An Optimal Incremental Algorithm for Minimizing Lateness with Rejection

Samir Khuller[1,⋆] and Julián Mestre[2,⋆⋆]

[1] University of Maryland, College Park, MD 20742, USA
[2] Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany

Abstract. This paper re-examines the classical problem of minimizing maximum lateness which is defined as follows: given a collection of n jobs with processing times and due dates, in what order should they be processed on a single machine to minimize maximum lateness? The lateness of a job is defined as its completion time minus its due date. This problem can be solved easily by ordering the jobs in non-decreasing due date order. We now consider the following question: which subset of k jobs should we reject to reduce the maximum lateness by the largest amount? While this problem can be solved optimally in polynomial time, we show the following surprising result: there is a fixed permutation of the jobs, such that for all k, if we reject the first k jobs from this permutation, we derive an optimal solution for the problem in which we are allowed to reject k jobs. This allows for an incremental solution in which we can keep incrementally rejecting jobs if we need a solution with lower maximum lateness value. Moreover, we also develop an optimal $O(n \log n)$ time algorithm to find this permutation.

1 Introduction

Scheduling problems arise in many contexts in computer science and operations research. Let us begin by defining the problem of *scheduling jobs to minimize maximum lateness*. Given a set of jobs A, each having a processing time and a due date, we want to schedule the jobs on a single machine. A job is considered to be *late* if it finishes after its due date, in which case its *lateness* is the difference between its finishing time and its due date; if a job finishes on time, its lateness is 0. Our objective is to find a schedule on a single machine minimizing the maximum lateness among all jobs.

More formally, let $A = \{1, \ldots, n\}$ be a set of jobs and let p_i and d_i denote the processing time and due date of job i. Without loss of generality, we can assume that in an optimal solution the machine is never idle and that the schedule is non-preemptive. Thus a schedule is specified with a permutation σ on n elements,

⋆ Research supported by NSF grant CCF 0728839.
⋆⋆ Research supported by an Alexander von Humboldt Fellowship.

D. Halperin and K. Mehlhorn (Eds.): ESA 2008, LNCS 5193, pp. 601–610, 2008.

where $\sigma(j)$ denotes position of job $j \in A$ in our schedule. Then the lateness of the ith job can be defined as

$$L_i = \sum_{j:\sigma(j)\leq\sigma(i)} p_j - d_i. \tag{1}$$

Our objective is to find a permutation σ minimizing $\max_i L_i$. It is well-known [6] that scheduling the jobs in non-decreasing due date order yields an optimal solution.

For the problem of *scheduling jobs to minimize maximum lateness with rejection*, in addition to the n jobs, we are given a budget k. Our objective is to identify a set of k jobs to reject, so as to minimize the maximum lateness of the remaining jobs. An incremental solution for the problem is a list of the jobs such that for any k, the first k jobs in the list form an optimal solution for minimizing lateness with k rejections. Our main contribution is to show that such a list always exists and that it can be computed in $O(n \log n)$ time. Not only does the incremental approach let us develop a faster algorithm, it also uncovers some surprising structural properties of the underlying problem. Moreover, if all due dates are identical then we need to order the jobs in non-increasing processing time order, since this is an optimal rejection order, so the problem is at least as hard as sorting.

Previous Work: Scheduling to minimize maximum lateness with rejection was studied by Sengupta [11]. In fact, he considered a more general formulation where each job j has a rejection penalty of e_j, and there is a bound on the total penalty of rejected jobs. In this case Sengupta shows that the problem is actually *NP*-complete. However, he also gives a simple dynamic programming solution that runs in time $O(nk + n \log n)$ when all $e_j = 1$, and there is a budget k on the number of jobs we can reject. This algorithm computes the optimal set of k jobs to reject, to get the maximum possible reduction in the maximum lateness. Of course, it may happen that the optimal set of k jobs to reject chosen by the algorithm is not a subset of the optimal set of $k+1$ jobs to reject chosen by the algorithm.

Other scheduling problems with rejection have been considered as well, both in the offline setting [3,7,10,1] and in the online setting [4,1]). To the best of our knowledge, none of these works have considered incremental solutions for scheduling with rejection.

Related Work on Incremental Algorithms: Perhaps the earliest example of an incremental algorithm is Gonzalez's algorithm for the K-center problem [5], which yields a 2 approximation. Mettu and Plaxton [9] defined the online median problem and showed that there is a way to choose centers incrementally, such that selecting the first K centers, gives a constant factor approximation to the K-median problem. Even though several constant factor approximations were developed for the basic K-median problem, there is no mechanism to enforce that the solution using K medians would be a subset of the solution using K' medians when $K' > K$. Mettu and Plaxton's work then led to several subsequent improvements and simpler proofs [8,2].

For the problem of minimizing maximum lateness with rejection, we develop an optimal solution for the rejection problem and moreover prove that the optimal solution can be computed incrementally. As a consequence it follows that there is an optimal rejection set of i jobs, that is a subset of the an optimal rejection set of $(i+1)$ jobs. However, in choosing an optimal rejection set of size i one has to be extremely careful, since there are many optimal solutions and not all of them have the incremental property.

2 Alternative Problem Formulation and Notation

Before presenting the algorithms, it is convenient to modify the problem formulation slightly. The fact that a job's lateness (1) is allowed to be negative can make the analysis cumbersome. A standard way [6] to avoid this issue is to add a sufficiently large constant to the right hand side of (1) so that the lateness of every job is always positive.

$$M_i = \sum_{j:\sigma(j)\leq\sigma(i)} p_j + \left(\max_{h\in A} d_h - d_i\right). \tag{2}$$

There is a natural interpretation of measure (2): After the machine finishes processing job i, the job must be *delivered*; only once the job is delivered we considered the job to be completed. The *delivery time* of the ith job is given by $s_i = \max_{h\in A} d_h - d_i$. Although our single machine can process only one job at a time, any number of jobs can be delivered in parallel (see Figure 1.) The objective is to minimize the makespan of the schedule, that is, the maximum completion time over all jobs. The two formulations are equivalent since a schedule with makespan $\max_{h\in A} d_h + \delta$ under (2) has lateness δ under (1), and vice versa.

Another way to deal with negative lateness is to minimize tardiness, which is defined as $T_i = \max\{0, L_i\}$. Clearly, if we have an optimal algorithm for minimizing lateness with rejection, we immediately get an algorithm for minimizing tardiness with rejection: Once the lateness becomes negative we can stop rejecting jobs for the tardiness objective.

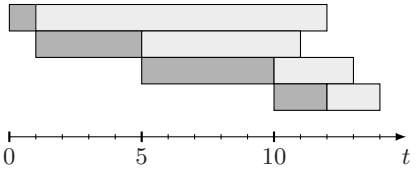
Notation: When talking about a set A, we use $A-j$ and $A+i$ to denote $A\setminus\{j\}$ and $A\cup\{i\}$ respectively. We use $p(A)$ as a shorthand for $\sum_{j\in A} p_j$. When talking about a sequence ℓ, we use $\ell(1)$ and $\ell(|\ell|)$ to denote the first and last elements of ℓ respectively.

For simplicity, from now on we assume that the jobs are given in non-increasing order of delivery time; that is, we assume that $s_1 \geq s_2 \geq \ldots \geq s_n$. Thus, for any set of jobs $X \subseteq A$, we can denote the completion time of job $i \in X$ in an optimal schedule for X with

$$M_i^X = p\big(\{j \in X \mid j \leq i\}\big) + s_i.$$

And the makespan of X with

$$M(X) = \max_{i\in X} M_i^X.$$



i	p_i	s_i
1	1	11
2	4	6
3	5	3
4	2	2

Fig. 1. Dark rectangles denote processing times and light rectangles denote delivery times. Why a greedy choice is not enough: The third job is a greedy choice, but the only optimal solution when $k = 2$ is to reject the first and the second jobs.

3 An Incremental Solution

Our goal is to produce an optimal incremental solution for scheduling with rejections to minimize lateness. In other words, we want to construct a list of jobs $x_1, x_2, \ldots, x_n$ such that for any k, the set $\{x_1, \ldots, x_k\}$ is an optimal solution for minimizing lateness with k rejections. Clearly, the only way to produce such a solution is to repeatedly remove the job that decreases the lateness of the remaining jobs the most, we call this a *greedy choice*.

Definition 1. *A job $i \in A$ is said to be a* greedy choice *for a set of jobs A if $M(A - i) \leq M(A - j)$ for all $j \in A$.*

Interestingly, repeatedly selecting a greedy choice may not lead to an optimal incremental solution. Consider the example in Figure 1. The third job is a greedy choice, but for $k = 2$ the unique optimal solution is to reject the first and the second jobs. There is still hope, however, since the instance does allow an optimal incremental solution, namely $\langle 2, 1, 3, 4 \rangle$. To get around this pitfall we need a notion stronger than greedy choice.

Definition 2. *Let $A = \{1, \ldots, n\}$ be a set of jobs. A job $i \in A$ is said to be a* strongly greedy choice *for A if i is a greedy choice for $\{j, \ldots, n\}$ for all $j \leq i$.*

Our algorithm, whose pseudo-code is given below, repeatedly identifies a strongly greedy choice for A, adds it to the list, and removes it from A. It is worth noting here that the existence of a strongly greedy choice is not obvious. Indeed, in the next section we show that such a job always exists.

```
INCREMENTAL(A)
1  ℓ ← ⟨ ⟩
2  while A ≠ ∅ do
3      i ← strongly greedy choice for A
4      insert i at the end of ℓ
5      A ← A − i
6  return ℓ
```

4 Analysis

In this section we prove that INCREMENTAL always finds an optimal incremental solution for minimizing lateness with rejections. To that end we introduce two lemmas, whose proofs make use of the following property.

Property 1. Let $A = \{1, \ldots, n\}$ and $A' = \{2, \ldots, n\}$. For any set $X \subseteq A'$ we have $M(A \setminus X) = \max\{p_1 + s_1, M(A' \setminus X) + p_1\}$.

Lemma 1 will establish that Line 3 in our algorithm is well defined, and Lemma 2 will show that the choice made there is the right one.

Lemma 1. *Every set A of jobs admits a strongly greedy choice.*

Proof. By induction on the size of $A = \{1, \ldots, n\}$. The base case ($n = 1$) is trivial. For the inductive step ($n > 1$), if 1 is a greedy choice then we are done since 1 is trivially a strongly greedy choice, so let us assume otherwise.

Let $A' = \{2, \ldots, n\}$. By induction, there exists a strongly greedy choice i for A'; thus, we only need to show that i is a greedy choice for A. Since i is a greedy choice for A', we have $M(A' - i) \leq M(A' - j)$ for any $j > 1$. By Property 1, it follows that $M(A - i) \leq M(A - j)$ for any $j > 1$. Furthermore, since 1 is not a greedy choice for A, we have $M(A - 1) > M(A - j)$ for some $j > 1$; thus, $M(A - i) < M(A - 1)$ and we are done. □

Lemma 2. *Let i be a strongly greedy choice for A. For any set $S \subseteq A - i$ there exists $j \in S$ such that $M(A \setminus (S - j + i)) \leq M(A \setminus S)$.*

Proof. By induction on the size of A and $k = |S|$. For the base case ($k = 1$) we note that i is a greedy choice for A so the lemma holds. For the inductive step ($k > 1$) let $A = \{1, \ldots, n\}$ and $A' = \{2, \ldots, n\}$. There are a few cases to consider depending on whether $1 \in S$ or $1 = i$.

First, consider the case $1 \notin S$ and $i \neq 1$. By Definition 2, i is a strongly greedy choice for A'. By induction, there exists $j \in S$ such that $M(A' \setminus (S - j + i)) \leq M(A' \setminus S)$. It follows, by Property 1, that $M(A \setminus (S - j + i)) \leq M(A \setminus S)$.

Second, consider the case $1 \in S$ and $i \neq 1$. Let $S' = S - 1$. Notice that $M(A \setminus S) = M(A' \setminus S')$. Again, i is a strongly greedy choice for A'. By inductive hypothesis on A and S' there is a job $j \in S'$ such that $M(A' \setminus (S' - j + i)) \leq M(A' \setminus S')$. Thus, it follows that $M(A \setminus (S - j + i)) \leq M(A \setminus S)$.

Third, consider the case $i = 1$. Let j be the smallest job in S. We will argue that $M(A \setminus (S - j + 1)) \leq M(A \setminus S)$. Let t be the leftmost job attaining the makespan of $M(A - j)$. If $t < j$ then $M(A - j) = M_t^A$ and $M(A \setminus S) = M_t^A$; furthermore, since 1 is a greedy choice, we have $M(A-1) \leq M(A-j) = M(A\setminus S)$. Otherwise, if $t > j$, we have $M(A - j) = M_t^A - p_j$. Since $M(A - 1) \geq M_t^A - p_1$, this implies $p_1 \geq p_j$. Clearly, the finishing time of all jobs other than j cannot increase since $p_1 \geq p_j$. We only need to show that the finishing time of j is at most $M(A \setminus S)$. Let $X = \{2, \ldots, j - 1\}$ be the set of jobs scheduled before j.

$$M_j^{A\setminus(S-j+1)} = p(X) + p_j + s_j \leq p(X) + p_1 + s_{j-1} = M_{j-1}^{A\setminus S} \leq M(A \setminus S).$$

We have exhausted all possible cases, so the lemma follows. □

Theorem 1. *The procedure* INCREMENTAL *outputs an optimal incremental solution.*

Proof. First we note that the algorithm actually outputs a solution since, by Lemma 1, Line 3 is well defined. Let $A = \{1, \ldots, n\}$ be the input of INCREMENTAL and $\langle x_1, \ldots, x_n \rangle$ be its output. We prove that $\{x_1, \ldots, x_k\}$ is an optimal solution with k rejections by induction on k and n. The base case, where $k = 1$ and $n \in Z^+$, is trivial since x_1 is a greedy choice for A.

For the inductive step, let S be an optimal solution with k rejections for A. By Lemma 2, we can assume without loss of generality that $x_1 \in S$. Therefore, $S - x_1$ is an optimal solution with $k - 1$ rejections for $A - x_1$. We can think of $x_2, \ldots x_n$ as the output of INCREMENT$(A - x_1)$. Thus, by induction, $\langle x_2, \ldots, x_k \rangle$ is an optimal solution with $k - 1$ rejections for $A - x_1$. It follows that $x_1, \ldots, x_k$ is an optimal solution with k rejections for A. □

5 Implementation

So far we have focused on the correctness of INCREMENTAL and have not discussed its running time. Although it is not difficult implement INCREMENTAL to run in $O(n^3)$ time, in this section we outline two variations of it that lead to faster running times. The first algorithm is based on divide and conquer and runs in $O(n^2)$ time. The second algorithm resembles insertion sort and runs in $O(n \log n)$ time. The reason for including the description of the slower algorithm is two-fold: First, its implementation details are more straightforward than the faster algorithm; second, its quadratic running time is a worst-case bound and it should perform better in practice.

In each case, to prove that the algorithms produce an optimal incremental solution we argue that their output coincides with INCREMENTAL. It should be noted right away that INCREMENTAL is underspecified, since there could be many strongly greedy choices to select from in Line 3. However, *every* possible execution produces a valid incremental solution. From now on, when we say "the output of algorithm X is the same as INCREMENTAL" we mean there exists an execution of INCREMENTAL whose output is the same as that of algorithm X.

5.1 Divide and Conquer

Consider the following divide and conquer algorithm. Let $A = \{1, \ldots, n\}$ be our input instance. First, we find the smallest greedy choice for A, denote this job by i. Second, we identify the smallest job j attaining the maximum lateness in $A - i$. If $j > i$ then i is a strongly greedy choice (this will be proven later) in which case, i must come first followed by an incremental solution for $A - i$. Otherwise $j < i$, in this case we make two recursive calls on $\{1, \ldots, j\}$ and $\{j + 1, \ldots, n\}$. To merge the solutions returned by the two calls, take the leading job from the second solution, followed by the jobs from the first solution (in order), followed by the remaining jobs from the second sequence (also in order). The pseudo-code for this procedure is given below.

```
DIVIDE-AND-CONQUER(A = {1, ..., n})
 1  i ← min{p | p is a greedy choice of A}
 2  j ← min{p | p has maximum lateness in A − i}
 3  if j > i
 4      ℓ ← DIVIDE-AND-CONQUER(A − i)
 5      insert i to the front of ℓ
 6  else
 7      ℓ ← DIVIDE-AND-CONQUER({j + 1, ..., n})
 8      ℓ′ ← DIVIDE-AND-CONQUER({1, ..., j})
 9      insert ℓ′ after the first element of ℓ
10  return ℓ
```

Theorem 2. *The procedure* DIVIDE-AND-CONQUER *can be implemented to run in $O(n^2)$ time and returns an optimal incremental solution for minimizing lateness with rejections.*

Proof. Let $T(n)$ be the running time of the algorithm on an instance with n jobs. It can be shown that finding the leftmost greedy choice, splitting the instance for the recursive calls, and the merging can be done in $O(n)$ time. Therefore, the running time obeys the recursion $T(n) = T(n_1) + T(n_2) + O(n)$ for some $n_1 + n_2 = n$. If we had control over how the instance is split we could choose $n_1 = n_2 = \frac{n}{2}$ to get a running time of $O(n \log n)$. Of course, we do not have control over this and in the worst case we have $n_1 = 1$ and $n_2 = n - 1$, which yields a running time of $T(n) = O(n^2)$.

To prove the correctness of the algorithm, let us show by induction on n that the output of DIVIDE-AND-CONQUER is the same as INCREMENTAL. The base case ($n = 1$) is obvious. For the inductive step ($n > 1$), if $j > i$ then we claim that i is strongly greedy, in which case both algorithms place i first and then process $A - i$, which by inductive hypothesis we can assume to be the same. Let i^* be the leftmost strongly greedy choice for A and assume, for the sake of contradiction, that $i < i^*$. This means that there exists $h < i$ such that for $A' = \{h, \ldots n\}$ we have $M(A' - i^*) < M(A' - i)$. Note, however, that

$$M(A - i^*) = \max\left\{M(A \setminus A'),\ p(A \setminus A') + M(A' - i^*)\right\}$$

equals

$$M(A - i) = \max\left\{M(A \setminus A'),\ p(A \setminus A') + M(A' - i)\right\}.$$

This mean that $M(A \setminus A') = M(A - i)$ contradicting the fact that j is the leftmost job with maximum lateness in $A - i$.

Consider the case when $j < i$. Let i^* be a strongly greedy choice for A. Clearly the makespan of $A - i^*$ is attained by j and $i^* \geq i > j$. Now consider what happens in the execution of INCREMENTAL(A) after processing i^*. For all $h > j$ in $A - i^*$ we have $M_h^{A-i^*} \leq M_j^{A-i^*}$. Since the finishing time of j in $A - i^*$ is larger than that of jobs $h > j$, the next job to be removed by INCREMENTAL must be less or equal than j. This is true until all jobs in $\{1, \ldots, j\}$ are removed:

Suppose the algorithm has removed so far the jobs $X \subset \{1, \ldots, j\}$ and let j' be the largest indexed job in $\{1, \ldots, j\} \setminus X$, then

$$M_{j'}^{(A-i^*)\setminus X} = M_j^{A-i^*} - p(X) - s_j + s_{j'} \geq M_h^{A-i^*} - p(X) = M_h^{(A-i^*)\setminus X}$$

This mean that after removing i^*, INCREMENTAL removes all jobs in $\{1, \ldots, j\}$ before removing any jobs from $\{j+1, \ldots, n\} - i^*$. By inductive hypothesis the recursive calls in Lines 7 and 8 find the optimal orderings for $\{1, \ldots, j\}$ and $\{j+1, \ldots, n\}$ respectively, which are then combined accordingly in Line 9. □

5.2 Fast Incremental

In order to further improve the running time, we introduce an interesting property about the structure of incremental solutions.

Lemma 3. *Let $A = \{1, \ldots, n\}$ be a set of jobs and $B = \{j, \ldots, n\}$ be any suffix of A. Then the order induced on B by the solution output by* INCREMENTAL(A) *and the order of the solution output by* INCREMENTAL(B) *are the same.*

Proof. As we already mentioned at the beginning of the section, the lemma statement does not imply that every execution of INCREMENTAL(A) and INCREMENTAL(B) will coincide; rather, we mean that for every execution of the former, there is an execution of the latter in which the orderings coincide, and vice versa.

By induction on n. Suppose that i is chosen by INCREMENTAL(A) as a strongly greedy choice for A. If $i \in A \setminus B$ then it does not affect INCREMENTAL(B), and by inductive hypothesis on $A - i$ and B their output is the same. Otherwise, i must also be a strongly greedy choice for B, so both algorithms agree on their first decision and by inductive hypothesis on $A - i$ and $B - i$ the rest of the output also coincides. Conversely, suppose i is chosen by INCREMENTAL(B) as strongly greedy choice for B. Let i^* be the leftmost greedy choice of A. If $i^* \in A \setminus B$, we let INCREMENTAL(A) use this job, by inductive hypothesis on $A - i^*$ and B the rest of the output coincides. Otherwise, $i^* \in B$ for A, in which case we claim that i is also a strongly greedy choice for A and by inductive hypothesis the lemma follows. Consider any suffix A' of A, by definition i^* is a greedy choice for A', furthermore

$$M(A' - i^*) = \max\{M(A' \setminus B),\ p(A' \setminus B) + M(B - i^*)\}.$$

Similarly,

$$M(A' - i) = \max\{M(A' \setminus B),\ p(A' \setminus B) + M(B - i)\}$$

However, since i is a (strongly) greedy choice for B we have $M(B-i) \leq M(B - i^*)$. Thus, it follows that i is a greedy choice for A'; that is, $M(A' - i) \leq M(A' - i^*)$. □

It is worth noting that a similar statement about the prefixes of A is not true, and it is ultimately the reason why we cannot modify DIVIDE-AND-CONQUER to run in $O(n \log n)$ time. Nevertheless, a scheme similar to insertion sort does achieve this running time. The underlying idea is very simple: Process jobs from right to left, maintaining an incremental solution for the jobs processed thus far.

```
FAST-INCREMENTAL(A = {1, ..., n})
1  ℓ ← ⟨ ⟩
2  for i ← n down to 1 do
3      let j ∈ {1, ..., |ℓ| + 1} be the smallest index such that i
         is a greedy choice for {i, ℓ(j), ..., ℓ(|ℓ|)}
4      insert i to the left of the jth position in ℓ
5  return ℓ
```

Theorem 3. *The procedure* FAST-INCREMENTAL *can be implemented to run in* $O(n \log n)$ *time and outputs an optimal incremental solution.*

Proof. Let us first argue the correctness of FAST-INCREMENTAL and then discuss the details behind its implementation. Consider the $k+1$st iteration of FAST-INCREMENTAL where we are trying to insert $i = n - k$ into ℓ and $|\ell| = k$, and denote by ℓ' the ordering after i is inserted. Let us show by induction on k that ℓ' is an incremental solution for $\{i, \ldots, n\}$. For the base case ($k = 0$) there is nothing to show. For the inductive step ($k > 0$), by Lemma 3 it suffices to prove that i is inserted in ℓ to the left of the jth element (or at the end if $j = k+1$) where j is the smallest index such that i is a strongly greedy choice for $\{i, \ell(j), \ldots, \ell(k)\}$, which happens if and only if i is a greedy choice for that set.

To argue the $O(n \log n)$ running time we show that Line 3 of the $k+1$st iteration can be carried out in $O(\log k)$ time. As a warm-up we first discuss a slower $O(k)$ time implementation. For the given sequence ℓ, define μ_j to be the makespan of $\{\ell(j), \ldots, \ell(k)\}$, that is, $\mu_j = M(\{\ell(j), \ldots, \ell(k)\})$ and $\mu_{k+1} = \mu_{k+2} = 0$. The following easy-to-prove property is the basis for our implementation of Line 3.

Property 2. Job i is a greedy choice for the set $\{i, \ell(j), \ldots, \ell(k)\}$ if and only if $\mu_j \leq \max\{\mu_{j+1}, s_i\} + p_i$.

Thus, provided with the μ-values we can find the correct position where to insert i in $O(k)$ time. Although computing the μ-values from scratch could take as much as $O(k^2)$ time, we can update the values from the previous iteration in just $O(k)$ time: If i is to be inserted to the left of the jth position in the sequence then we set $\mu'_h = \max\{s_i, \mu_h\} + p_i = \mu_h + p_i$ for $1 \leq h < j$, $\mu'_j = \max\{s_i, \mu_j\} + p_i$, and $\mu'_{h+1} = \mu_h$ for $j \leq h \leq k$.

To improve upon this, we need to keep track of the differences of the μ-values instead of the μ-values themselves. Let $\delta_j = \mu_j - \mu_{j+1}$ for $1 \leq j \leq k$, where μ_{k+1} is taken to be 0. To find out where to insert i first we identify the smallest j' such that $\mu_{j'} < s_i$. Observe that j' fulfills the condition of Property 2; thus, we only need to check whether there exists $j'' < j'$ for which the same condition holds.

For $j'' = j' - 1$ we can check directly. For $j'' \leq j' - 2$, since $\mu_{j''+1} \geq \mu_{j'-1} \geq s_i$, the condition of Property 2 is the equivalent to $\mu_{j''} - \mu_{j''+1} \leq p_i$, in which case j'' is the smallest index such that $\delta_{j''} \leq p_i$ and $j'' \leq j' - 2$, if there is any.

All these tests can be performed in $O(\log k)$ time if we maintain an augmented balanced binary tree whose leaves are the values $\delta_1, \ldots, \delta_k$, where each internal node keeps track of the sum of the δ-values, and the minimum δ-value in its subtree. When inserting i to the left of the jth position, the effect of setting $\mu'_h = \mu_h + p_i$ for all $1 \leq h < j$, $\mu'_j = \max\{s_i, \mu_{j+1}\} + p_i$, and $\mu'_{h+1} = \mu'_h$ for all $j \leq h \leq k$ can be easily achieved by inserting a new value $\delta'_j = \mu'_j - \mu'_{j+1}$ and setting $\delta'_{j-1} = \mu'_{j-1} - \mu'_j$. The remaining values are left unchanged since $\delta'_h = \mu'_h - \mu'_{h+1} = \mu_h + p_i - \mu_h - p_i = \delta_h$ for $h \leq j - 2$ and $\delta'_h = \mu'_h - \mu'_{h+1} = \mu_{h-1} - \mu_{h-2} = \delta_{h-1}$ for $h > j$. Thus, in each iteration, the tree can be updated in $O(\log k)$ time as well. □

Acknowledgements. We would like to thank the anonymous referees for helpful suggestions.

References

1. Bartal, Y., Leonardi, S., Marchetti-Spaccamela, A., Sgall, J., Stougie, L.: Multiprocessor scheduling with rejection. SIAM J. Discrete Math. 13(1), 64–78 (2000)
2. Chrobak, M., Kenyon, C., Noga, J., Young, N.E.: Oblivious medians via online bidding. In: Proceedings of the 13th Latin American Symposium on Theoretical Informatics, pp. 311–322 (2006)
3. Engels, D.W., Karger, D.R., Kolliopoulos, S.G., Sengupta, S., Uma, R.N., Wein, J.: Techniques for scheduling with rejection. J. Algorithms 49(1), 175–191 (2003)
4. Epstein, L., Noga, J., Woeginger, G.J.: On-line scheduling of unit time jobs with rejection: minimizing the total completion time. Operations Research Letters 30(6), 415–420 (2002)
5. Gonzalez, T.F.: Clustering to minimize the maximum intercluster distance. Theoretical Computer Science 38, 293–306 (1985)
6. Hall, L.A.: Approximation Algorithms for NP-Hard Problems, ch. 2. PWS Publishing Company (1997)
7. Hoogeveen, H., Skutella, M., Woeginger, G.J.: Preemptive scheduling with rejection. Mathematical Programming 94(2-3), 361–374 (2003)
8. Lin, G., Nagarajan, C., Rajaraman, R., Williamson, D.P.: A general approach for incremental approximation and hierarchical clustering. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1147–1156 (2006)
9. Mettu, R.R., Plaxton, C.G.: The online median problem. SIAM Journal on Computing 32(3), 816–832 (2003)
10. Seiden, S.S.: Preemptive multiprocessor scheduling with rejection. Theor. Comput. Sci. 262(1), 437–458 (2001)
11. Sengupta, S.: Algorithms and approximation schemes for minimum lateness/tardiness scheduling with rejection. In: Proceedings of the 15th International Workshop on Algorithms and Data Structures, pp. 79–90 (2003)