Robust Predictive-Reactive Scheduling : an Information-Based Decision Tree Model

Tom Portoleau, Christian Artigues, Romain Guillaume

Robust Predictive-Reactive Scheduling : an Information-Based Decision Tree Model

Tom Portoleau[1,2], Christian Artigues[1], Romain Guillaume[2]

[1] LAAS-CNRS, Université de Toulouse, CNRS, France,
{tom.portoleau,christian.artigues}@laas.fr
[2] ANITI, IRIT-CNRS, Université de Toulouse, France,
romain.guillaume@irit.fr

Abstract. In this paper we introduce a proactive-reactive approach to deal with uncertain scheduling problems. The method constructs a robust decision tree for a decision maker that is reusable as long as the problem parameters remain in the uncertainty set. At each node of the tree we assume that the scheduler has access to some knowledge about the ongoing scenario, reducing the level of uncertainty and allowing the computation of less conservative solutions with robustness guarantees. However, obtaining information on the uncertain parameters can be costly and frequent rescheduling can be disturbing. We first formally define the robust decision tree and the information refining concepts in the context of uncertainty scenarios. Then we propose algorithms to build such a tree. Finally, focusing on a simple single machine scheduling problem, we provide experimental comparisons highlighting the potential of the decision tree approach compared with reactive algorithms for obtaining more robust solutions with fewer information updates and schedule changes.

1 Introduction

Dealing with uncertainty in scheduling problems is an issue that has been widely studied over the last decade. Many approaches emerged in order to cope with uncertainties. Proactive methods elaborate an initial baseline schedule while taking into account possible incoming breaks to make it as robust as possible. There exist several robustness criteria [7] that are widely used in these methods. However, one major flaw of proactive scheduling methods is their conservatism [9], the more robust solutions tend to be low-quality objective-wise. It is particularly the case when uncertainties are large and frequent. In these cases, reactive methods are more appropriate as they do not specially focus on the initial schedule but rather on how to modify it online, that is to say during the execution. They are usually based on priority rules [10] that allow decision maker to compute quickly new solutions according to the running scenario. However, on the contrary of proactive approaches, these is no guarantee regarding the objective value of solutions computed with a reactive method. In order to provide solutions that are well balanced between robustness and quality, many hybrid

methods have been suggested. The recoverable robustness framework [1] considers two stage decisions where robust decisions are taken at the first stage and where recovery algorithms are used to restore feasibility for a realised scenario on the second stage. This framework is linked to adjustable robust optimisation [12] that roughly transposes the concept of two-stage stochastic programming to robust optimisation: some recourse variables can be adjusted to the realised scenario.

In the scheduling literature, approaches that mix a proactive phase aiming at issuing a robust baseline schedule and a reactive phase that adapts the baseline scheduling in case of major disturbances have been widely studied under the proactive-reactive scheduling terminology [11]. Recently [3] remarked that in this series of approaches, the proactive and the reactive phases were rather treated separately while they should mutually influence each other. So in [3, 4] the authors propose an integrated proactive reactive approach where they aim to find the best policy, which is in their case a robust initial schedule and a set of reactions giving transitions from a schedule to another schedule in response to a disruption, given a certain reaction cost. In a pioneering work, [6] proposed the so-called Just In Case approach, in which they compute a multiple contingent schedule, where transitions from a baseline schedule to alternative schedules were anticipated at some events having a high probability of break. This approach has been since then largely developed for AI planning problems, addressing among others incrementality and memory limits issues [5, 8]

In this paper, we are interested in transposing contingency planning concepts to robust scheduling problems. We consider repeated scheduling problems where some parameters are known and constant at each scheduling iteration and some of them are known with imprecision, according to scenarios. In order to take advantage of known and constant parameters, we introduce a proactive reactive method in which we suppose that the decision maker may have access to some information about the imprecise parameters to reduce the uncertainty at predefined time points of the schedule execution, where the schedule can be changed. However, obtaining information on the uncertain parameters can be costly and frequent rescheduling can be disturbing. Hence at each decision time point the scheduler has the choice to use the information or not and to react or not depending on the impact of the reaction on the schedule robustness. Using intelligently these information, we build off-line a decision tree that will be used to schedule the problem at each repetition.

The problem we chose to validate our approach is the simple scheduling problem $1||L_{max}$, in which we suppose that the processing time are known and fixed, and the due dates are uncertain. We detail why we chose this problem in Section 4.

The outline of the paper is the following. Section 2 formally define the uncertainty and information models as well as the robust decision tree concept and its related problem. Then, we detail the algorithm we designed to solve these problems, namely the general algorithm for building the robust decision tree (Section 3) and the algorithm for solving the robust partitioning subproblem (Section 4).

We then present some numerical results in Section 5 comparing our approach to a more standard proactive-reactive scheduling algorithm.

2 Notations and Definitions

2.1 Uncertainty model

In practice, it is often easier for a decision-maker to establish bounds over uncertain parameters, like processing times or due dates, than to build an accurate probabilistic model which often requires a large amount of data. In this paper, we consider interval uncertainty. Given an uncertain parameter x we denote by $X = [x_{min}, x_{max}]$ the interval in which it takes its value. We make no assumption about which probabilistic law is followed by x in this interval. Given a set of uncertain parameters $(x_i)_{i \in I}$ we define the set of possible assignments of parameters by $\Omega = \prod_{i \in I} X_i$ (i.e. it is assumed that there is no correlation between them, all realisation x_i are independent). We call a scenario an assignment of each parameter $x_i, \forall i \in I$ such that $(x_i)_{i \in I} \in \Omega$.
From now on, when ω is a scenario, s a schedule and f the objective , we denote by $f(\omega, s)$ the objective value of s in scenario ω (note that for our application case, $f = L_{max}$).
In this paper we consider the $1||L_{max}$ problem, that is to say we want to schedule a set of I jobs with deterministic processing times $p_i, i \leq I$ and uncertain due dates $d_i \in D_i, i \leq I$ on a single machine so that the maximum lateness is minimum.

Example 1. *We consider a small instance of the problem* $1||L_{max}$ *with three tasks :*

- *task 1 :* $p_1 = 10$, $d_1 \in D_1 = [10, 11]$
- *task 2 :* $p_2 = 6$, $d_2 \in D_2 = [11, 17]$
- *task 3 :* $p_3 = 4$, $d_3 \in D_3 = [13, 20]$

The set of scenarios for this instance is $\Omega = D_1 \times D_2 \times D_3$ *and* $\omega = (10, 12, 19)$ *is a scenario. We will keep this instance all along this paper to exemplify the notions and algorithms we introduce.*

2.2 Information Model

As explained in Section 1 we suppose that at some time during the execution of the schedule, some information become accessible. In our model, an information allows the scheduler to tighten the interval of uncertainty of a future realisation.

Definition 1. *For a given uncertain parameter* $x \in X$ *and a moment of decision* t *during the execution of the schedule, we call an information about* x *a value* k_x^t *and an operator in* $\{\leq, \geq\}$.

For instance, an information is $x \leq k_x^t$. So the decision maker, from time t on, is able to reduce the set of possible assignments by updating the interval X, $x \in X_{k_x^t}^{inf} = [x_{min}, k_x^t]$ or $x \in X_{k_x^t}^{sup} =]k_x^t, x_{max}]$. Note that the information depends on t, so the scheduler may have to ask for an information about the same data several times during the execution of the planning. Our model aims to make the best use of available information, and select the more relevant ones. For the problem considered in this paper we suppose that for a given moment of decision t and a task i, we have an information k_d^t if $\min(t + p_i, 2t) \in D_i$. If so, $k_d^t = \min(t + p_i, 2t)$. Otherwise we consider that we have no information about the task i . This hypothesis on the availability of information is arbitrary, but in fact it expresses two natural questions a scheduler may ask about uncertain due dates : "If task i started now, can it be completed without being late ? If so, is the due date d_i far from now ?" In any case, an answer to these questions allows the scheduler to bound the uncertainty of a due date d_i.

Example 2. *Let us look again at the instance from Example 1. We suppose that we are at a moment of decision $t = 10$ and that task 1 has been scheduled first. Given the hypothesis we made about information availability, we have:*

- *task 1 is completed, so there is no relevant information about it.*
- $\min(t + p_2, 2t) = 16 \in D_2$, *so* $k_{d_2}^t = 16$.
- $\min(t + p_3, 2t) = 14 \in D_3$, *so* $k_{d_3}^t = 14$.

Therefore, for any $\omega \in \Omega$, the scheduler is able to determine, from time $t = 10$, if $\omega_2 \leq 16$ or if $\omega_2 > 16$, and if $\omega_3 \leq 14$ or if $\omega_3 > 14$.

2.3 Robust Decision Tree

Definition 2. *A robust decision tree T is a tree where the nodes are labeled with a subset of Ω, and the arcs are labeled with a partial schedule. If n is a node of T, Ω^n denotes a subset of Ω associated to n. A robust decision tree satisfies the following properties:*

(i) *Let us denote by $(n_j)_{j \leq J}$ the children of node n. Then $\bigcup_{j \leq J} \Omega^{n_j} = \Omega^n$.*
(ii) *For any path $(n_0, ..., n_m)$ where n_0 is the root of T and n_m is a leaf, the schedule obtained by concatenating all the partial schedules on the arcs along the path is feasible.*
(iii) *Let n and n' be two nodes of T. The partial schedule s' on the arc (n, n') is robust:*

$$s' = \underset{s \in S}{\operatorname{argmin}} \max_{\omega \in \Omega^{n'}} f(\omega, s)$$

where S is the set of admissible partial solutions.

A robust decision tree can be seen as a compact representation of a set of solutions. Given that the decision maker has access to information that allow to split the set of scenarios, the tree makes it possible to retrieve a solution, with a robustness guarantee, for certain subsets of scenario. A generic illustration of a

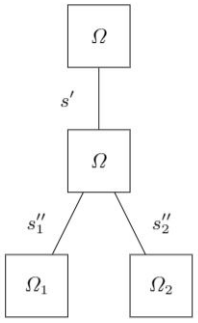


Fig. 1. Generic example of a Robust Decision Tree.

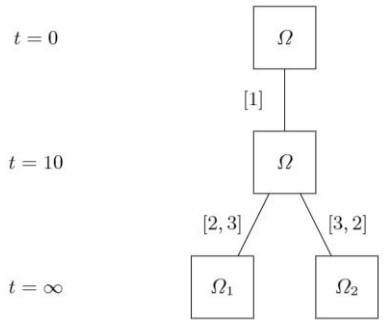


Fig. 2. Example of a robust decision for Example 3.

robust decision tree is displayed in Figure 1. In this case, if the decision maker is able to know if an ongoing scenario ω is in Ω_1 or in Ω_2, he can pick the more robust solution for each case.

In this paper our goal is, for a given set of scenario, to compute a robust decision tree that is actually usable by a decision maker during the execution of the schedule, allowing to adapt the schedule online and to make it as robust as possible, depending on some knowledge or information the decision maker has access to. For this purpose, we consider in our model that each level j in the tree coincide with a fixed moment of time t_j during the progress of the schedule. These moments are the points in time when the decision maker has access to some knowledge about uncertain parameters. An illustration of this is given in Example 3.

Example 3. *Still using data from Example 1, if we consider one moment of decision $t = 10$ we can use the information computed in Example 2. For this example, we set $\Omega_1 = D_1 \times [11, 16] \times D_3$ and $\Omega_2 = \Omega \setminus \Omega_1$. Then the robust decision tree shown in Figure 2 is valid. The first task to be scheduled is task 1, regardless of the ongoing scenario, then thanks to the information accessible at time $t = 10$, the decision maker knows if the ongoing scenario ω is in Ω_1 or in Ω_2, and then can switch to the more robust solution (respectively scheduling task 2 before task 3 if $\omega \in \Omega_1$ and task 3 before task 2 otherwise).*

2.4 Partitioning the Scenarios

The core problem of our method is, for any node n, computing a robust partition of the scenario set, but how do one compare the robustness of two different partitions ? We propose the following criterion. We define the Robustness Score (RS) of a partition P as the sorted vector of the worst case objective values of the optimal robust solution (considering absolute robustness criterion [7]) on

each element of P :

$$RS(P) = (\min_{s \in S} \max_{\omega \in p} f(\omega, s))_{p \in P}$$

where S is the set of feasible solutions. Now, given two partitions P and P', we say that P is a better partition than P' if $RS(P) <_{lexi} RS(P')$ where $<_{lexi}$ is the lexicographical order. The intuition behind this criterion is the following: each value of this vector is the objective value of the solution that minimises its objective value on the worst case scenario of every subset making up the partition. Since these vectors are sorted in the increasing order, the first value is the minimum minmax objective value. In other words, the subset of scenarios corresponding to this value has the best worst-case objective value. By comparing this value first we know which partition allows the scheduler to improve the robustness of the solution the most.

Example 4. *Let us consider the partition P of $\Omega = \Omega_1 \cup \Omega_2$ from Example 3. Let us denote by $RV(\Omega)$ the minmax objective value on Ω, or more formally $RV(\Omega) = \min_{s \in S} \max_{\omega \in \Omega} f(\omega, s)$. We have:*
$RS(P) = (\min(RV(\Omega_1), RV(\Omega_2)), \max(RV(\Omega_1), RV(\Omega_2)))$
$RS(P) = (6, 7)$
Let now be P' another partition of $\Omega = \Omega'_1 \cup \Omega'_2$, with $\Omega'_1 = D_1 \times D_2 \times [13, 14]$ and $\Omega_2 = \Omega \setminus \Omega_1$. We compute $RS(P') = (4, 7)$. Since $RS(P') <_{lexi} RS(P)$, P' is a better partition considering our criterion.

As we have seen, an information k_x^t allows us to split in two the set of scenarios, since it enables us to distinguish scenarios where the data $x \in X^{inf}_{k_x^t}$ from those where $x \in X^{sup}_{k_x^t}$. More generally, if m information are available, we can split the set of scenarios in 2^m subsets. We denote by K^t the set of all information available at time t. Our goal is to use these subsets to create a size-limited partition of the set of scenarios and compute for each subset of scenarios within the partition a new robust solution. The idea is that diminishing the size of the set of scenarios necessarily improves the worst-case scenario and thus leads to better robust solutions. The maximum size of this new partition is a parameter L, decided by the decision maker. Moreover, the maximum number of information the partition is allowed to use is another parameter Q.

We express the core problem (our method involves solving it multiple times) of our approach, the Robust Partitioning Problem (RPP):

ROBUST PARTITIONING PROBLEM
INSTANCE: A set of scenarios Ω of dimension I, a set of information $K^t$and two integers Q and $L \leq 2^Q$.
SOLUTION: A partition P of Ω that verifies:

1- $|P| \leq L$
2- $\forall p \in P, \exists J_p \in \mathbb{N}$, $p = \bigcup_{j \leq J_p} \prod_{i \leq I} p_{i,j}$ with $p_{i,j} \in \{X^{inf}_{k^t_{x_i}}, X^{sup}_{k^t_{x_i}}, X_i\}$ if $k^t_{x_i} \in K^t$ and $p_{i,j} = X_i$ otherwise
3- $|\{k^t_{x_i} \in K^t | \exists p \in P, \exists j \leq J_p, \exists i \leq I, p_i \in \{X^{inf}_{k^t_{x_i}}, X^{sup}_{k^t_{x_i}}\}| \leq Q$

MEASURE: $RS(P)$ minimal for the lexicographical order.

Constraint 1 limits the size of the solution partition so it must be lower than L. Constraint 2 forces the partition to be composed of subsets formed by the information from K^t. In other words, a partition such that any of its subset cuts through the hyperplane defined by $\omega_i = k^t_{x_i}$ is not a feasible solution. And, finally, constraint 3 forces the maximum number of information to be lower than Q. Note that in the solution if we have $p_{i,j} = X_i$ for all $p_{j,i}$ and $k^t_{x_i} \in K^t$, for a parameter i, this means that despite the availability of a new information on parameter i at time t, this information has been ignored as the uncertainty set X_i is not partitioned according to $k^t_{x_i}$. This can be due to the limit on the number Q of information that can be used (Constraint 3) or to the absence of positive impact on this partition on the lexmin objective (i.e. the information is not locally relevant to improve the robustness).

Now that our model is set, the next sections introduce the algorithms we implemented to produce a robust decision tree and solve the RPP.

3 Robust Decision Tree Algorithm

In this section, we detail the general algorithm to build a robust decision tree.

Using the previous definitions we now propose a method to build a robust decision tree (see Definition 2). In this paper, we consider that the moments of decision (i.e. moments when the scheduler is able to access new information and change the schedule), denoted by $(t_j)_{j \in J}$ are fixed in advance. This may correspond in practice to special times, such as the end of a working day, or a shift change where the planning can be updated. Every decision moment corresponds to a level in the decision tree, such that t_1 corresponds to the first level, t_2 to the second one, etc... In that respect, the depth of the tree is controlled by the number of decision moments. At each fork at a level j in the tree, a new partial solution, consistent with the partial schedule that has been accomplished until t_j, is proposed according to the current set of scenarios. The root of the tree, that we consider being the level 0 corresponds to the time $t_0 = 0$, the beginning of the schedule. At this point no information is known, so only one robust solution

is proposed. Thus, a single node is created at level 1. At this node, we retrieve all the information available at time t_1. Using up to Q information, we split the set of scenarios into -at most 2^Q- subsets forming a partition P. We then solve the Robust Partition Problem (we detail more about this solution procedure in Section 4), and obtain a robust partition P'. For each subsets in P' a new solution is proposed and a new branch is set up, leading to a new node at the next level. The set of scenarios considered in this node is the one from which it originated in P'. These steps are repeated until the last decision moment is reached.

4 Robust partition algorithm

In the general case, one can clearly see that this problem is highly combinatorial. Indeed we have to, for each combination of information, compute the best partition using these information. The complexity of the RPP depends on three factors. The first two are the number of tasks to schedule (let say n), since it gives an upper bound of the number of information available at each moment of decision, and Q the number of information we are allowed to use at each moment of decision. The number of possible combinations at a moment of decision is bounded by $\sum_{q=1}^{Q} \binom{n}{q}$. The third factor is the complexity of computing the min-max robust objective value $\min_{s\in S} \max_{\omega\in\Omega} f_\omega(s)$ for any Ω. Clearly, if the deterministic problem is already difficult, so is its robust variant. However, there are cases where a deterministic scheduling problem is polynomial, while its uncertain alternative is NP-Hard. We chose the problem $1||L_{max}$ to test our approach specifically because its robust min-max alternative is still polynomial [2].

Proposition 1. *If f admits a global worst case scenario, that is to say that there exists a scenario ω^{wc} such that:*

$$\forall \omega' \in \Omega, \forall s \in S, f(\omega^{wc}, s) \geq f(\omega', s)$$

then given a partition P of Ω and an integer L, it is possible to compute in polynomial time a partition P' so that $RS(P')$ is minimal and that satisfies:

(i) $|P'| = \min(L, |P|)$
(ii) for all $p \in P$ *there exists* $p' \in P'$ *such that* $p \subset p'$

Remark : For the $1||L_{max}$ problem, the global worst case scenario is $\omega = (d_{i_{min}})_{i\in I}$.

Proof. We prove the proposition by showing that any partition returned by Algorithm 1 verifies the properties from Proposition 1.
By construction, a partition P' returned by Algorithm 1 satisfies (i) and (ii). Now we must prove that $RS(P')$ is minimal. First we can observe that:

$$\min_{s\in S} \max_{\omega\in\bigcup_{j\in J} P_j} f(\omega, s) = \min_{s\in S} \max_{j\in J} \max_{\omega\in P_j} f(\omega, s)$$

for any family of disjoint sets $(P_j)_{j\in J}$. From that observation we can derive that for any partition P'' that satisfies (ii), the values contained in $RS(P'')$ are necessarily in $RS(P)$. Thus for any other partition P'' that verifies (ii) we have, for $i \leq L-1$,

$$RS(P')_i \leq RS(P'')_i \tag{1}$$

because, by construction, $RS(P')_i$ is the i-th smallest possible value and RS vectors are sorted in the increasing order. In addition, since f admits a global worst case scenario ω^{wc}, there exists $p' \in P$ such that $\omega^{wc} \in p'$. Thus, for any union of subset of the form $\bigcup_{p\in P} p$ we have:

$$\min_{s\in S} \max_{\omega \in \bigcup_{p\in P} p} f(\omega, s) = \min_{s\in S} \max_{\omega\in p'} f(\omega, s) = \min_{s\in S} f(\omega^{wc}, s)$$

Thereby, for any partition P'' the last value in $RS(P'')$ is necessarily $\min_{s\in S} f(\omega^{wc}, s)$. Finally, from this result and (1) we conclude that $RS(P')$ is minimal. □ □

An example of the execution of Algorithm 1 is given in Example 5.

Algorithm 1

Require: $P = [P_1, P_2, ..., P_j]$ a partition of Ω and an integer L.
  for $i \leq j$ **do**
    $RS[i] \leftarrow \min_{s\in S} \max_{\omega\in p_i} f(\omega, s)$
  end for
  for $i \leq j$ **do**
    $RS_S[i] \leftarrow RS[\phi(i)]$
    $P_S[i] \leftarrow P[\phi(i)]$
    where ϕ is the permutation that sorts RS in the increasing order.
  end for
  $j' \leftarrow$ the index such that the global worst case scenario ω^{wc} is in $P[j']$
  $P_S[j], P_S[j'] \leftarrow P_s[j'], P_s[j]$
  if $|P_S| > L$ **then**
    $P_S \leftarrow [P_S[1], P_S[2], ..., P_S[L-1], \bigcup_{i=M}^{j} P_S[i]]$
  end if
  return P_S

Example 5. *Once again, let us consider instance from Example 1. Let us suppose that $t = 10$ is a moment of decision and we have to develop a node from the tree. In Example 2, we saw that $K^{t=10} = \{k_{d_2}^{t=10}, k_{d_3}^{t=10}\}$ with $k_{d_2}^{t=10} = 16$ and $k_{d_3}^{t=10} = 14$. We can split the set of scenarios into four subsets (as shown in Figure 3).*
Let us apply Algorithm 1 with $P = [A, B, C, D]$ and $L = 3$. Keeping the same notation, we have $RSV = [7, 6, 4, 4]$, then $RSV_s = [4, 4, 6, 7]$ and $P_s = [C, D, B, A]$. As $|P_s| > L$, we modify P_s to $P_s = [C, D, A \cup B]$ and its robustness score is $[4, 4, 7]$.

Remark : Note that by definition of the shape of P any union of rectangles is feasible. In other words, even a non rectangle shape is acceptable. For instance, considering notations from Example 5, the partition $[A, B \cup C \cup D]$ is a feasible solution.

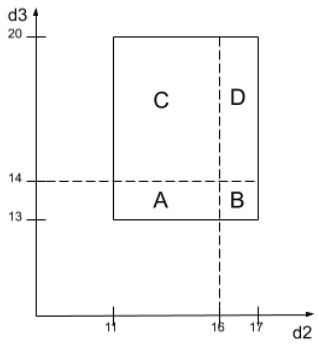


Fig. 3. Set of scenarios for Example 5

We now propose Algorithm 2 to solve the RPP problem. It is an exhaustive algorithm, that calls Algorithm 1 for each combination (smaller than Q) of information, so it has an exponential complexity.

We are now able to build a robust decision tree with the procedure introduced in Section 3.

Algorithm 2

Require: A set of scenarios Ω, a set of information K^t and two integers Q and L
  $P^* = [\Omega]$
  for $K \subseteq K^t$ **do**
    if $|K| \leq Q$ **then**
      $P = \{p | p = \bigcup_{j \in J_p} \prod_{i \in I} p_{i,j}$ with $p_{i,j} \in \{X^{inf}_{k^t_{x_i}}, X^{sup}_{k^t_{x_i}}, X_i\}$ if $k^t_{x_i} \in K^t$ and $p_{i,j} = X_i$ otherwise $\}$
      $P' = Algo1(P, L)$
      if $RS(P') \leq_{lexi} RS(P^*)$ **then**
        $P^* = P'$
      end if
    end if
  end for
  return P^*

5 Experimentations

The objective of the carried out experiments is to evaluate the robustness of our robust decision tree model, the quality of the selected information used for its construction and its stability in terms of number of reactions. For the numerical tests, we generated different types of instances for the $1||L_{max}$ problem with uncertainty on the due dates according to two parameters. The first parameter is the class of the instance. We distinguish two classes: the A class with small uncertainty intervals, and the B class with large uncertainty intervals. The second parameter is the number of tasks to schedule. For example, the instance A_{10} is an instance with 10 tasks, with small uncertainty intervals on the tasks' due dates.

As our approach uses the notion of information to reduce uncertainties to provide more robust solution, we compare it to a more standard proactive-reactive algorithms. This algorithm takes two parameters as input (in addition to the instance), a reaction rate $\rho_r \in [0, 1]$ and an information rate $\rho_i \in [0, 1]$. The principle of the algorithm is the following. We start the execution of the planning with a robust schedule in the sense of the min max criterion. At the end of each task, the algorithm reacts with a ρ_r probability. When it reacts, it computes a new robust solution using at most $100\rho_i\%$ of the available information. The definition of an information and the way it is accessible are strictly the same as the ones used to build a robust decision tree. To build a robust decision tree with our method, we need a couple of parameters : a list of moments of decision $(t_j)_{j\in J}$, the maximum number of information we are allowed to use at each moment of decision Q, and the maximum size of the partition we compute at each moment of decision L. In order to test different lists of moment of decision, we split the total schedule duration in T equal intervals. In our experiment we computed robust decision trees using the following values:

- Number of moment of decision $T \in [2, .., 10]$.
- Maximum number of information at each moment $Q \in [1, .., 10]$.
- Maximum size of partitions at each moment $L \in [2, .., 4]$.

As such, a tree computed with these parameter is denoted by $\text{Tree}_{Q,L,T}$. These values may seem small, but as we discuss it earlier, these parameters not only impact the computation time to solve the RPP, but also the size of the tree. In order to keep the total computation time of a tree reasonable, we had to keep these values not too high.

As we have seen before, the proactive-reactive algorithm takes two parameters, ρ_i and ρ_r. Since an execution of this algorithm is fast to simulate, we enumerate for ρ_i and ρ_r every values in [0, 1] with 0.1 steps.

For each instance, we randomly pick 500 scenarios (with an uniform distribution). Then, for each set of parameters and each instance, we compute a robust decision tree, and go through the tree following the path corresponding to each random scenario. The same protocol is used with the robust reactive algorithms, we simulate them on each random scenarios.

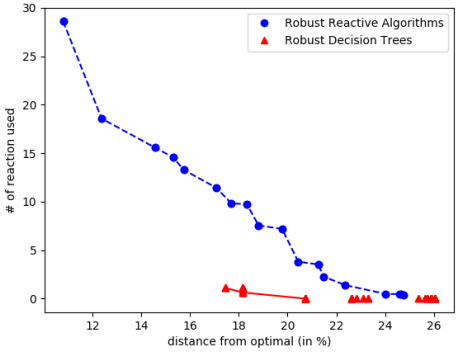


Fig. 4. Pareto frontier for both criterion on instance A_{50}

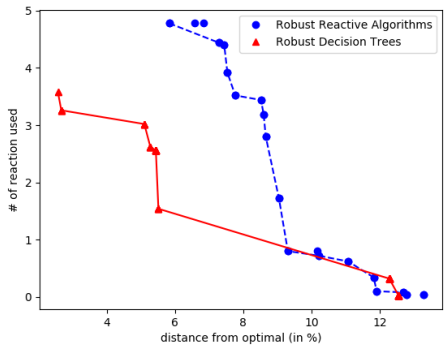
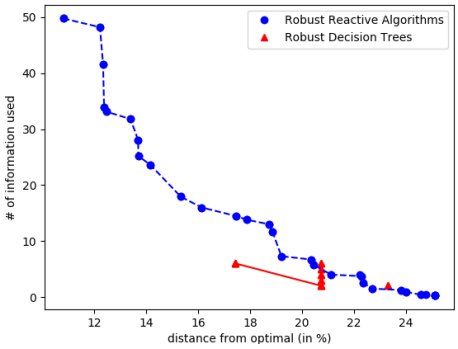
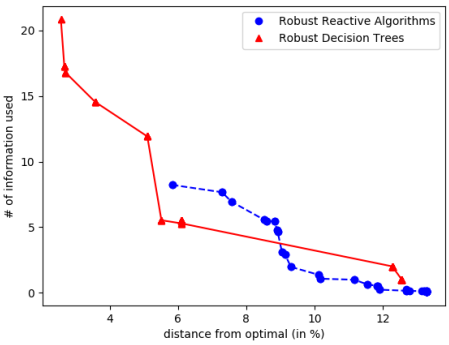


Fig. 5. Pareto frontier for both criterion on instance B_{50}





5.1 Information/Reaction Efficiency

We assess the relevance of the information used to build the tree, and the stability of the planning proposed by the tree. To do so we collected, for each instance, the mean number (over the 500 random scenarios) of information used, the mean number of reactions (when a global solution changes between two moments of decisions) and the distance from optimum, which is, for a given scenario ω and a solution s :

$$100 \cdot \frac{L_{max}(\omega, s) - L_{max}(\omega, s^*)}{L_{max}(\omega, s^*)}$$

where s^* is the optimal solution minimising its L_{max} value on scenario ω.
We consider two couples of criterion: (distance from optimum, number of reactions) and (distance from optimum, number of information), and for both of them, we draw two Pareto frontiers, one for the robust reactive algorithm and one for the robust decision tree. Some of the results are shown in Figure 4 and 5. Due to the fact that most trees could not be computed before the time limit, there are less point for the decision trees for instance A_{50}.

For the instance A_{10}, the best robust decision trees produce worse solutions than the best reactive robust algorithms when little numbers of reactions and information are used, but it performs better with more reactions and information. On instances B_{10} A_{50}, the best decision trees perform better than the best reactive algorithms on both criterion. More generally, we observe that the robust decision trees provide better solutions (for a given number of tasks) when uncertainty intervals are larger. Intuitively, this can be explained by the fact that decision trees are very constrained by the moments of decision while the reactive algorithms is not. So, larger uncertainties allow robust decision trees to acquire more new information than it does with robust reactive algorithms.

	Reaction		Information	
	extreme 1	extreme 2	extreme 1	extreme 2
A_{10}	$\text{Tree}_{10,4,2}$	$\text{Tree}_{3,2,14}$	$\text{Tree}_{10,4,2}$	$\text{Tree}_{3,2,14}$
B_{10}	$\text{Tree}_{10,4,2}$	$\text{Tree}_{10,4,6}$	$\text{Tree}_{10,4,2}$	$\text{Tree}_{2,4,6}$
A_{50}	$\text{Tree}_{10,4,3}$	$\text{Tree}_{3,4,6}$	$\text{Tree}_{2,4,3}$	$\text{Tree}_{2,4,6}$

Table 1. Robust decision trees corresponding to extreme point in the Pareto frontiers shown in Figures 4 and 5.

However, this way of presenting the results does not show which robust decision trees appear in the Pareto frontier. Table 1 shows which trees (and the set of parameters they were computed with) are the extreme points of the different Pareto frontiers. Quite intuitively, the decision trees propose the best solutions with few information and reactions, when the number of moments of decision is low and the maximum number of usable information is high. With the same idea, the decision trees built with a high number of moments of decision (i.e the deepest ones) but with few information available at each moment yield good solutions as well. Interestingly, this shows that the depth of the tree is more important to produce quality solutions than the maximum number of information. The table also shows that the maximum partition size was used for the best trees, which implies the results could be improved by increasing this value.

6 Conclusion

In this paper we introduce a proactive-reactive approach to deal with uncertain scheduling problems. The method constructs a robust decision tree for a decision maker that is reusable as long as the problem parameters belong to the uncertainty set. At each node of the tree we assume that the scheduler has access to some knowledge about the ongoing scenario, reducing the level of uncertainty and allowing the computation of less conservative solutions with robustness guarantees. However, obtaining information on the uncertain parameters can be costly and frequent rescheduling can be disturbing. We first formally define the robust

decision tree and the information refining concepts in the context of uncertainty scenarios. We then introduce the Robust Partition Problem, the core problem of our approach. Then we propose algorithms to solve this problem and build such a tree. Finally, focusing on a simple single machine scheduling problem, we provide experimental comparisons highlighting the potential of the decision tree approach compared with reactive algorithms for obtaining more robust solutions with fewer information updates and schedule changes.
In the view of the encouraging results we believe that this method could be used in industrial cases. For our future works, we plan to apply and extend our method to hard problems, such as the Resource-Constrained Project Scheduling Problem.

References

1. Marjan van den Akker, Han Hoogeveen, J.S.: Combining two-stage stochastic programming and recoverable robustness to minimize the number of late jobs in the case of uncertain processing times. J. Scheduling **21**(6), 607–617 (2018)
2. Aloulou, M.A., Della Croce, F.: Complexity of single machine scheduling problems under scenario-based uncertainty. Operations Research Letters **36**(3), 338–342 (2008)
3. Davari, M., Demeulemeester, E.: The proactive and reactive resource-constrained project scheduling problem. Journal of Scheduling pp. 1–27 (2017)
4. Davari, M., Demeulemeester, E.: Important classes of reactions for the proactive and reactive resource-constrained project scheduling problem. Annals of Operations Research **274**(1-2), 187–210 (2019)
5. Dearden, R., Meuleau, N., Ramakrishnan, S., Smith, D.E., Washington, R.: Incremental contingency planning. In: ICAPS-03 Workshop on Planning under Uncertainty (2003)
6. Drummond, M., Bresina, J., Swanson, K.: Just-in-case scheduling. In: AAAI. vol. 94, pp. 1098–1104 (1994)
7. Kouvelis, P., Yu, G.: Robust Discrete Optimization and Its Applications. Kluwer Academic Publishers (1997)
8. Meuleau, N., Smith, D.E.: Optimal limited contingency planning. In: Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence. pp. 417–426. Morgan Kaufmann Publishers Inc. (2002)
9. Nikulin, Y.: Robustness in combinatorial optimization and scheduling theory: An extended annotated bibliography. Tech. rep., Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel (2006)
10. Rajendran, C., Holthaus, O.: A comparative study of dispatching rules in dynamic flowshops and jobshops. European Journal of Operational Research **116**(1), 156 – 170 (1999). https://doi.org/https://doi.org/10.1016/S0377-2217(98)00023-X, http://www.sciencedirect.com/science/article/pii/S037722179800023X
11. Van de Vonder, S., Demeulemeester, E., Herroelen, W.: A classification of predictive-reactive project scheduling procedures. Journal of scheduling **10**(3), 195–207 (2007)
12. Yanıkoğlu, İ., Gorissen, B.L., den Hertog, D.: A survey of adjustable robust optimization. European Journal of Operational Research **277**(3), 799–813 (2019)