Scheduling under Uncertainty: A Query-based Approach

Luciana Arantes[1], **Evripidis Bampis**[1], **Alexander Kononov**[2,3], **Manthos Letsios**[1], **Giorgio Lucarelli**[4], **Pierre Sens**[1]
[1] Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, F-75005 Paris, France
[2] Sobolev Institute of Mathematics, Novosibirsk, Russia
[3] Novosibirsk State University, Department of Mechanics and Mathematics, Russia
[4] Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble, France

Abstract

We consider a single machine, a set of unit-time jobs, and a set of unit-time errors. We assume that the time-slot at which each error will occur is not known in advance but, for every error, there exists an uncertainty area during which the error will take place. In order to find if the error occurs in a specific time-slot, it is necessary to issue a query to it. In this work, we study two problems: (i) the error-query scheduling problem, whose aim is to reveal enough error-free slots with the minimum number of queries, and (ii) the lexicographic error-query scheduling problem where we seek the earliest error-free slots with the minimum number of queries. We consider both the off-line and the on-line versions of the above problems. In the former, the whole instance and its characteristics are known in advance and we give a polynomial-time algorithm for the error-query scheduling problem. In the latter, the adversary has the power to decide, in an on-line way, the time-slot of appearance for each error. We propose then both lower bounds and algorithms whose competitive ratios asymptotically match these lower bounds.

1 Introduction

Scheduling problems under uncertainty have been studied in different contexts such as on-line optimization, robust optimization, and stochastic optimization. A common assumption in all these models is that the uncertain information cannot be known before the construction of the schedule. However, in some applications it is possible to know the exact input information at some extra cost [Erlebach and Hoffmann, 2015]. In this paper, we consider scheduling with faults/errors and we introduce a new non-probabilistic model with explorable (query-able) uncertainty. Each unit-time error is characterized by an uncertainty area during which the error will occur, and it is possible to learn the exact slot at which it will appear by issuing a query operation of unit cost. Our goal is to reveal enough error-free slots by querying the smallest number of errors in order to be able to successfully execute n unit-time jobs. Note that a query can be seen as an unsuccessful attempt of execution. In Figure 1 (left), we give a simple instance with 6 errors: error e_1 with uncertainty area $(0, 4]$, error e_2 with $(0, 2]$, and so on.

More formally, assume that the time-line is split into unit-size slots. For simplicity, we refer to the slot $(t-1, t]$, for any $t \geq 1$, by its right limit, i.e., t. We are given a single machine, n unit-time jobs and a set E of k unit-time errors. The execution of each job takes place into a single slot. Each error $e \in E$ is associated with an *uncertainty area* $A_e = (s_e, f_e]$ and its actual *slot of appearance* $t^*_e \in (s_e, f_e]$. We assume that, for any two errors $e, e' \in E$, it holds that $t^*_e \neq t^*_{e'}$. A slot t is called *error-free* if for each error $e \in E$ where $t \subseteq A_e$ it holds that $t \neq t^*_e$. In order to reveal an error-free slot we need to *query* all errors whose uncertainty area includes it. We propose to study the following two problems: (i) *The error-query scheduling problem:* Find n error-free slots by performing the minimum number of queries; (ii) *The lexicographic error-query scheduling problem:* Find the earliest n error-free slots by performing the minimum number of queries.

We consider two versions of the above problems. In the *off-line* version the whole instance and its characteristics are known in advance. In the *on-line* version the set of errors and their uncertainty areas are known to the algorithm, but the adversary can decide in an on-line way the slot of appearance t^*_e for each $e \in E$. In other words, there is an iterative game between the on-line algorithm which queries an error and the adversary which reveals the slot of appearance of this error, and so on. Even-though the off-line version seems of marginal interest, it can be used as a lower bound on the number of queries for the on-line case. As a measure of efficiency for the on-line problems we use the *competitive ratio*. An algorithm is ρ-competitive if it makes at most ρ times the optimum number of queries for any instance of the problem.

In what follows, we define two special families of instances concerning the uncertainty areas of the errors. Both families are widely studied in scheduling and graph theory literature.
Agreeable instances: A set of uncertainty areas is agreeable, if for any two errors e and e' with $s_e \leq s_{e'}$, it holds that $f_e \leq f_{e'}$. In other words, there is a unique ordering of the uncertainty areas of the errors with respect to both their left

and their right limits, i.e., if $s_{e_1} \leq s_{e_2} \ldots \leq s_{e_k}$ then $f_{e_1} \leq f_{e_2} \ldots \leq f_{e_k}$. Note that agreeable instances correspond to *proper interval* graphs.

Laminar instances: A set of uncertainty areas is laminar, if for any two errors e and e' with $s_e \leq s_{e'}$, it holds that either $f_e \leq s_{e'}$ or $f_e \geq f_{e'}$. The laminar instances have a tree-like structure and they can be represented as a forest. In order to see such a structure, consider the following construction. Each error e_ℓ, $1 \leq \ell \leq k$, is assigned to a node v_ℓ. For any two nodes v_ℓ and $v_{\ell'}$ with $A_{e_{\ell'}} \subset A_{e_\ell}$, an edge connects them if there is no other error $e_{\ell''}$ such that $A_{e_{\ell'}} \subset A_{e_{\ell''}} \subset A_{e_\ell}$. In the case of multiple errors with the same uncertainty area, we consider an arbitrary order of them and edges between nodes of any two consecutive errors are added to the tree based on this order. Note that the trees created by this construction are considered to be rooted. Specifically, the root of each tree is the node that corresponds to its error with the largest uncertainty area. Moreover, there is an order among the children of a given node which is determined by the order of the uncertainty areas of the corresponding errors in the time-line. An example of a laminar instance along with its tree representation can be seen at Figure 1.

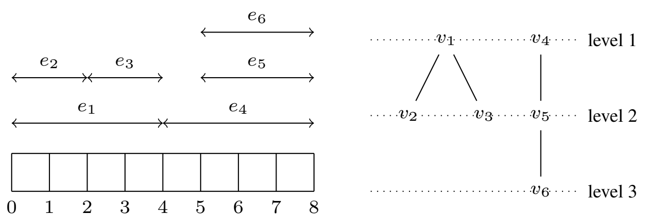

Figure 1: A laminar instance and its representation as a forest.

Related work: The work in the area of explorable uncertainty has been initiated in the seminal paper of Kahan [Kahan,] for selection problems. Since then, a series of papers study many other problems in this context: finding the k-th smallest value in a set of uncertainty intervals [Feder *et al.*, 2003; Gupta *et al.*, 2011; Kahan,], caching problems in distributed databases [Olston and Widom,], the shortest path problem [Feder *et al.*, 2007], the knapsack problem [Goerigk *et al.*, 2015], the minimum spanning tree problem [Hoffmann *et al.*, ; Megow *et al.*, 2015]. In most of these works the aim is to minimize the number of queries to guarantee an exact optimum solution, while some other study the trade-off between the number of queries and the precision of the returned solution [Olston and Widom,]. Recently, explorable uncertainty has been introduced in scheduling [Dürr *et al.*,]. The uncertain information concerns the processing time of each job for which an upper bound is known in advance. It is possible to learn the exact processing time by testing at the price of a unit cost. This line of research has similarities with models that have been extensively studied in AI for computation decision support [Boutilier *et al.*, 2006], especially with models treating adaptive preference elicitation to combinatorial domains [Benabbou and Perny, b; a; Drummond and Boutilier, ; Gelain *et al.*, 2010].

However, our model is also related to *search theory* [Angelopoulos, 2016]. For instance, consider an environment where the search domain is modeled by a star, i.e., consists of a set of rays which have a common intersection point, and there are more than one targets. The goal of the searcher is to locate some of these targets by adopting the *expanding search* paradigm in which the cost of re-exploration is negligible [Alpern and Lidbetter, 2013]. It is not difficult to see that the instance of Figure 2 can be interpreted as a star with b rays and one target (corresponding to one error-free slot) in every ray. The goal is to find some among these targets, while a query corresponds to the exploration of a ray by an additional unit of distance. It has to be pointed out here that star search offers an abstraction for modeling different applications such as interruptible algorithms [Angelopoulos,], or database query optimization [Condon *et al.*, 2009].

Our contribution: We first give an optimal polynomial-time algorithm for the off-line error-query scheduling problem. For the on-line version of the problem, we prove that the competitive ratio of any on-line algorithm is $\Omega(\sqrt{k})$ even if the instance is laminar and agreeable and even if we search for only one free slot. Then, we propose a $O(\sqrt{k})$-competitive algorithm for laminar instances. Finally, in Section 4, we deal with the lexicographic error-query scheduling problem by matching lower and upper bounds of the competitive ratio for the on-line version. Table 1 summarizes our results.

	Error-query scheduling		Lexicographic error-query scheduling	
	Lower Bound	Upper Bound	Lower Bound	Upper Bound
general	$\Omega(\sqrt{k})$	-	$\Omega(\log_2 k)$	$O(\log_2 k)$
agreeable	$\Omega(\sqrt{k})$	-	$\Omega(\log_2 k)$	$O(\log_2 k)$
laminar	$\Omega(\sqrt{k})$	$O(\sqrt{k})$	optimal	optimal

Table 1: Summary of results.

2 Preliminaries

Let's consider the set $\mathcal{T} = \{t_1, t_2, \ldots, t_{|\mathcal{T}|}\}$ of all s_i's and f_i's sorted in increasing order (we eliminate all duplicate values). We define the set $\mathcal{I} = \{I_1, I_2, \ldots, I_{|\mathcal{I}|}\}$ of *elementary intervals* where $|\mathcal{I}| = |\mathcal{T}| - 1$ and each I_i, $1 \leq i \leq |\mathcal{I}|$, spans in $[t_i, t_{i+1}]$. Note that the number of elementary intervals is at most twice the number of errors, i.e., $|\mathcal{I}| = O(k)$. Let $E(I_i)$ be the set of errors whose span includes the elementary interval I_i. Moreover, let $F(E')$ be the set of error-free slots that are revealed if we query the errors of the set $E' \subseteq E$. The elementary interval is an important concept in most of our proofs because of the following property:

Property 1. *Given an elementary interval I_i, all errors in $E(I_i)$ must be queried in order to reveal an error-free slot during I_i. Moreover, if all errors in $E(I_i)$ are queried then all error-free slots of the set $F(E(I_i))$ are revealed.*

Based on the above property, it is meaningless to ask fewer than $|E(I_i)|$ errors of $E(I_i)$ when targeting some error-free slots of I_i. Moreover, there is no reason for not using all error-free slots in $F(E(I_i))$ when all errors in $E(I_i)$ are already queried, unless there is not a sufficient number of unscheduled jobs. Therefore, a set of elementary intervals is also sufficient to represent a solution to our problem. More specifically, given a set of elementary intervals, we need to query all errors whose uncertainty area intersects them. If the number of error-free slots that are revealed is greater than n, then we select arbitrarily n of them in the case of the error-query scheduling problem, or the earliest n of them in the case of the lexicographic error-query scheduling problem.

Finally, we give some additional notation regarding the laminar instances. Given a vertex v_ℓ of depth d_ℓ (i.e., the distance in hops from v_ℓ to the root of its tree is equal to d_ℓ), we define the *level* of the error e_ℓ to be equal to $d_\ell + 1$. Moreover, for any node v_ℓ in the tree-like representation of a laminar instance, let $T(v_\ell)$ be the subtree rooted at v_ℓ. For any (sub)tree T, let $E(T)$ be the set of errors corresponding to the nodes of T. Due to the laminar structure, an algorithm does not have any interest to query an error e_ℓ if it has not already queried all errors corresponding to the nodes in the path from v_ℓ to the root. Given a set E' of already queried errors which follows this observation, we say that a subtree $T(v_\ell)$ is *maximal* with respect to E' if $e_\ell \notin E'$ and either v_ℓ has no ancestors or the errors corresponding to all ancestors of v_ℓ are in E'. For example, for the instance depicted in Figure 1, if $E' = \{e_4\}$ then the subtrees $T(v_1)$ and $T(v_5)$ are maximal.

3 The Error-query Scheduling Problem

In this section, we firstly present a polynomial-time algorithm for the off-line version of the error-query scheduling problem. Then, we consider the on-line version of the problem and we present a lower bound to the competitive ratio as well as a $O(\sqrt{k})$-competitive algorithm for laminar instances.

3.1 A Polynomial-time Algorithm

We present here a polynomial-time algorithm for finding an optimal off-line solution for the error-query scheduling problem. The proposed algorithm uses the representation of a solution to the problem by just indicating a set of elementary intervals, as this is implied by Property 1 and it is explained in preliminaries. Such a representation can be also used for *partial solutions*, i.e., solutions that reveal less than n error-free slots. Moreover, each (partial) solution S corresponds to a triplet (i, j, ℓ), where i is the index of the latest elementary interval used, j is the number of error-free slots that are revealed and ℓ is the number of errors that are queried. Note that a triplet (i, j, ℓ) may correspond to more than one solutions. By slightly abusing the notation, we denote by $E(S)$ the set of errors that are queried by the solution S.

Our algorithm keeps a set of partial solutions and it iterates on the elementary intervals: at iteration i, $1 \le i \le |\mathcal{I}|$, the elementary interval I_i is considered. Let $\mathcal{S}_{i-1}$ be the set of partial solutions at the end of the iteration $i-1$. We extend each $S \in \mathcal{S}_{i-1}$ to the partial solution S' by including to it the elementary interval I_i and we add both S and S' to $\mathcal{S}_i$: if S corresponds to the triplet (i', j, ℓ) with $i' \le i-1$, then $S' = S \cup \{I_i\}$ corresponds to the triplet $(i, j + |F(E(I_i))|, |E(S) \cup E(I_i)|)$, where $\ell \le |E(S) \cup E(I_i)| \le \ell + |E(I_i)|$.

Note that several partial solutions can be dominated by others. We say that a partial solution S_1 corresponding to (i_1, j_1, ℓ_1) is dominated by the partial solution S_2 corresponding to (i_2, j_2, ℓ_2) if $i_1 \le i_2$, $j_1 \le j_2$ and $\ell_1 \ge \ell_2$. At the end of the iteration i we eliminate all dominated partial solutions from $\mathcal{S}_i$. Observe that this elimination phase removes also all but one partial solutions that correspond to the same triplet (i, j, ℓ).

A formal description of the algorithm is given below.

Algorithm 1

```
1: Initialization: $S \leftarrow \emptyset$ (which corresponds to $(0,0,0)$), $\mathcal{S}_0 \leftarrow \{S\}$, $A \leftarrow \bigcup I_i$
2: for $i = 1$ to $|\mathcal{I}|$ do
3:     $\mathcal{S}_i \leftarrow \mathcal{S}_{i-1}$
4:     for each $S \in \mathcal{S}_{i-1}$ corresponding to $(i', j, \ell)$ do
5:         Create a new partial solution $S' \leftarrow S \cup \{I_i\}$ corresponding to $(i, j + |F(E(I_i))|, |E(S) \cup E(I_i)|)$
6:         if $|F(E(S'))| \ge n$ and $|E(S')| < |E(A)|$ then
7:             $A \leftarrow S'$
8:         else if $|F(E(S'))| < n$ then
9:             $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{S'\}$
10:    for each pair $S_1, S_2 \in \mathcal{S}_i$ where $S_1$ corresponds to $(i_1, j_1, \ell_1)$ and $S_2$ corresponds to $(i_2, j_2, \ell_2)$ do
11:        if $i_1 \le i_2$ and $j_1 \le j_2$ and $\ell_1 \ge \ell_2$ then
12:            $\mathcal{S}_i \leftarrow \mathcal{S}_i \setminus \{S_1\}$
13: return $A$
```

Theorem 1. *Algorithm 1 finds an optimal solution for the off-line error-query scheduling problem in time polynomial in k.*

Proof. Note that, if we remove the elimination step in Lines 10–12 of the algorithm, then it will create all possible solutions, i.e., all possible combinations of elementary intervals. Hence, in order to prove the correctness of our algorithm, it is sufficient to show that the elimination step does not remove any partial solution which can lead to a better final solution.

For the sake of contradiction, assume that a partial solution S_1 corresponding to (i_1, j_1, ℓ_1) is eliminated by the algorithm due to its domination by the partial solution S_2 corresponding to (i_2, j_2, ℓ_2) in iteration i. Moreover, let $\mathcal{I}_1$ be the set of elementary intervals that can be added to S_1 and get the final solution $S_1 \cup \mathcal{I}_1$ which queries strictly less errors than the solution obtained by the algorithm. Note that the solution $S_1 \cup \mathcal{I}_1$ reveals $j_1 + |F(E(\mathcal{I}_1))|$ error-free slots. Consider now the solution $S_2 \cup \mathcal{I}_1$ which reveals $j_2 + |F(E(\mathcal{I}_1))|$ error-free slots. By the definition of the elimination rules we have that $j_1 \le j_2$, and hence $S_2 \cup \mathcal{I}_1$ reveals at least as many error-free slots as $S_1 \cup \mathcal{I}_1$. For the errors that are queried by $S_1 \cup \mathcal{I}_1$ we have that $|E(S_1 \cup \mathcal{I}_1)| = |E(S_1) \cup E(\mathcal{I}_1)| = |E(S_1)| + |E(\mathcal{I}_i)| - |E(S_1) \cap E(\mathcal{I}_1)| = \ell_1 + |E(\mathcal{I}_i)| - |E(S_1) \cap E(\mathcal{I}_1)|$. Similarly, for $S_2 \cup \mathcal{I}_1$ it holds that $|E(S_2 \cup \mathcal{I}_1)| = \ell_2 + |E(\mathcal{I}_i)| - |E(S_2) \cap E(\mathcal{I}_1)|$. Since $i_1 \le i_2$, we have that $|E(S_1) \cap E(\mathcal{I}_1)| \le |E(S_2) \cap E(\mathcal{I}_1)|$. By taking into account the elimination rule $\ell_1 \ge \ell_2$, we have that $|E(S_1 \cup \mathcal{I}_1)| \ge |E(S_2 \cup \mathcal{I}_1)|$, which is a contradiction to the assumption that $S_1 \cup \mathcal{I}_1$ queries strictly less errors than the solution obtained by the algorithm.

For the complexity of the algorithm, note first that it iterates $O(|\mathcal{I}|) = O(k)$ times. By the definition of the elimination rules, in each iteration we keep only one value of j for each pair of i and ℓ. Thus, the parameter j can take at most $\min\{n, k^2\}$ different values. Therefore, the number of partial solutions in each iteration is at most $\min\{n, k^2\} \times k \times |\mathcal{I}| < k^4$. Finally, the elimination step considers all pairs of partial solutions, whose number is polynomial to k, and hence the theorem follows. □

3.2 A Lower Bound for the On-line Problem

We next consider the on-line version of the error-query scheduling problem. Recall that we assume that the errors and their uncertainty areas are known to the on-line algorithm, but not their slots of appearance.

In this section we present a lower bound to the competitive ratio of any on-line algorithm. Note that this lower bound is valid for particularly simple instances, i.e., there is only one job to execute as well as the uncertainty areas of the errors belong in both laminar and agreeable families of instances.

Theorem 2. *The competitive ratio of any on-line algorithm for the error-query scheduling problem is $\Omega(\sqrt{k})$, even in the case where $n = 1$ and the instance is both laminar and agreeable.*

Proof. Consider the instance shown in Figure 2 which consists of $k = b^2$ errors. The errors are partitioned into b groups, $E_1, E_2, \ldots, E_b$, each one containing b errors. The uncertainty area of the error $e_{\ell,i}$, i.e., the i-th error in E_ℓ, spans into the interval $((\ell-1)(b+1)+i-1, \ell(b+1)]$, where $1 \le i \le b$ and $1 \le \ell \le b$. The uncertainty areas of any two errors belonging in different groups do not intersect. Specifically, in the interval $((\ell-1)(b+1), \ell(b+1)]$, $1 \le \ell \le b$, there exist only the uncertainty areas of the b errors of E_ℓ. Since this interval spans into $b+1$ slots, there is a unique error-free slot available in it. The goal is to find one error-free slot in the whole instance in order to execute one job. Note that the constructed instance is both laminar and agreeable.

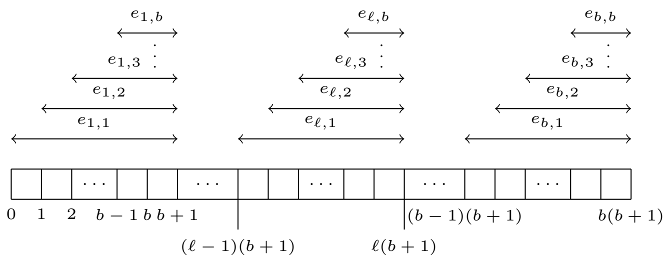

Figure 2: An instance of the error-query scheduling problem for which the adversary queries only one error while it can oblige any on-line algorithm to query $\sqrt{k}$ errors.

Given the above instance, we claim that any on-line algorithm can be forced by the adversary to query at least b errors, while the optimal solution will query only one. According to Property 1 there is no reason to query error $e_{\ell,i}$ without also querying all errors in $\{e_{\ell,1}, \ldots, e_{\ell,i-1}\}$. In particular, if the error-free slot of the set E_ℓ appears in position $\ell(b+1)$, then any algorithm must query all errors in $\{e_{\ell,1}, \ldots, e_{\ell,b}\}$ in order to reveal it. On the other hand, if the error-free slot appears in position $(\ell-1)(b+1)+1$ one request is sufficient to find it.

Assume now that an on-line algorithm has queried any $b-1$ errors. We say that E_ℓ is unimpaired if the on-line algorithm queries no errors from this group. Let us denote the set of unimpaired groups by $\mathcal{E}_0$ and let $\mathcal{E}_1 = \{E_1, E_2, \ldots, E_b\} \setminus \mathcal{E}_0$. We note that both sets are non-empty and each group E_i contains at least one not updated error.

If $E_\ell \in E_1$ the adversary sets the unique error-free slot in the position $\ell(b+1)$. Otherwise, the adversary sets the unique error-free slot in the position $(\ell-1)(b+1)+1$. It follows that the on-line algorithm find no error-free slot after $b-1$ queries and must update at least one more error. In its turn, the optimal solution asks the error $e_{l,1}$ for $E_\ell \in E_0$ and reveal the error-free slot $(\ell-1)(b+1)+1$.

Concluding, the adversary can work in such a way that forces the on-line algorithm to query at least b errors, while the constructed instance contains an error-free slot that can be revealed by only one query. By recalling that $b = \sqrt{k}$, the theorem follows. □

3.3 An O($\sqrt{k}$)-competitive Algorithm for Laminar Instances

In this section we give an $O(\sqrt{k})$-competitive algorithm for laminar instances that exploits their tree-like structure. In each iteration, the algorithm updates the set of already queried errors E' and considers the set of all maximal subtrees with respect to E' as these are defined at the end of Section 2. We say that the subtree $T(v_\ell)$ is *big* if it contains more than $\sqrt{k}$ nodes, that is the uncertainty area of e_ℓ includes the uncertainty areas of more than $\sqrt{k}$ errors. Otherwise, we say that $T(v_\ell)$ is *small*. For simplicity, we call also big and small the errors corresponding to the root of big and small subtrees, respectively. Then, given a set of maximal subtrees, our algorithm at each iteration queries the following errors. For each big maximal subtree $T(v_\ell)$ the error e_ℓ is queried. Moreover, the algorithm queries the errors corresponding to all nodes of the small maximal subtree that contains the maximum number of error-free slots. Note that the number of error-free slots in a maximal subtree can be easily computed. Algorithm 2 presents a formal description of the procedure.

Theorem 3. *Algorithm 2 is $O(\sqrt{k})$-competitive for the error-query scheduling problem on laminar instances.*

Proof. A big subtree cannot be part of a small subtree, that is if v_ℓ is the parent of $v_{\ell'}$ and $T(v_{\ell'})$ is big, then $T(v_\ell)$ is also big. Based on this, we observe that the algorithm at iteration i considers all big subtrees of level i. Since these subtrees are node-independent and each of them contains more than $\sqrt{k}$ errors, their number cannot be greater than $\sqrt{k}$. Hence, the algorithm handles at most $\sqrt{k}$ big subtrees per iteration in Lines 5–10, and for each subtree it queries only one error. Moreover, by the definition of small subtrees, the algorithm queries at most $\sqrt{k}$ errors in Line 12. Therefore, at each iteration the algorithm queries at most $2\sqrt{k}$ errors.

Algorithm 2

```
1: $i \leftarrow 0$; $E' \leftarrow \emptyset$
2: $S_0$: the set of trees in the forest representation of the initial
   instance (the initial set of maximal subtrees)
3: while $n > 0$ do
4:     $i \leftarrow i + 1$; $S_i \leftarrow S_{i-1}$
5:     for each big subtree $T(v_\ell) \in S_{i-1}$ do
6:         Query the error $e_\ell$ and set $E' \leftarrow E' \cup \{e_\ell\}$
7:         Reduce $n$ by the number of the error-free slots that are
           revealed due to the query of $e_\ell$
8:         $S_i \leftarrow S_i \setminus \{T(v_\ell)\}$
9:         for each children $v_{\ell'}$ of $v_\ell$ do
10:            $S_i \leftarrow S_i \cup \{T(v_{\ell'})\}$
11:    Let $T^i \in S_{i-1}$ be the small maximal subtree that can reveal
       the maximum number of error-free slots
12:    Query all errors in $E(T^i)$ and set $E' \leftarrow E' \cup E(T^i)$
13:    $S_i \leftarrow S_i \setminus \{T^i\}$
14:    Reduce $n$ by the number of the error-free slots that are re-
       vealed due to the query of the errors in $E(T^i)$
15: return $E'$
```

In order to finish the proof, we need to show the following claim: the error-free slots revealed by the algorithm up to the end of iteration i, are at least equal to the error-free slots revealed by any solution after querying exactly i errors. This claim can be rephrased as: if the algorithm performs i^* iterations in total then the optimal solution should query at least i^* errors, and hence the competitive ratio is at most $\frac{2\sqrt{k}\cdot i^*}{i^*} = O(\sqrt{k})$.

In order to prove the claim, consider a solution of any algorithm $\mathcal{A}$ that has queried i errors. Let i_{big} and i_{small} be the number of the big and the small errors, respectively, in this solution, i.e., $i = i_{big} + i_{small}$. As mentioned above, there is no interest to query an error without previously querying its parent in the forest representation, since the instance is laminar. Hence, all i errors queried by $\mathcal{A}$ are of level at most i. Then, all i_{big} big errors queried by $\mathcal{A}$ are also queried by our algorithm, since the later one has queried all big errors of levels $1, 2, \ldots, i$ at the end of iteration i. Thus, it remains to show that the i_{small} errors queried by $\mathcal{A}$ cannot reveal more error-free slots than the errors in $T^1 \cup T^2 \cup \ldots \cup T^i$ queried by our algorithm. This is true, since $i_{small} \leq i$ and these i small subtrees are selected by our algorithm in order to contain the maximum number of error-free slots. □

4 The Lexicographic Error-query Scheduling Problem

In this section we consider the lexicographic variant of our problem in which we need to query the minimum number of errors such that to reveal the n earliest error-free slots. The off-line case for this problem is trivial, since the algorithm knows the slot of appearance of each error, and hence the n earliest error-free slots. Thus, it suffices to query all errors whose uncertainty areas contain these slots. In the following, we deal only with the on-line case. We first present a lower bound which holds also for agreeable instances, and then we propose a competitive algorithm for general instances which matches asymptotically this lower bound.

Note that the lexicographic error-query scheduling problem is easy for laminar instances and an optimal on-line algorithm can be designed. This algorithm is based on maximal subtrees and the fact that the number of error-free slots in a maximal subtree can be immediately calculated. Indeed, given the set E' of the already queried errors and a maximal subtree $T(v_\ell)$ with respect to E', the number of error-free slots in $(s_{e_\ell}, f_{e_\ell}]$ is equal to

$$(f_{e_\ell} - s_{e_\ell}) - |E(T(v_\ell))| - a(E', T(v_\ell))$$

where $E(T(v_\ell))$ is the set of errors corresponding to the nodes of $T(v_\ell)$ and $a(E', T(v_\ell))$ is the number of errors in E' whose appearance slot is in $(s_{e_\ell}, f_{e_\ell}]$. Then, the algorithm for laminar instances queries at each iteration the error that corresponds to the root of the leftmost maximal subtree which contains at least one error-free slot. This procedure terminates when the n earliest error-free slots are revealed. The optimality of the algorithm comes from the fact that the queried errors reveal the n earliest error-free slots, while the algorithm queries an error only if it can reveal at least one of these error-free slots.

Theorem 4. *There is an optimal on-line algorithm for the lexicographic error-query scheduling problem on laminar instances.*

4.1 A Lower Bound

We present here a lower bound to the competitive ratio of any on-line algorithm. This lower bound is valid for instances where there is only one job to execute, one error-free slot available as well as the uncertainty areas of the errors form an agreeable instance. Note that the lexicographic property is implied by the uniqueness of the error-free slot.

Theorem 5. *The competitive ratio of any on-line algorithm for the lexicographic error-query scheduling problem is $\Omega(\log_2 k)$, even if $n = 1$ and the instance is agreeable.*

Proof. We consider the instance shown in Figure 3 which spans into the time interval $(0, k + 1]$. In this instance there are k errors, while the uncertainty area of the error e_ℓ spans in $(\ell - 1, \ell + 1]$. Note that there exists exactly one error-free slot in this instance which should be found by the on-line algorithm in order to execute the $n = 1$ job.

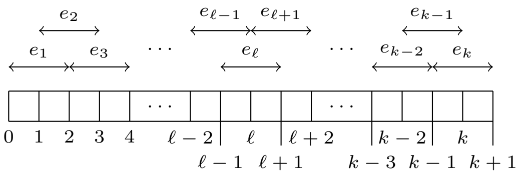

Figure 3: An agreeable instance of the lexicographic error-query scheduling problem for which the adversary queries at most two errors while it can oblige any on-line algorithm to query $\log_2 k$ errors.

Consider any on-line algorithm $\mathcal{A}$ and assume that it initially queries the error e_ℓ. The adversary has two choices: the slot of appearance of e_ℓ is set to be either $(\ell-1, \ell]$ or $(\ell, \ell+1]$. In the first case $\mathcal{A}$ immediately learns that the error-free slot cannot be in the interval $(0, \ell]$. Indeed, there are $\ell - 1$ errors whose uncertainty area entirely spans in this interval plus e_ℓ whose appearance slot is $(\ell - 1, \ell]$. Hence, there are ℓ errors whose appearance slots are in $(0, \ell]$. Due to the assumption that two errors cannot have the same uncertainty slot, we conclude that there is not an error-free slot in $(0, \ell]$. In a similar way, if the adversary sets the slot of appearance of e_ℓ to be $(\ell, \ell + 1]$ then $\mathcal{A}$ immediately learns that the error-free slot cannot be in the interval $(\ell, k + 1]$. In both above cases, the algorithm $\mathcal{A}$ can eliminate a part of the instance and continue the search for the error-free slot by querying a new error to the other part. We assume that $\mathcal{A}$ is a *reasonable* algorithm in the sense that it does not query errors for which it is easy to verify that cannot reveal any error-free slot. Note that the sub-instance in which $\mathcal{A}$ will continue its search, has exactly the same form as the initial instance, but it has a smaller spanning interval and less errors.

Based on the above observation, we consider that the adversary applies always the following policy: given a sub-instance that spans in the interval $(\ell_1, \ell_2 + 1]$ and the query for an error e_ℓ, where $\ell_1 \leq \ell \leq \ell_2$, set $t^*_{e_\ell} = (\ell - 1, \ell]$ if $\ell - \ell_1 \leq \ell_2 + 1 - \ell$, and $t^*_{e_\ell} = (\ell, \ell + 1]$ otherwise. In this way, the spanning interval of the remaining sub-instance which will be considered by the reasonable algorithm $\mathcal{A}$ is maximized. Specifically, the new sub-instance spans to an interval of size $\max\{\ell - \ell_1, \ell_2 + 1 - \ell\}$ and it contains $\max\{\ell - \ell_1, \ell_2 + 1 - \ell\} - 1$ errors. On the other hand, the algorithm $\mathcal{A}$ should select the error e_ℓ such that the new sub-instance is of minimum size, which can be achieved if $\ell = \lceil \frac{\ell_2 - \ell_1}{2} \rceil$. In other words, at each step the spanning interval of the sub-instance is decreased by half, and hence $\mathcal{A}$ is forced to query $O(\log_2 k)$ errors before revealing the error-free slot. Since all slots are included in the uncertainty area of at most two errors, the adversary needs at most two queries to reveal the error-free slot, and the theorem follows. ☐

4.2 An $O(\log_2 k)$-competitive Algorithm

In this section we propose an on-line algorithm for the lexicographic error-query scheduling problem whose main idea is to do a pruning of the intervals in which either no error-free slots exist or an error-free slot exists in an earliest interval. In order to safely decide if an interval contains an error-free slot, we adapt the definition of maximal subtrees used for laminar instances as follows: given a set E' of already queried errors, we say that a time interval $I = (t_a, t_b]$ is *maximal* with respect to E' if for each $e \in E \setminus E'$ either $(s_e, f_e] \subseteq (t_a, t_b]$ or $(s_e, f_e] \cap (t_a, t_b] = \emptyset$. Let $E(I)$ be the set of errors in $E \setminus E'$ that satisfy the first condition, i.e., $e \in E(I)$ if $(s_e, f_e] \subseteq (t_a, t_b]$. Then, the number of error-free slots in I is equal to $(t_b - t_a) - |E(I)| - c$, where c is the number of errors in E' whose slot of appearance occurs in I.

In what follows, we consider the slots partitioned into groups: a slot t belongs to group G_i, $i \geq 2$, if the number of errors whose uncertainty area contain t is between $2^{i-2} + 1$ and 2^{i-1}. For convenience, let G_0 and G_1 be the groups of slots that are contained to the uncertainty area of zero and one errors, respectively. The total number of groups is in $O(\log_2 k)$. Given two slots t, t' with $t \leq t'$, let $G_i[t, t']$ be the set of slots of group G_i that appear in the interval $(t - 1, t']$. We define the *middle* slot of a set $G_i[t, t']$ to be the slot in position $\lceil |G_i[t, t']| / 2 \rceil$ if we consider the slots in $G_i[t, t']$ ordered from left to right with respect to the time-line.

In each iteration, the goal of our algorithm is to discover exactly one error-free slot: the earliest one which is not yet revealed. To do this, it performs a kind of binary search in the slots of the time horizon, starting from the slots of G_1, then of G_2, and so on. Specifically, the algorithm first considers the middle slot in G_1 and queries the error which contains it. Depending on the existence of at least one error-free slot on the left of the middle slot, we eliminate the interval on the right or on the left of this slot. The middle slot of the new interval that belongs to G_1 is sought, and we repeat the above procedure until the reduced interval does not contain any other slot of G_1. Then, we continue the procedure by choosing the middle slot of the reduced interval that belongs to G_2. If such a slot exists, we query all errors that contain it, and so on. Note that at the end of each step of the binary search, all errors containing the current middle slot are queried. However, we do not have to query again already queried errors. Algorithm 3 describes formally the above procedure.

Algorithm 3

```
1:  $E' \leftarrow \emptyset, SOL \leftarrow \emptyset$
2:  for $j \leftarrow 1$ to $n$ do
3:      $left \leftarrow 1$, $right \leftarrow$ last slot, $found \leftarrow False$, $i \leftarrow 1$
4:      while $found = False$ do
5:          if there is an error-free slot $t \notin SOL$ with $left \leq t \leq$
            $right$ such that there is no error-free slot $t' \notin SOL$ with
            $left \leq t' \leq t - 1$ and all errors containing $t$ are in $E'$
            then
6:              $SOL \leftarrow SOL \cup \{t\}$, $found \leftarrow True$
7:              Update the groups $G_i$ with respect to the new errors
                added in $E'$ during current iteration
8:          else if $|G_i[left, right]| \leq 2$ then
9:              $i \leftarrow i + 1$
10:         else
11:             $mid \leftarrow$ the middle slot of $G_i[left, right]$
12:             Query all errors in $E \setminus E'$ that contain $mid$ and add
                them to $E'$
13:             if there is an error-free slot $t \notin SOL$ with $left \leq t \leq$
                $right$ then
14:                 $right \leftarrow mid$
15:             else
16:                 $left \leftarrow mid$
```

Note that the intervals $(left - 1, right]$ in Lines 5 and 13 of Algorithm 3 are maximal with respect to the current set of errors E', and hence these two conditions can be justified as described in the beginning of the section.

Theorem 6. *Algorithm 3 is $O(log_2 k)$-competitive for the lexicographic error-query scheduling problem.*

Proof. By the definition of the problem, the algorithm and the adversary find exactly the same set of n error-free slots, let $\{t_{j_1}, t_{j_2}, \ldots, t_{j_n}\}$. We consider these slots ordered with respect to the time-line. Note that the algorithm reveals them according to this order. Without loss of generality, we can assume that the adversary uses also the same order. Let E^*_ℓ, $1 \leq \ell \leq n$, be the set of errors queried by the adversary in order to reveal t_{j_ℓ}, without querying again the errors in $E_1^* \cup E_2^* \cup \ldots \cup E_{\ell-1}^*$. Then, the optimal solution queries $\sum_{\ell=1}^{n} |E_\ell^*|$ errors. In a similar way, let E_ℓ, $1 \leq \ell \leq n$, be the set of errors queried by the algorithm in iteration ℓ, i.e., the algorithm queries $\sum_{\ell=1}^{n} |E_\ell|$ errors.

Since both the algorithm and the adversary reveal the slots in the same order and do not query again already queried errors, we can assume that after revealing the slot t_{j_ℓ} the instance is reduced by eliminating the queried errors (see Line 7 of the algorithm). Moreover, we consider that the groups G_i are appropriately updated. Assume that the slot t_{j_ℓ}, $1 \leq \ell \leq n$, belongs to the group G_{i_ℓ} at the iteration during which it is revealed. If $i_\ell = 0$, then both algorithm and adversary does not query any additional error. If $i_\ell \geq 1$, then the adversary queries at least $2^{i_\ell - 2} + 1$ errors, while the algorithm at most $\sum_{i=1}^{i_\ell} 2^{i-1} \log_2 |G_i| \leq 2^{i_\ell} \log_2 k$ additional errors. Therefore, the algorithm queries at most a factor of $4 \log_2 k$ more errors than the adversary for revealing each error-free slot, and hence the theorem follows. □

5 Conclusion

In this paper, we gave a polynomial-time algorithm for the off-line error-query scheduling problem. For the on-line version of the problem, we proved that the competitive ratio of any on-line algorithm is $\Omega(\sqrt{k})$ even if the instance is laminar, or agreeable and even if we search for only one free slot. Then, we proposed on $O(\sqrt{k})$-competitive algorithm for laminar instances. Finally, we studied the on-line lexicographic error-query scheduling problem and we proved that its competitive ratio is in $\Theta(\log_2 n)$. A research question that remains open is the development of efficient on-line algorithms for general instances for the error-query scheduling problem. Going further, studying non-preemptive jobs of arbitrary processing times is an interesting direction.

Acknowledgments

This work has been supported by the French ANR-14-CE24-0007-01 project "CoCoRICo-CoDeC" and by a LIP6 project.

References

[Alpern and Lidbetter, 2013] Steve Alpern and Thomas Lidbetter. Mining coal or finding terrorists: The expanding search paradigm. *Operations Research*, 61:265–279, 2013.

[Angelopoulos,] Spyros Angelopoulos. Further connections between contract-scheduling and ray-searching problems. In *IJCAI 2015*, pages 1516–1522.

[Angelopoulos, 2016] Spyros Angelopoulos. Deterministic searching on the line. In *Encyclopedia of Algorithms*, pages 531–533. 2016.

[Benabbou and Perny, a] Nawal Benabbou and Patrice Perny. Adaptive elicitation of preferences under uncertainty in sequential decision making problems. In *IJCAI 2017*, pages 4566–4572.

[Benabbou and Perny, b] Nawal Benabbou and Patrice Perny. Incremental weight elicitation for multiobjective state space search. In *AAAI 2015*, pages 1093–1099.

[Boutilier *et al.*, 2006] Craig Boutilier, Relu Patrascu, Pascal Poupart, and Dale Schuurmans. Constraint-based optimization and utility elicitation using the minimax decision criterion. *Artif. Intell.*, 170:686–713, 2006.

[Condon *et al.*, 2009] Anne Condon, Amol Deshpande, Lisa Hellerstein, and Ning Wu. Algorithms for distributional and adversarial pipelined filter ordering problems. *ACM Trans. Algorithms*, 5:24:1–24:34, 2009.

[Drummond and Boutilier,] Joanna Drummond and Craig Boutilier. Elicitation and approximately stable matching with partial preferences. In *IJCAI 2013*, pages 97–105.

[Dürr *et al.*,] Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. Scheduling with explorable uncertainty. In *ITCS 2018*, pages 30:1–30:14.

[Erlebach and Hoffmann, 2015] Thomas Erlebach and Michael Hoffmann. Query-competitive algorithms for computing with uncertainty. *Bulletin of the EATCS*, 116, 2015.

[Feder *et al.*, 2003] Tomás Feder, Rajeev Motwani, Rina Panigrahy, Chris Olston, and Jennifer Widom. Computing the median with uncertainty. *SIAM J. Comput.*, 32:538–547, 2003.

[Feder *et al.*, 2007] Tomás Feder, Rajeev Motwani, Liadan O'Callaghan, Chris Olston, and Rina Panigrahy. Computing shortest paths with uncertainty. *J. Algorithms*, 62:1–18, 2007.

[Gelain *et al.*, 2010] Mirco Gelain, Maria Silvia Pini, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Elicitation strategies for soft constraint problems with missing preferences: Properties, algorithms and experimental studies. *Artif. Intell.*, 174:270–294, 2010.

[Goerigk *et al.*, 2015] Marc Goerigk, Manoj Gupta, Jonas Ide, Anita Schöbel, and Sandeep Sen. The robust knapsack problem with queries. *Computers & OR*, 55:12–22, 2015.

[Gupta *et al.*, 2011] Manoj Gupta, Yogish Sabharwal, and Sandeep Sen. The update complexity of selection and related problems. In *FSTTCS*, pages 325–338, 2011.

[Hoffmann *et al.*,] Michael Hoffmann, Thomas Erlebach, Danny Krizanc, Matús Mihalák, and Rajeev Raman. Computing minimum spanning trees with uncertainty. In *STACS 2008*, pages 277–288.

[Kahan,] Simon Kahan. A model for data in motion. In *ACM STOC 1991*, pages 267–277.

[Megow *et al.*, 2015] Nicole Megow, Julie Meißner, and Martin Skutella. Randomization helps computing a minimum spanning tree under uncertainty. In *ESA*, pages 878–890, 2015.

[Olston and Widom,] Chris Olston and Jennifer Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *VLDB 2000*, pages 144–155.