theta\in\Theta} \frac{1}{|\Xi_{\text{train}}|} \sum_{\xi\in\Xi_{\text{train}}} P\big(S^{\pi_\theta}(\xi),\xi\big) \tag{9}$$

where Θ is the set of all possible time points and conditions, P is the global performance of the final scheduling on training set scenarios, namely the sum of job tardinesses.

We choose to solve the problem in a constructive way (as shown in sec III-B): We build a training set of scenarios Ξ_{train}, train a classifier and output $(t_\theta, condition_\theta)$. An example of applying the rescheduling condition to an ongoing scenario and computing the new schedule S^1; is shown in the example: III-C.

B. Learning the rescheduling checkpoint and rule $(t_\theta, condition_\theta)$

To learn the checkpoint time t_θ and the classification rule $condition_\theta$, we proceed through a training phase. First, we need to build the training set where the features are speeds at different time points for every line k $\{v_k(t_i)\}_{k,i}$. In addition, for each time t_i and each line k, we also compute the running average of the speed up to t_i, defined as $\bar{v}_k(t_i) = \frac{1}{i}\sum_{j=1}^{i} v_k(t_j)$, and include $\bar{v}_k(t_i)$ among the features. Classes are *Good* and *Bad*. But we can not directly access to the classification and we need to solve the problem and evaluate the solution on the scenarios to determine the class. More precisely, we sample a set of training scenarios Ξ_{train}, and compute the initial proactive schedule S^0 on the average scenario $\widehat{\Xi}_{train}$. Then, for each scenario $\xi \in \Xi_{train}$, we simulate S^0 on ξ using a simulator (*sim*), and assign it a label towards S^0 based on a threshold for the number of PCBs that are on time (satisfying their due date). Formally:

$$\text{label}(\xi) = \text{sim}(S^0,\xi) \in \{Good, Bad\} \tag{10}$$

To train the best $(t_\theta, condition_\theta)$ which will be used online to decide whether to trigger a reschedule at t_θ, we propose to use a decision tree classifier such that the rule is explicit and understandable by the decision maker.

1) Tree quality criterion: In our context, our goal is therefore to find the *earliest* t_θ at which *Bad* scenarios can still be reliably detected. Before defining formally the criteria, we need to introduce some notations. Let be $\mathscr{T} \in \mathbf{T}$ a decision tree, $\mathbf{B}^{\mathscr{T}}_{Bad}$ the set of branches of $\mathscr{T}$ with majority of *Bad* and $t^{max}_{\mathscr{T}}$ the maximum value of index t over all split features of the decision tree $\mathscr{T}$. We are now able to introduce

a quality criterion based on the notion of *Best Bad-Majority Branch* (BBMB) to select the best $(t_\theta, condition_\theta)$.

For a given tree $\mathscr{T}$, a *bad-majority branch* is a branch having a leaf node where the majority of samples are labeled *Bad*. Among all such branches, the BBMB is the one with the highest proportion of *Bad* samples. We define two metrics to assess the tree's quality:

- **Purity**: the purity of a tree is defined as the purity of its BBMB, i.e: the proportion of *Bad* samples in the BBMB:

$$Purity^{\mathscr{T}} = max_{branch \in \mathbf{B}^{\mathscr{T}}_{Bad}} \frac{n^{branch}_{\text{Bad}}}{n^{branch}_{\text{total}}} = \frac{n^{BBMB}_{\text{Bad}}}{n^{BBMB}_{\text{total}}} \tag{11}$$

- **Coverage**: the coverage of a tree is defined as the proportion of the training set captured by its BBMB:

$$Coverage^{\mathscr{T}} = \frac{n^{BBMB}_{\text{total}}}{N_{\text{train}}} \tag{12}$$

Fig. 3 shows how we compute the purity and the coverage for a tree branch example. The tree is considered of sufficient quality if both metrics exceed predefined thresholds purity_{min} and coverage_{min}.

Formally the best $(t_\theta, condition_\theta)$ is the optimal decision tree among all trees $\mathbf{T}$ of the problem :

$$\min_{\mathscr{T} \in \mathbf{T}} \{t^{max}_{\mathscr{T}} | Purity^{\mathscr{T}} \geq Purity_{min}, Coverage^{\mathscr{T}} \geq Coverage_{min}\} \tag{13}$$

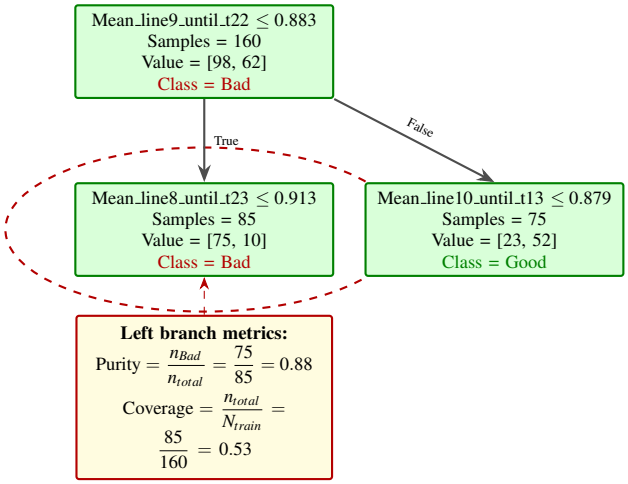


Fig. 3. Example of a classification tree. The left branch (circled in red) contains 85 samples with 75 labeled *Bad*, yielding a purity of $75/85 = 0.88$ and a coverage of $85/160 = 0.53$.

2) Dichotomy-based feature reduction algorithm: To find the best t_θ (Eq. (13)), we propose an iterative dichotomy-based feature reduction. Starting from the full time horizon $t_{max} = T$, we progressively halve the maximum time of the speed features used to train the tree, as long as the quality criterion remains satisfied. The procedure is described in Algorithm 1.

The algorithm outputs the classification rule $condition_\theta$ at the smallest $t_\theta = t^{max}_{tree^*}$ for which the quality criterion is still met on the tree, ensuring that rescheduling is triggered as early as possible while remaining reliable.

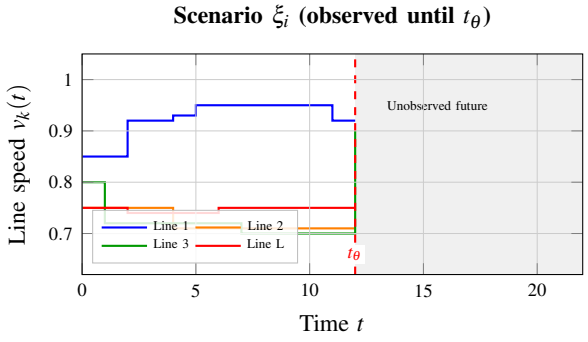
Algorithm 1: Dichotomy-based feature reduction

Input: Training set Ξ_{train} with labels w.r.t. S^0, maximum horizon T
Output: Checkpoint time t_θ and classification rule $condition_\theta$

$t_{max} \leftarrow T$;
$tree^* \leftarrow \emptyset$;
while $t_{max} > 0$ **do**
  Train classification tree $\mathscr{T}$ on features $\{v_k(t), \bar{v}_k(t_i)\}_{t \leq t_{max}}$;
  if $Purity^{\mathscr{T}} \geq Purity_{\min}$ *and* $Coverage^{\mathscr{T}} \geq Coverage_{\min}$ **then**
    $tree^* \leftarrow \mathscr{T}$;
    $t_{max} \leftarrow \lfloor t_{max}/2 \rfloor$;
  else
    break;
return $t_\theta = t^{max}_{tree^*}$, $condition_\theta$ *from* $tree^*$

C. Example

We suppose that the condition for being *Good* for S^0 $\text{Good}_\theta(\xi[0, t_\theta])$ is $v_{Line3}(t = 12) \geq 0.8$. In online phase, as shown in Fig. 4 we have an ongoing scenario ξ_i observed until $t_\theta = 12$, and ignored for the future. The speed of Line3 according to this scenario is $v_{Line3}(t = 12) = 0.7$. Therefore, it's considered as *Bad* for S^0, and then an update should be triggered.



Input scenario at $t_\theta = 12$, with $\mathbf{v\_line3(t = 12)} = 0.7$

Fig. 4. Example of a scenario that is classified as *Bad* for S^0 at $t_\theta = 12$

To update the schedule S^0, we consider its progress on the ongoing scenario at t_θ. The non-started tasks are rescheduled and possibly reallocated to other lines using the same CP model described in Section II-B. Fore sake of simplification, let's say that we have 2 non-started jobs J_1 and J_2 that we can reallocate/reschedule at t_θ. Moreover, we suppose that the set

of *bad* classified scenarios from the training set Ξ_{train} (we described the training phase in Sec. III-B), noted $\Xi_{\text{bad\_for\_}S^0}$, includes just 2 scenarios : ξ_1 and ξ_2. According to the partial future scenario $\xi_1[t_\theta, T]$ (respectively $\xi_2[t_\theta, T]$), every line M_i has an average speed equal to $\overline{v_{M_i}\xi_1[t_\theta,T]}$ (respectively $\overline{v_{M_i}\xi_2[t_\theta,T]}$). Hence, according to the *bads* future average scenario $\widehat{\Xi}_{\text{bad\_for\_}S^0}[t_\theta, T]$, the future speed of each line is $\overline{v_{M_i}\widehat{\Xi}_{\text{bad\_for\_}S^0}[t_\theta,T]} = \frac{\overline{v_{M_i}\xi_1[t_\theta,T]} + \overline{v_{M_i}\xi_2[t_\theta,T]}}{2}$. The total considered scenario is shown in Fig. 5.

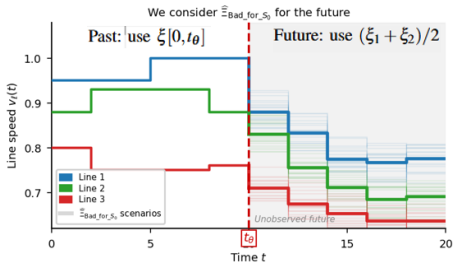


Fig. 5. Scenario used at checkpoint t_θ: the past follows the ongoing scenario $\xi[0,t_\theta]$, while the future uses the mean of bad scenarios $\frac{\xi_1+\xi_2}{2} = \widehat{\Xi}_{\text{bad\_for\_}S^0}[t_\theta, T]$.

Then to compute the duration of each job $J_i \in \{J_1, J_2\}$, we use speed $\overline{v_{M_i}\widehat{\Xi}_{\text{bad\_for\_}S^0}[t_\theta,T]}$ in Eq. (3). Aside from this change in speed input, the CP model of Section II-B is used as is (precedence, sequence-dependent setups, non-overlap) and is solved on $\mathscr{J} = \{J_1, J_2\}$. The resulting line assignments and task sequences are then integrated into the ongoing plan at t_θ: jobs already started or completed keep their positions, while the newly scheduled tasks are positioned after t_θ on their selected lines. Fig. 6 illustrates the process of rescheduling.

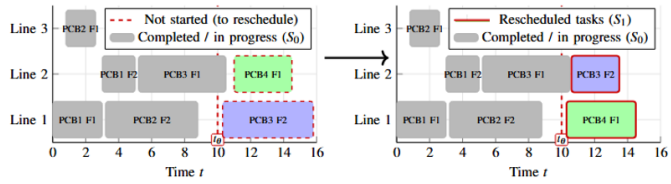


Fig. 6. Rescheduling at checkpoint t_θ: left, schedule S^0 where the vertical dashed line cuts ongoing tasks; right, schedule S^1 where non-started tasks at t_θ are reassigned and inserted without overlap on their selected lines.

IV. Numerical Experiments

Experiments are conducted on stochastically generated speed scenarios to emulate line speed variability. Each training instance consists of 200 scenarios, obtained by sampling 20 scenarios from each of 10 Markov chains. Each chain models the speed of a single line as a 2-state process with states $\{1.0, 0.8\}$, representing nominal and degraded operating conditions respectively. The 10 chains differ by their transition matrices, chosen as follows: $M_1 = [[0.5,0.5],[0.5,0.5]]$, $M_2 = [[0.5,0.5],[0.2,0.8]]$, $M_3 = [[0.6,0.4],[0.2,0.8]]$, $M_4 = [[0.5,0.5],[0.3,0.7]]$, $M_5 = [[0.6,0.4],[0.3,0.7]]$, $M_6 = [[0.5,0.5],[0.1,0.9]]$, $M_7 = [[0.6,0.4],[0.1,0.9]]$, $M_8 = [[0.5,0.5],[0.4,0.6]]$, $M_9 = [[0.6,0.4],[0.4,0.6]]$, $M_{10} = [[0.5,0.5],[0.25,0.75]]$. A common seed parameter controls the sampling within each chain before merging all scenarios into the final dataset, that we call a training instance. We sample 13 training instances with different seeds.

For each training instance, in order to label each scenario in this instance as *Good* or *Bad* towards S^0, we use a threshold of having at least 40% of in-time PCBs. This value reflects the operational requirement at the Schaeffler plant, where a schedule is considered acceptable only if a sufficient share of PCBs meets its due date; values below this level lead to penalties in downstream production stages. Also, for the selection of the tree having the best t_θ; as mentioned in Algorithm 1; we use $Purity_{min} = 0.8$ and $Coverage_{min} = 0.25$. The purity threshold $Purity_{min} = 0.8$ ensures that the BBMB is reliable enough so that the cost of triggering an unnecessary reschedule remains marginal, while the coverage threshold $Coverage_{min} = 0.25$ guarantees that the rule captures a non-negligible fraction of *Bad* scenarios. These values were also empirically validated on preliminary instances as the best compromise between detection reliability and earliness of t_θ.

A. Analysis on training instances

Learning $condition_\theta$ splits Ξ_{train} into $\Xi_{\text{classified\_bad\_for\_}S^0}$ and $\Xi_{\text{classified\_good\_for\_}S^0}$. We assess whether the rescheduling effectively transforms initially *Bad* scenarios for S^0 into *Good* ones for S^1, by computing the following conversion rate:

$$\text{Rate}_{bad\_for\_S^0 \to good\_for\_S^1} = \frac{|\{\xi \in \Xi_{\text{classified\_bad\_for\_}S^0} \mid \text{label}_{S^0}(\xi) = \text{Bad},\ \text{label}_{S^1}(\xi) = \text{Good}\}|}{|\Xi_{\text{classified\_bad\_for\_}S^0}|} \tag{14}$$

where $\text{label}_{S^0}(\xi) = \text{sim}(S^0, \xi)$ and $\text{label}_{S^1}(\xi) = \text{sim}(S^1, \xi)$ denote the outcome of simulating respectively S^0 and S^1 on scenario ξ.

Fig. 7 shows the impact of rescheduling across the 13 instances. We can see that the conversion rate varies across instances and exceeds the 50% threshold on 5 out of 13 instances (about 38.5%), with several other instances close to this threshold; this confirms that the decision tree scheduling manages, on a meaningful share of cases, to turn initially *Bad* scenarios for S^0 into *Good* ones for S^1.

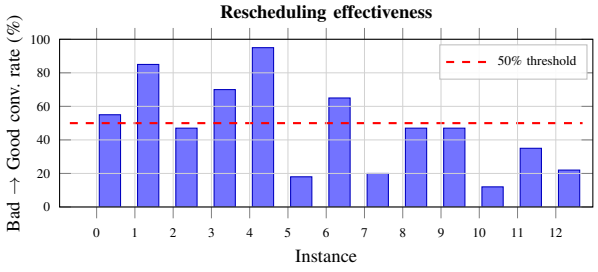


Fig. 7. *Bad_for_*$S^0 \to$ *Good_for_*S^1 conversion rate per instance, across 13 training instances.

Fig. 8 reports the distribution of three key metrics of the learned classification tree across all instances: the conversion rate, the rescheduling checkpoint t_θ and the quality of the obtained tree classifier (purity and coverage). The conversion rate has a mean of 47.6% with high variance, reflecting instance-dependent difficulty. The median rescheduling time is $t_\theta = 6$, which is reasonably early in the horizon $[0, T]$, showing that our dichotomy-based reduction effectively avoids late checkpoints. Purity remains consistently high (median 100%), confirming the reliability of the learned classification rule.

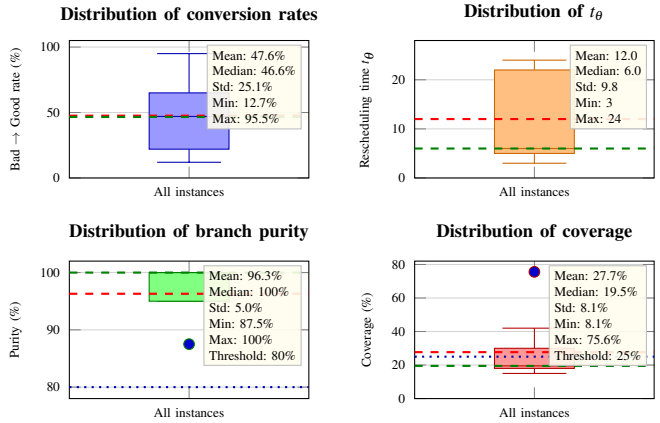


Fig. 8. Distribution of conversion rate (top left), rescheduling time t_θ (top right), and (purity,coverage) of the tree (down left and right) across 13 training instances.

Fig. 9 illustrates the extent to which our approach is able to find a compromise between providing good quality of *bad* scenarios detection, and to react as quickly as possible. We see that most training instances give classifiers with high purity and coverage, showing their ability to detect as well as possible the *bad* scenarios. Moreover, points colors show small learned checkpoint times t_θ, meaning that our Algorithm 1 finds a good trade-off between early anticipation and good detection of bad situations.

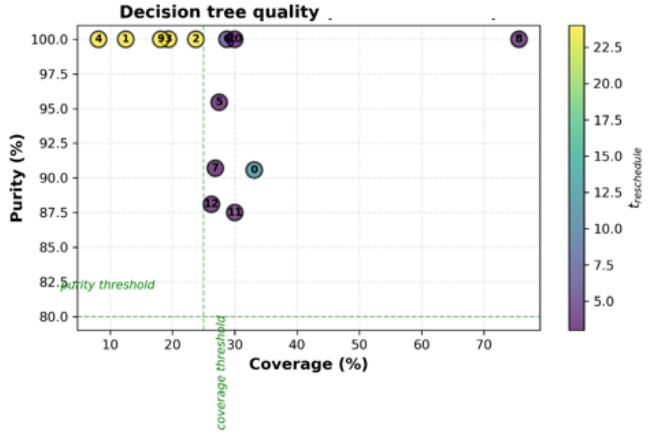


Fig. 9. This figure represents for each training instance 3 key metrics : (purity, coverage) of the tree classifier which shows its quality detection of *bad* scenarios; and learned checkpoint t_θ which shows its ability to react early.

B. Analysis on a test set instance

To evaluate the impact of our rescheduling approach on the objective function, we train our dichotomy-based classifier on one training instance to output $(t_\theta, condition_\theta)$, then sample a test instance with another seed and draw 15 random scenarios from it. For each scenario ξ, we simulate consecutively S^0 and S^1 on ξ and compute the sum of job tardinesses for each schedule ($P(S^0(\xi), \xi)$ and $P(S^1(\xi), \xi)$, Eq. (9)). As shown in Fig. 10, 11 out of 15 scenarios (73%) have a reduced sum of tardinesses after rescheduling, with an average gain $P(S^0(\xi), \xi) - P(S^1(\xi), \xi) = 2.6$. Overall, our approach improves scheduling performance.

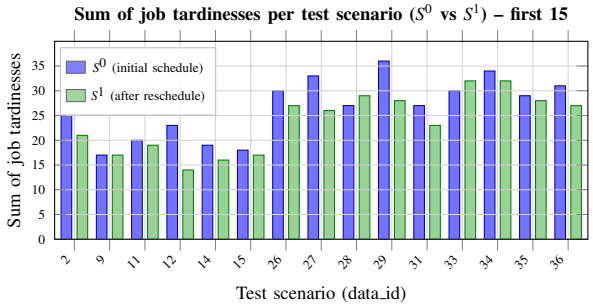


Fig. 10. job tardinesses of S^0 and S^1 on 15 test set scenarios

V. CONCLUSIONS

We proposed a proactive–reactive approach based on a two-node scheduling decision tree that learns a checkpoint time t_θ and a rescheduling rule $condition_\theta$. Applied to uniform parallel machines with uncertain line-speed evolution, it improves performance on training and test instances by converting scenarios initially *Bad* for S^0 into *Good* for S^1, while balancing detection reliability and early reaction. Future work will extend experiments to the Schaeffler use case and study sensitivity to the labeling threshold and other parameters (e.g., number of lines and tasks). The approach also generalizes to trees with multiple nodes: after computing S^1, the constructive step is repeated to create *Good*/*Bad* branches, and ongoing scenarios switch from a schedule to its update when conditions are met.

REFERENCES

[1] J. R. Birge and F. Louveaux, *Introduction to Stochastic Programming*, 2nd ed. New York, NY: Springer, 2011.

[2] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338–353, 1965.

[3] D. Dubois and H. Prade, *Possibility Theory: An Approach to Computerized Processing of Uncertainty*. New York, NY: Plenum Press, 1988.

[4] D. Bertsimas and M. Sim, "The price of robustness," *Operations Research*, vol. 52, no. 1, pp. 35–53, 2004.

[5] G. E. Vieira, J. W. Herrmann, and E. Lin, "Rescheduling manufacturing systems: A framework of strategies, policies, and methods," *Journal of Scheduling*, vol. 6, pp. 39–62, 2003.

[6] W. Herroelen and R. Leus, "Project scheduling under uncertainty: Survey and research potentials," *European Journal of Operational Research*, vol. 165, no. 2, pp. 289–306, 2005.

[7] T. Portoleau, C. Artigues, and R. Guillaume, "Robust decision trees for the multi-mode project scheduling problem with a resource investment objective and uncertain activity duration," *European Journal of Operational Research*, vol. 312, 2024.