DIMACS Technical Report 2000-34
December 2000

An Online Scheduling Problem with Job Set-ups

by

Srikrishnan Divakaran [1]
Department of Computer Science
Rutgers University
New Brunswick, NJ
divakara@paul.rutgers.edu

Michael Saks [2]
Department of Mathematics
Rutgers University
New Brunswick, NJ
saks@math.rutgers.edu

[1] Supported in Part by NSF grant CCR-9700239, NSF CCR 91-19999 and NSA MDA 904-97-C-0566 .
[2] Supported in Part by NSF grant CCR-9700239.

ABSTRACT

Machine scheduling and paging are two fundamental problems that have been studied extensively from the view of online algorithms. In both problems, one processes a sequence $J_1, \ldots, J_n$ of *jobs* (or *page requests*) that arrive over time. In the paging problem, requests must be answered in the order of arrival because the requests arise in the context of the execution of a sequential program. The need for decision making comes from the presence of a cache which can store a limited number of pages, where serving a cached page is considerably cheaper than an uncached one. Thus algorithms for paging are focused around the choice of caching policy. In scheduling, there is no cache, but the scheduler has (at least limited) freedom to reorder the jobs, and choosing the order is the major issue in the design of online scheduling algorithms. In computer systems, however, there are important situations where both caching and job reordering are available. One very natural setting where this happens is handling web page requests at a high volume web site. As in the usual paging problem, the server must manage a cache. But here the requests are (essentially) independent, so the server is free to reorder them.

In this paper, we begin the study of online algorithms for scheduling systems where both caching and reordering are available. We consider a simple, but already interesting and nontrivial, case: that of a single machine with a cache of unit size, and uniform page size. This particular special case is also natural in the usual machine scheduling context where jobs are classified into one of $\{1,2,...,F\}$ job types, and before processing a job, the machine must be configured (or set-up) according to the type of the job. This configuring (or set-up) is required unless the previously served job is of the same type as the current one. The current configuration of the machine in this scheduling problem, corresponds exactly to the contents of the unit cache in the page server problem.

In this context, we consider the online problem of scheduling a sequence $J = (J_1, J_2, ..., J_N)$ of jobs with release times, processing times and sequence independent set-up times, on a single machine, to minimize the maximum flow time. A partitioning of the jobs into F job types is given. A set-up is required at the start of each batch, where a batch is the largest set of contiguously scheduled jobs of the same job type. This problem is a special case of the maximum lateness problem, a problem known to be NP-hard even for very restrictive special cases and for which there are no known polynomial time $O(1)$ approximation algorithms. In this paper, we present a polynomial time $O(1)$-competitive online algorithm for the maximum flow time problem and also prove the NP-Hardness of the offline version of this problem.

1 Introduction

Machine scheduling and paging are two fundamental problems that have been studied extensively from the view of online algorithms. In both problems, one processes a sequence $J_1, \ldots, J_n$ of *jobs* (or *page requests*)that arrive over time. In the paging problem, requests must be answered in the order of arrival because the requests arise in the context of the execution of a sequential program. The need for decision making comes from the presence of a cache which can store a limited number of pages, where serving a cached page is considerably cheaper than an uncached one. Thus algorithms for paging are focused around the choice of caching policy. In scheduling, there is no cache, but the scheduler has (at least limited) freedom to reorder the jobs, and the choosing the order is a major issue in the design of online scheduling algorithms. In computer systems, however, there are important situations where both caching and job reordering are available. One very natural setting where this happens is handling web page requests at a high volume web site. As in the usual paging problem, the server must manage a cache. But here the requests are (essentially) independent, so the server is free to reorder them.

In this paper, we begin the study of online algorithms for scheduling systems where both caching and reordering are available. We consider a simple, but already interesting and nontrivial, case: that of a single machine with a cache of unit size, and uniform page size. This particular special case is also natural in the usual machine scheduling context. Consider a situation where jobs are classified into one of $\{1,2,...,F\}$ types, and before processing a job, the machine must be configured according to the type of the job. This configuring (or set-up) is required unless the previously served job is of the same type as the current one. The current configuration of the machine in this scheduling problem, corresponds exactly to the contents of the unit cache in the page server problem. In the language of machine scheduling, each arriving job J_i belongs to one of F types $\{1,2,...,F\}$, has a release time r_i and requires an integer processing time of p_i. For each job type $f \in \{1,...,F\}$, there is a set-up time s_f, which is the time needed to configure M for processing a job of type f. This set-up is required each time a job of type f is processed, unless the most recently processed job was also of type f, in which case the set-up time is 0. The incoming jobs need not be processed in the sequence in which they are released.

For such a system, a schedule for a job sequence $J = (J_1, J_2, ..., J_N)$ specifies for each job in J, a time interval during which M performs the set-up and the processing of that job. For a given schedule σ for the job sequence J, the flow time of a job is the difference between the completion time and the release time of that job and the maximum flow time is the maximum among the flow times of the jobs in J. In this paper, we focus on the *maximum flow time problem*, which is :

Given a job sequence J, determine a schedule that minimizes the maximum flow time.

One application for this problem is a simple file server system that is presented with a sequence of requests for files that arrive over a period of time, each with a known processing time (the time to read the file from memory and send it to the requester). If the requested file is in memory, the file may be sent instantly; otherwise the contents of the memory are abandoned and a set-up time is suffered to load the file to memory. This problem differs from the usual caching problem because requests need not be processed in order of arrival. The file server needs to perform a set-up every time it switches processing from one file to another. So, in order to avoid spending a lot of time on set-ups, the file server can attempt to group similar requests into batches and thereby reduce the total time spent on set-ups. On the other hand, serving requests in large batches may result in certain requests to be processed after a long wait and thereby increasing the maximum amount of time any job spends in the system. In this case, the waiting can be reduced by processing requests in smaller batches so as to accommodate requests that have been waiting for a long time. So we need to schedule requests so as to maintain a balance between the time spent on set-ups and the maximum amount of time any request spends in the system.

1.1 Related Work

The maximum flow time problem can be formulated as a special case of the *maximum lateness problem* [3,6,10], which is

> Given a job sequence $J = (J_1, ..., J_N)$, where job J_i is one of F types $\{1, ..., F\}$, has a release time $r_i \geq 0$ (a non-negative integer), processing time p_i and a due date $d_i \geq 0$ (a non-negative integer), determine a schedule in $\Phi(J)$ that minimizes the *maximum lateness*, where, for a schedule $\sigma \in \Phi(J)$, the lateness of a job is defined to be the maximum of 0 and the difference between the completion time and due date of that job and the maximum lateness is the maximum among the lateness of the jobs in J.

The maximum flow time problem can be formulated as a special case of the maximum lateness problem [3,6,10] when the due date of a job equals its release time. Lenstra, Rinnooy Kan and Brucker [10] showed that the maximum lateness problem is *NP*-hard, even if the set-up time for all job types is 0, and Bruno and Downey [3] showed that the maximum lateness problem for an arbitrary F (i.e. number of job types) is NP-hard even when all jobs have the same release dates. Later, Monma and Potts [6] presented an $O(F^2N^{2F})$ time dynamic programming algorithm for the above problem. Unfortunately, when F is large their dynamic programming approach is not of much practical use. Due to the impracticality of the dynamic programming algorithm of Monma and Potts, Zdarka [14,15], Hariri and Potts [8] proposed heuristics for the above problem under the assumption that all due dates are non-positive. However, in the presence of non-trivial (i.e. unequal) release times, the assumption that all jobs have non-positive due dates restricts the practical use of these heuristics. Schutten et al. [11,12] developed a branch and bound procedure for the maximum lateness problem when the release times are non-trivial. Though their branch and bound procedure performs well in solving small instances of the problem, they do not provide any guarantees regarding the performance of their procedure and also their procedure can only be used to solve the offline version of the problem.

Our problem can also be viewed as a special case of a dynamic version of the traveling salesman problem. In this version, each job is represented by a node in a directed graph with edge and node weights. The weight of the node is the processing time of the job and the weight of a directed edge (v, w) is the time needed to reconfigure the machine to service w immediately after v. The graph grows dynamically with a node being added when the corresponding job is released. The dynamic version of the TSP problem has not been considered in this generality, however a related model was considered in [5]. In their model [5], given a partial order on n nodes each of which is labelled by one of k points in a metric space, they try to find a visit order consistent with precedence constraints that minimizes the total cost of the corresponding path in the metric space. Notice that our problem is a special case of the dynamic TSP problem where the cost of an edge (v,w) is a function of only node w, whereas in [5] the directed graph is static and the cost of a directed edge (v,w) is a function of both v and w. In addition, they allow precedence constraints between jobs. For this model, they show that there is no k^{α} approximation algorithm for some $\alpha > 0$, unless $P = NP$.

1.2 Our Results

We first present a polynomial time $O(1)$ competitive online algorithm for the maximum flow time problem. Then, we give a proof of NP-Hardness of the offline version of the maximum flow time problem.

1.3 Contents

In Sections 2, we describe the maximum flow time problem. In section 3 we present three natural but non-competitive algorithms for the maximum flow time problem. In section 4, we present an $O(1)$-competitive online algorithm for the maximum flow time problem. In section 5, we present the analysis of our algorithm. In section 6, we present the proof of hardness for the maximum flow time problem. Due to space restrictions, we include some of our proofs and analysis in Appendix A, B and C.

1.4 Notation

The following are some of the notation that we use in this paper :

N	= the number of jobs;
F	= the number of job types;
J	= $(J_1,\ldots,J_N)$ a job sequence;
h,i,j	= job indices. $1 \le h,i,j \le N$;
f	= a job type in $\{1,\ldots,F\}$;
J_i	= the i^{th} job in J;
s_f	= the set-up time for jobs of type f;
$J_{[t_1,t_2]}$	= the subsequence of jobs in J that were released in the time interval $[t_1,t_2]$;
$J^{<f>}$	= the subsequence of jobs in J that are of type f;
$\Phi(J)$	= the set of all valid schedules for J;
σ	= a schedule in $\Phi(J)$;
$OPT(J)$	= the schedule in $\Phi(J)$ with the optimal maximum flow time ;
$G(J)$	= the schedule determined by algorithm G for the job sequence J;

2 Problem Description

We consider a machine M that can process jobs of F different *types* $\{1,...,F\}$. For each job type $f \in \{1,...,F\}$, s_f is the time needed to configure M for processing a job of type f. The system gets a sequence $J = (J_1,...,J_N)$ of jobs, where J_i is of type $f_i \in \{1,...,F\}$, is released at time $r_i \geq 0$ (a non-negative integer) and requires an integer processing time of p_i. We consider the problem of designing a scheduling algorithm that assigns jobs to M. A job can be assigned to M only after it is released and M is idle. Once the job J_i is assigned, M immediately services J_i. The servicing of J_i consists of a set-up stage where M is configured for processing a job of type f_i, immediately followed by a processing stage. The set-up for J_i takes 0 time units if the previously serviced job was of type f_i and takes s_{f_i} otherwise. The processing stage takes p_i time units.

A schedule for a job sequence J specifies for each job $J_i \in J$, the time $A_i^\sigma \geq r_i$, at which the servicing of job J_i begins. Let $\Phi(J)$ represent the set of valid schedules for J. For any given schedule $\sigma \in \Phi(J)$, we can compute for each job J_i :

set-up time $S_i^\sigma = \begin{cases} 0 & \text{if at time } A_i^\sigma \text{, the most recently serviced job is of type } f_i; \\ s_{f_i} & \text{otherwise;} \end{cases}$
starting time $B_i^\sigma = A_i^\sigma + S_i^\sigma$;
completion time $C_i^\sigma = B_i^\sigma + p_i$;
flow time $F_i^\sigma = C_i^\sigma - r_i$;

Note that the condition for a schedule to be valid is that the time intervals $[A_i^\sigma, C_i^\sigma]$ are disjoint and $A_i^\sigma \geq r_i$ for all i. For any schedule $\sigma \in \Phi(J)$, we define F_{max}^σ to be the *maximum flow time* of J (i.e. $F_{max}^\sigma = max_{i \in N} F_i^\sigma$). Our goal in scheduling is to minimize the maximum flow time.

A scheduling algorithm G assigns to each job sequence J, a schedule $G(J) \in \Phi(J)$. In addition, if the jobs in J are revealed to G only at the time they are released and at all times G makes its decisions only based on the past information. (i.e. knowledge of jobs that have been released) , then G is an online scheduling algorithm. More formally, for $t \geq 0$ if two job sequences J and $\hat{J}$ are identical up to time t (i.e. same job types, processing times and job release times) then the schedules assigned by G to J and $\hat{J}$ will be identical up to time t.
A schedule σ is optimal for J if $F_{max}^\sigma \leq F_{max}^\phi$ for any other schedule $\phi \in \Phi$. It is easy to show that every sequence has an optimal schedule. Let OPT be an algorithm (not necessarily online) that assigns to each sequence J an optimal schedule for J.
Let G be some arbitrary online scheduling algorithm for the maximum flow time problem. For $c \in R$, G is c-competitive if there is a constant α such that for any job sequence J, $F_{max}^{G(J)} \leq c * F_{max}^{OPT(J)} + \alpha$. In this paper, our objective is to design an $O(1)$-competitive online scheduling algorithm for the maximum flow time problem.

3 Three natural but non-competitive online algorithms

In this section, to help motivate our competitive algorithm, we present three natural but non-competitive online algorithms for the maximum flow time problem. All of these algorithms have the same basic structure. At each time t, we maintain

$Q(t)$ - the subsequence of jobs in $J_{[0,t]}$ that have not been serviced;
$Q^{<f>}(t)$- the subsequence of $Q(t)$ consisting of jobs of type f;

We also maintain for each job type $f \in \{1,...,F\}$ a real number priority, $priority^{<f>}(t)$ (which will depend on the particular algorithm). At any time t, let $high(t)$ be the job type with the highest non-negative priority. If there are no job types with non-negative priority then $high(t) = null$. The algorithm is given by

Procedure *Main*
while (true)
{
  If (M is busy) or ($Q(t) = \emptyset$) then continue;
  If ($high(t) \neq$ null) then
    choose the job of type $high(t)$ with the earliest release time as the next job for servicing;
}

That is, at any time t, when M is free (i.e. not servicing any job) *Main* chooses the job of type $high(t)$ with the earliest release time as the next job for servicing. Now, depending on how we define $priority^{<f>}(t)$, for $t > 0$ and $f \in \{1,...,F\}$, we obtain different algorithms for the maximum flow time problem. We now consider three natural choices for defining $priority^{<f>}(t)$. For this we need to maintain two additional parameters :

$active(t)$ - the job type of the most recently serviced job;
$oldest^{<f>}(t)$- the release time of the job in $Q^{<f>}(t)$ with the earliest release time. If $Q^{<f>}(t)$ is empty then $oldest^{<f>}(t) = \infty$;

Algorithm **Earliest-First** : For $f \in \{1,...,F\}$ and $t > 0$, $priority^{<f>}(t) = t - oldest^{<f>}(t)$.

In this algorithm, jobs are processed in the increasing order of their release times. If the set-up time for all job types is 0 (i.e. $s_f = 0$, for all $f \in \{1,...,F\}$), then it is easy to observe that the schedule determined by **Earliest-First** is optimal. However, if the set-up times are non-trivial (i.e. $s_f \neq 0$, for some $f \in \{1,...,F\}$), the following example will show that **Earliest-First** does not have a finite competitive ratio.
Example 1. Consider the sequence $J = (J_1, J_2, J_3, J_4)$ of jobs, where all jobs in J have a unit processing time and all job types have a unit set-up time. Let the Jobs J_1, J_3 be of type 1 and have release times 1 and 3 respectively. Let the jobs J_2, J_4 be of type 2 and have release times 2 and 4 respectively.

It is easy to observe that in the optimal schedule, jobs will be processed in the sequence (J_1, J_3, J_2, J_4), the maximum flow time of any job in J will be 4 and all jobs in J will be completed by the end of the 6^{th} time unit. However, for the schedule determined by **Earliest-First**, jobs will be processed in the sequence (J_1, J_2, J_3, J_4), the maximum flow time of any job in J will be 5 and all jobs in J will be completed only at the end of the 9^{th} time unit. Now, if the input job sequence J is repeated every

6 time units, it is easy to observe that the maximum response time in the optimal schedule will still be 4, but the maximum response time in the schedule determined by **Earliest-First** will increase by 2 every 6 time units. Hence, we can conclude that **Earliest-First** does not have a finite competitive ratio for the maximum flow time problem.

From the above example, we observe that **Earliest-First** does not perform well since it does not try to avoid time spent on set-ups by batching jobs of the same type. So, we will now consider an algorithm that will try to avoid set-up by batching jobs of the same type.

Algorithm **Avoid-Setup** : For $f \in \{1,...,F\}$ and $t > 0$,

$$priority^{<f>}(t) = \begin{cases} \infty & \text{if } f = active(t) \text{ and } Q^{<f>}(t) \neq \emptyset \\ t - oldest^{<f>}(t) & \text{otherwise} \end{cases}$$

In this algorithm, at any time t, M tries to avoid time spent on set-ups by continuing to service jobs of type $active(t)$ until there are no more jobs to service. At this time, M switches to the job type with the oldest job (i.e. the unserviced job with the earliest release time).

Example 2 . Consider the sequence $J = (J_1, J_2, ..., J_N)$ of jobs, where all jobs in J have a unit processing time and all job types have a unit set-up time. Let job J_2 be of type 2 and all other jobs be of type 1. For $i \in \{1,...,N\}$, let i be the release time of J_i.

It is easy to observe that in an optimal schedule, the maximum flow time is at most 4. However, for the schedule determined by **Avoid-Setup**, the maximum flow time is N because J_2 will be processed only after all the other jobs. Thus, **Avoid-Setup** does not have a bounded competitive ratio.

From the above example, we observe that putting highest priority on avoiding set-up can lead to starving some jobs. So, we will now try to consider a algorithm that tries to maintain a balance between avoiding set-ups and avoiding long delays in the servicing of any particular job. The following algorithm depends on a non-negative parameter τ.

Algorithm **Reduce-Setup**(τ) : At any time t, for $f \in \{1,...,F\}$,

$$priority^{<f>}(t) = \begin{cases} t - oldest^{<f>}(t) + \tau & \text{if } f = active(t) \text{ and } Q^{<f>}(t) \neq \emptyset \\ t - oldest^{<f>}(t) & \text{otherwise} \end{cases}$$

It is easy to observe that **Earliest-First** is a special case of **Reduce-Setup**(τ) when $\tau = 0$ and **Avoid-Setup** is a special case of **Reduce-Setup**(τ) when $\tau = \infty$. In this algorithm, at any time t, M continues to process jobs of the type $active(t)$ until the release time of the oldest unprocessed job of type $active(t)$ exceeds the release time of the oldest job of any other type by more than τ time units. At this time, M switches to the job type with the oldest job. In this algorithm, the parameter τ helps in restricting the batch size so as to accommodate jobs that have been waiting for a long time. However, for non-trivial set-up times (i.e. $s_f \neq 0$, $f \in \{1,...,F\}$), the following example will show that **Reduce-Setup**(τ) does not have a finite competitive ratio.

Example 3. Consider the sequence $J = (J_1, J_2, ..., J_{\tau+3})$ of jobs, where the set up time for all job types is 2 (i.e. $s_f = 2$ for all $f \in \{1,...,F\}$) and for $i \in \{1,...,\tau+3\}$, job J_i has a release time $r_i = i$

and a processing time $p_i = 1$. The job J_2 is of type 2 and all other jobs are of type 1.

It is easy to observe that in the optimal schedule, jobs will be processed in the sequence $(J_2, J_1, J_3, J_4, ..., J_{\tau+3})$, the maximum flow time of any job in J will be 7 and all jobs in J will be completed by the end of the $\tau + 9^{th}$ time unit. However, for the schedule determined by **Reduce-Setup**(τ), jobs will be processed in the sequence $(J_1, J_3, J_4, ..., J_{\tau+2}, J_2, J_{\tau+3})$, the maximum flow time will be $\tau + 2$ and all jobs in J will be completed only at the end of the $\tau + 10^{th}$ time unit.

Now, if the input job sequence J is repeated every $\tau + 8$ time units except that the jobs of type 1 and 2 are replaced by jobs of two new types i and j respectively, then we can observe that the maximum flow time of the optimal schedule will still remain the same whereas the maximum flow time of the schedule determined by **Reduce-Setup**(τ) will increase by 1 every $\tau + 8$ time units. Therefore, we can conclude that **Reduce-Setup**(τ) does not have a finite competitive ratio for the maximum flow time problem.

In the above example **Avoid-Setup**(τ) processes J_1 before J_2. Once this happens, either J_2 must be delayed until the completion of all other jobs, or we must split the jobs of type 1 into two batches and pay for two set-ups. However, the optimal algorithm processes J_2 before J_1, to avoid either of these bad occurrences. Here job type 1 occurs frequently while job type 2 is isolated in the sense that no other jobs of the same type are released. In this case it seems that we want to complete the isolated jobs first to avoid extra set-ups for the frequent jobs.

4 An $O(1)$-competitive online algorithm

In this section, we will first present an online algorithm $G_{(\tau_1,\tau_2)}$ defined in terms of two real number parameters τ_1 and τ_2. Then, using this algorithm as a building block, we will define an $O(1)$-competitive online algorithm for the maximum flow time problem.

4.1 Building Block $G_{(\tau_1,\tau_2)}$

In this section, we present an online algorithm $G_{(\tau_1,\tau_2)}$ that tries to fix the problem faced by the previous algorithm by identifying jobs that are isolated (say a job J_i is *isolated* if no other job of type f_i was released in the interval $[r_i, r_i + \tau_1]$). Notice that we cannot know that J_i is isolated until time $r_i + \tau_1$. So J_i is known to be isolated at time t if it is isolated and $r_i + \tau_1 < t$. For this reason we define at any time t and for $f \in \{1, ..., F\}$, the set of jobs $J_i \in Q^{<f>}(t)$ that are known to be isolated at time t. More formally, at any time t and for $f \in \{1, ..., F\}$, we define

$isolated^{<f>}(t)$ = { $J_i \in Q^{<f>}(t)$, such that there is no job of type f that was released within τ_1 time units after the release of J_i and $r_i + \tau_1 < t$};

$firstgap^{<f>}(t)$ = is the release time of the job in $isolated^{<f>}(t)$ with the earliest release time. If $isolated^{<f>}(t) = \emptyset$ then $firstgap^{<f>}(t) = \infty$;

At any time t, we define a job type f to be *active* if $f = active(t)$. At any time t, we define a job type f to be *isolated* if f is not *active*,and there is at least one known isolated job of type f (i.e.

$isolated^{<f>}(t) \neq \emptyset$) and the release time of the oldest isolated job of type f is within $(\tau_2 - \tau_1)$ units of the release time of the oldest unserviced job of type f (i.e. $firstgap^{<f>}(t) \leq oldest^{<f>}(t) + \tau_2 - \tau_1$), otherwise f is *non-isolated.*

At any time t and for $f \in \{1, ..., F\}$, $G_{(\tau_1,\tau_2)}$ associates with each job type f a priority

$$priority^{<f>}(t) = \begin{cases} t - oldest^{<f>}(t) & \text{if } f \text{ is } \textit{active} \\ t - (firstgap^{<f>}(t) + \tau_1) & \text{if } f \text{ is } \textit{isolated} \\ t - (oldest^{<f>}(t) + \tau_2) & \text{otherwise} \end{cases}$$

In this algorithm, the priority of a job type f is calculated differently depending on whether it is *active*, *isolated* or *non-isolated.* Thus at time t, the priority of an active job type is determined by the release time of the oldest unprocessed job of that type, the priority of an isolated job type is determined by the parameter τ_1 and the release time of the oldest isolated job of that type, and the priority of other job types are determined by the parameter τ_2 and the release time of the oldest unprocessed job of that type. The reason for defining the priority in this way is to avoid extra set-ups by either trying to delay the processing of non-isolated jobs until the existing isolated jobs have completed or delay the processing of isolated jobs until a long batch of non-isolated jobs is completed. Here, the parameter τ_1 and τ_2 help in maintaining a balance between avoiding set-ups and avoiding long delays in any particular job.

For $\Lambda > 0$, let J_Λ be the set of all job sequences whose optimal value for the maximum flow time is at most Λ. We now show that for any Λ, we can choose τ_1 and τ_2 so that, for any $J \in J_\Lambda$ and the online schedule $G_{(\tau_1,\tau_2)}(J)$, the maximum flow time of any job in J is at most $O(\Lambda)$.

Theorem 1: Let $\lambda = \max_{f\in\{1,...,F\}} s_f$, $\Lambda \geq \lambda$, $\tau_1 \geq (\Lambda + \lambda)$ and $\tau_2 \geq 2(\Lambda + \lambda)$, then for any $J \in J_\Lambda$, $F_{max}^{G_{(\tau_1,\tau_2)}(J)} \leq \tau_1 + 2\tau_2 \leq 5(\Lambda + \lambda) \leq 10\Lambda$

The proof of Theorem 1 is given in Section 5.1.

4.2 An $O(1)$-competitive online algorithm H

Using Theorem 1, we now easily construct a competitive algorithm H that works without assuming an upper bound Λ, using a standard 'doubling technique'.

Algorithm $G_{(\tau_1,\tau_2)}$ guarantees flow time $\leq \tau_1 + 2\tau_2$ provided that the given job sequence has optimal cost at most Λ. We now modify $G_{(\tau_1,\tau_2)}$ by adding the following premature termination condition :

> For each job to be serviced, before servicing it, compute the time t^* it will complete. If at t^* there will be an unserviced job J_i that will have $t^* - r_i > \tau_1 + 2\tau_2$ then the algorithm exits without serving the next job.

Now, we will introduce certain notions that will be help define algorithm H.

DEFINITION 4.1: Suppose we are running a scheduling algorithm on an input sequence J and let t be a time at which some job in J is completed. The *residual job sequence* at time t is defined as the subsequence $J^{(t)}$ consisting of those jobs in J that have not been serviced by time t. We then obtain the *adjusted residual job sequence* $\bar{J^{(t)}}$ from $J^{(t)}$ by changing each release time r_i to $r_i + (t - r_{min})$, where r_{min} is the release time of the job in $J^{(t)}$ with the earliest release time.

Algorithm H
Let $i = 1$, $t_1 = 0$ and $\bar{J^{(t_1)}} = J$;
Let $\Lambda_1 = \lambda$, $\tau_1^1 = \Lambda_1 + \lambda$ and $\tau_2^1 = 2(\Lambda_1 + \lambda)$;
while ($\bar{J^{(t_i)}} \neq \emptyset$)
{
  Execute algorithm $G_{(\tau_1^i, \tau_2^i)}$ for sequence $\bar{J^{(t_i)}}$;
  Let $i = i + 1$;
  Let t_i be the most recent time at which some job in J completed;
  Let $\bar{J^{(t_i)}}$ be the adjusted residual job sequence at time t_i;
  Let $\Lambda_i = 2^{i-1}\lambda$, $\tau_1^i = \Lambda_i + \lambda$ and $\tau_2^i = 2(\Lambda_i + \lambda)$;
}

Starting at time $t_1 = 0$, the algorithm H services the jobs in J in stages. For $i \in N$, during stage i, H executes algorithm $G_{(\tau_1^i, \tau_2^i)}$ for the adjusted residual job sequence $\bar{J}^{(t_i)}$. We now show that H is an $O(1)$ competitive online algorithm.

Theorem 2: The algorithm H is 20 competitive.

Proof . Let h be the stage during which H completes servicing all jobs in J. If $h = 1$, then clearly $F_{max}^{H(J)} \leq 5(\Lambda_1 + \lambda) \leq 20F_{max}^{OPT(J)}$. Otherwise, $h > 1$. Since H entered stage h it must have exited stage $h-1$. Now, we will get a lower bound on $F_{max}^{OPT(J)}$. Let K be the subsequence of jobs in $J^{(t_{h-1})}$ that were serviced by $G_{(\tau_1^{h-1}, \tau_2^{h-1})}$. Let $\bar{K}$ be the subsequence of jobs in $\bar{J}^{(t_{h-1})}$ that were serviced by $G_{(\tau_1^{h-1}, \tau_2^{h-1})}$. Let $\bar{\sigma}$ denote the online schedule $G_{(\tau_1^{h-1}, \tau_2^{h-1})}(\bar{K})$. σ^* the optimal schedule $OPT(K)$ and $\bar{\sigma}^*$ denote the optimal schedule $OPT(\bar{K})$. Since H exited stage $h-1$, we know that $F_{max}^{\bar{\sigma}} > \tau_1^{h-1} + 2\tau_2^{h-1}$. From this and Theorem 1, we get $F_{max}^{\bar{\sigma}^*} \geq \Lambda_{h-1}$. From the definition of K and $\bar{K}$, $F_{max}^{\sigma^*} = F_{max}^{\bar{\sigma}^*}$. Therefore, from the previous two observations, we get $F_{max}^{OPT(J)} \geq F_{max}^{\sigma^*} \geq \Lambda_{h-1}$. Now, we will get an upper bound on $F_{max}^{H(J)}$. Consider a job that completed during stage $i \leq h$. By definition of $G_{(\tau_1^i, \tau_2^i)}$ the flow time of that job with respect to its adjusted release time in $\bar{J}^{(i)} \leq \tau_1^i + 2\tau_2^i \leq 5(\Lambda_i + \lambda)$. To get the actual flow time we must add in the total adjustment to the release time over the previous stages. From the definition of H, we observe that during stage i, for $i \in \{1, .., h-1\}$, the adjustment of release time for any job in J is at most $\tau_1^i + 2\tau_2^i \leq 5(\Lambda_i + \lambda)$. Therefore, $F_{max}^{H(J)} \leq 5(\Lambda_h + \lambda) + 5\sum_{j \in [1, h-1]}(\Lambda_j + \lambda) \leq 5(\Lambda_{h+1} + \lambda)$. This implies that $F_{max}^{H(J)} \leq 5(\Lambda_{h+1} + \lambda) \leq 20F_{max}^{OPT(J)} + 5\lambda$. □
.

5 Analysis of Algorithm $G_{(\tau_1,\tau_2)}$

For reference, here are some terms that are often used during the analysis of our algorithm.

$i\downarrow^{\sigma}$ = the index of the job in J that was serviced just before J_i in the schedule σ;
A_i^{σ} = the time at which the servicing of job J_i begins;
S_i^{σ} = the set-up time for job J_i in the schedule σ;
B_i^{σ} = $A_i^{\sigma}+S_i^{\sigma}$ the time at which the processing of job J_i begins in the schedule σ;
C_i^{σ} = $B_i^{\sigma}+p_i$ the completion time of job J_i in the schedule σ;
I_i^{σ} = $A_i^{\sigma}-C_{i\downarrow}^{\sigma}$ the idle time of job J_i in the schedule σ (i.e. the difference between the time at which the servicing of job J_i begins and the completion time of its predecessor $J_{i\downarrow}$ in the schedule σ);
F_i^{σ} = $C_i^{\sigma}-r_i$ the flow time of job J_i in schedule σ;
F_{max}^{σ} = $\max_{i\in\{1,\ldots,N\}}\{C_i^{\sigma}-r_i\}$ the maximum flow time for the schedule σ;
$S^{\sigma}(J)$ = $\sum_{J_h\in J}S_h^{\sigma}$ the total time spent on set-ups for jobs in J in the schedule σ;
$P^{\sigma}(J)$ = $\sum_{i\in\{1,\ldots,N\}}p_i$ the sum of the processing times of jobs in J;
$I^{\sigma}(J)$ = $\sum_{J_h\in J}I_h^{\sigma}$ the total idle time of jobs in J in the schedule σ;

Throughout the analysis, the job sequence $J=(J_1,\ldots,J_N)\in J_{\Lambda}$ is fixed.

5.1 Proof of Theorem 1

Proof . Let Λ, τ_1 and τ_2 be as in the hypothesis of Theorem 1 and σ^* denote the optimal schedule $OPT(J)$ and σ the online schedule $G_{(\tau_1,\tau_2)}(J)$ and let $c=\tau_1+2\tau_2$. For $t>0$, we want to show that for the online schedule σ, by time t, the machine M would have serviced all jobs in $J_{[0,t-c]}$. For the online schedule σ, let $\hat{t}$ be the last time prior to t, when M was idle (i.e. M was free but no job was assigned to M for servicing). By definition of algorithm $G_{(\tau_1,\tau_2)}$, M is idle at $\hat{t}$ only if all jobs prior to $\hat{t}-\tau_2$ have completed. Thus at time $\hat{t}$, the entire sequence $J_{[0,\hat{t}-\tau_2]}$ has been serviced, so if $\hat{t}\geq t-(\tau_1+\tau_2)$, we are done. Thus we may assume $\hat{t}<t-(\tau_1+\tau_2)$. We divide the analysis into two cases.

case 1 : By time t, M has started servicing at least one job in $J_{[t-(\tau_1+\tau_2),t]}$
From the definition of $G_{(\tau_1,\tau_2)}$, we know that M services jobs in J in non-increasing order of priority value and at all times the priority value of any job in $J_{[0,t-(\tau_1+2\tau_2)]}$ is higher than the priority value of any job in $J_{[t-(\tau_1+\tau_2),t]}$. Therefore by time t, if M had started servicing any job in $J_{[t-(\tau_1+\tau_2),t]}$ then M would have also serviced all jobs in $J_{[0,t-(\tau_1+2\tau_2)]}$. Since, $c=\tau_1+2\tau_2$, by time t, M would have serviced all jobs in $J_{[0,t-c]}$.

case 2 : By time t, M has not started servicing any job in $J_{[t-(\tau_1+\tau_2),t]}$
Let K represent the segment of jobs in J that arrived in the time interval $[\hat{t}-\tau_2,t-(\tau_1+\tau_2)]$ (i.e. $K=J_{[\hat{t}-\tau_2,t-(\tau_1+\tau_2)]}$) and K^* represent the subsequence of jobs in J that was serviced in the optimal schedule σ^* during the time interval $[\hat{t}-\tau_2,t-(\tau_1+\tau_2)+\Lambda]$. From the definition of K and K^* and the fact that σ^* has a maximum flow time of Λ, $K\subseteq K^*$.

The following lemma relates the total set-up time of jobs in K for the online schedule σ to the total set-up and idle time of jobs in K^* for the optimal schedule σ^*.

Lemma 3: Let $K = (J_i, ..., J_j)$ be some arbitrary segment of J and K^* be the subsequence of jobs in J that was serviced in the optimal schedule σ^* during the time interval $[r_i, r_j + \Lambda]$. Then, $S^{\sigma^*}(K^*) + I^{\sigma^*}(K^*) + \lambda \ \geq \ S^{\sigma}(K)$

Assuming the lemma we complete the proof of the theorem. Suppose for contradiction that under σ, M has not serviced all jobs in K by time t. Then, for the online schedule σ, since M does not idle during the time interval $[\hat{t}, t]$, we get

$$P^{\sigma}(K) + S^{\sigma}(K) \ > \ t - \hat{t} \tag{1}$$

Also, since $K \subseteq K^*$, we get

$$P^{\sigma^*}(K^*) \ \geq \ P^{\sigma^*}(K) \ = \ P^{\sigma}(K) \tag{2}$$

From the definition of K^*, we get

$$P^{\sigma^*}(K^*) + S^{\sigma^*}(K^*) + I^{\sigma^*}(K^*) \ \leq \ t - (\tau_1 + \tau_2) + \Lambda - (\hat{t} - \tau_2) \ \leq \ t - \hat{t} - \tau_1 + \Lambda$$

Substituting for $(t - \hat{t})$ and $P^{\sigma^*}(K^*)$ from inequalities (1) and (2) respectively, we get

$$S^{\sigma^*}(K^*) + I^{\sigma^*}(K^*) \ < \ S^{\sigma}(K) - \tau_1 + \Lambda = S^{\sigma}(K) - \lambda \tag{3}$$

The Inequality (3) contradicts Lemma 3, which proves the theorem. □

5.2 Proof of Lemma 3

Proof . We prove the lemma by partitioning K and K^* into subsequences according to the job type. For $f \in \{1, ..., F\}$, let α_f be the number of set-ups performed by M in the online schedule σ, for jobs in $K^{<f>}$ (i.e. jobs in K of type f).

By definition,

$$S^{\sigma}(K) = \sum_{f \in \{1,...,F\}} S^{\sigma}(K^{<f>}) = \sum_{f \in \{1,...,F\}} s_f * \alpha_f \tag{4}$$

$$S^{\sigma^*}(K^*) + I^{\sigma^*}(K^*) = \sum_{f \in \{1,...,F\}} (\ S^{\sigma^*}(K^{*<f>}) + I^{\sigma^*}(K^{*<f>})\ ) \tag{5}$$

The lemma follows immediately from the following claim, by summing over f. □

Claim . For the optimal schedule σ^*, let f^* be the job type of the most recently serviced job at time r_i. For the subsequence of jobs K^* and any job type $f \in \{1, ..., F\}$,

$$S^{\sigma^*}(K^{*<f>}) + I^{\sigma^*}(K^{*<f>}) \geq \begin{cases} s_f * (\alpha_f - 1) & \text{if } f = f^* \\ s_f * \alpha_f & \text{otherwise} \end{cases}$$

To prove the claim, fix a job type $\bar{f}$, and let $\alpha = \alpha_{\bar{f}}$. Now, we will introduce certain definitions that will be useful in proving the claim.

DEFINITION 5.1: For the online schedule σ, let $t_1, t_2, ..., t_\alpha$ be the times at which M performs a set-up for a job in $K^{<\bar{f}>}$ (i.e. a job in K of type $\bar{f}$), and let $\pi(h)$, $1 \leq h \leq \alpha$, be the index between i and j of the job in J for which M is performing the set-up at time t_h. We use $J_{\pi(1)}, J_{\pi(2)}, ..., J_{\pi(\alpha)}$ to partition the sequence K^* into $\alpha + 1$ subsequences $D(0), D(1), ..., D(\alpha)$, where $D(0)$ is the subsequence of jobs in K^* that were released on or before the release of job $J_{\pi(1)}$, for $1 \leq h < \alpha$, $D(h)$ is the segment of jobs $(J_{\pi(h)+1}, ..., J_{\pi(h+1)})$, and $D(\alpha)$ is the subsequence of jobs in K^* that were released after the release of job $J_{\pi(\alpha)}$.

Now, for the optimal schedule σ^*, we evaluate the total set-up and idle time of jobs in $K^{*<\bar{f}>}$ (i.e. jobs of type $\bar{f}$ in K^*), by summing over $h \in \{0, 1, ..., \alpha\}$, the total set-up and idle time of jobs of $D^{<\bar{f}>}(h)$ (i.e. jobs of type $\bar{f}$ in $D(h)$). More formally,

$$\begin{aligned} S^{\sigma^*}(K^{*<\bar{f}>}) + I^{\sigma^*}(K^{*<\bar{f}>}) &= \sum_{h \in [0,\alpha]} S^{\sigma^*}(D^{<\bar{f}>}(h)) + \sum_{h \in [0,\alpha]} I^{\sigma^*}(D^{<\bar{f}>}(h)) \\ &\geq S^{\sigma^*}(D^{<\bar{f}>}(0)) + \sum_{h \in [1,\alpha-1]} (\, S^{\sigma^*}(D^{<\bar{f}>}(h)) + I^{\sigma^*}(D^{<\bar{f}>}(h)) \,) \qquad (6) \end{aligned}$$

Now, we bound $S^{\sigma^*}(D^{<\bar{f}>}(0))$ by using the following lemma :

Lemma 4: If $(\bar{f} \neq f^*)$ then $S^{\sigma^*}(D^{<\bar{f}>}(0)) \geq s_{\bar{f}}$

For $h \in \{1, ..., \alpha - 1\}$, we bound $S^{\sigma^*}(D^{<\bar{f}>}(h)) + I^{\sigma^*}(D^{<\bar{f}>}(h))$ by using the following lemma :

Lemma 5: For any $h \in \{1, ..., \alpha - 1\}$, $S^{\sigma^*}(D^{<\bar{f}>}(h)) + I^{\sigma^*}(D^{<\bar{f}>}(h)) \geq s_{\bar{f}}$

Now, evaluating (6) using Lemma 4 and Lemma 5, the claim follows immediately.

Proof of Lemma 4. Without loss of generality we assume that in the optimal schedule σ^* all jobs of the same type are processed in the non-decreasing order of their release times. From definition we know that job $J_{\pi(1)} \in D^{<\bar{f}>}(0)$ and has a release time of $r_{\pi(1)} \geq r_i$. For σ^*, at time r_i, if the most recently serviced job is of type $f^* \neq \bar{f}$, then M will have to perform at least one set -up before processing $J_{\pi(1)}$. Therefore, if $\bar{f} \neq f^*$, then $S^{\sigma^*}(D^{<\bar{f}>}(0)) \geq s_{\bar{f}}$. □

Proof of Lemma 5. Let h be some arbitrary integer in $[1, \alpha - 1]$. Our objective is to show that for the optimal schedule σ^*, either the total set-up time for jobs in $D^{<\bar{f}>}(h)$ (i.e. jobs of type $\bar{f}$ in $D(h)$) is greater than $s_{\bar{f}}$ or or the total idle time of jobs in $D^{<\bar{f}>}(h)$ is greater than $s_{\bar{f}}$. More formally, we want to show that either $S^{\sigma^*}(D^{<\bar{f}>}(h)) \geq s_{\bar{f}}$ or $I^{\sigma^*}(D^{<\bar{f}>}(h)) \geq s_{\bar{f}}$.

For the online schedule σ, the jobs in $D^{<\bar{f}>}(h)$ are serviced by M during the time interval $[t_h, t_{h+1}]$. At time t_h, the job type $\bar{f}$ can be one of the following :
case 1 : $\bar{f}$ is isolated at time t_h
From the condition of case 1, we know that at time t_h, there exists an unserviced job J_l of type $\bar{f}$ with release time r_l such that there are no other jobs of type $\bar{f}$ that are released in the interval $[r_l, r_l + \tau_1]$.

From the definition of σ and J_l, we observe that $r_l \geq r_{\pi(h)}$. Otherwise, for the online schedule σ, at time t_h, M would have serviced job J_l instead of $J_{\pi(h)}$. From the definition of σ and $D(h)$, we know that the job $J_{\pi(h+1)} \in D^{<\bar{f}>}(h)$ and has a release time greater than the release time of job J_l. We also know from the definition of J_l that no jobs of type $\bar{f}$ are released during the time interval $[r_l, r_l + \tau_1]$. From the previous two observations we can conclude that $r_{\pi(h+1)} > r_l + \tau_1$. Therefore, $r_l \in [r_{\pi(h)}, r_{\pi(h+1)} - \tau_1]$.

For the optimal schedule σ^*, if M services a job of type $f \neq \bar{f}$ during the time interval between the completion of J_l and the release of $J_{\pi(h+1)}$, then we are done. So, we only need to consider the case when M only services jobs of type $\bar{f}$ during the time interval between the completion of J_l and the release of job $J_{\pi(h+1)}$. Now, we will analyze σ^*. From the definition of Λ, we know that for σ^*, all jobs in $D(h)$ that are released on or before the release of job J_l must be completed by time $r_l + \Lambda$. In addition, we know that J_l is of type $\bar{f}$ and there are no other jobs of type $\bar{f}$ that are released in the interval $[r_l, r_l + \tau_1]$. Therefore, for σ^*, if M only services jobs of type $\bar{f}$ during the time interval between the completion of J_l and the release of job $J_{\pi(h+1)}$, then M will idle during the time interval $[r_l + \Lambda, r_l + \tau_1]$. Since, $\tau_1 - \Lambda = \lambda >= s_{\bar{f}}$, we are done.

case 2 : $\bar{f}$ is not isolated at time t_h

From the definition of t_h and t_{h+1}, we observe that for σ, there exists a time in the interval (t_h, t_{h+1}), when M switches to a job of type $\hat{f} \neq \bar{f}$. Let $\hat{t}$ be the earliest time in (t_h, t_{h+1}) when M switches to a job of type $\hat{f}$. At time $\hat{t}$, $\hat{f}$ can be one of the following :

case 2a : $\hat{f}$ is isolated at time $\hat{t}$

From the condition of case 2a, we observe that for σ, at time $\hat{t}$, there exists an unserviced job of type $\hat{f}$ such that there are no other jobs of type $\hat{f}$ that are released within τ_1 units of the release of that job. At time $\hat{t}$, let J_l be the first unserviced job of type $\hat{f}$ such that there are no other jobs of type $\hat{f}$ that are released within the time interval $[r_l, r_l + \tau_1]$. From the definition of J_l and σ, we know that for σ, M will process $J_{\pi(h)}$ before J_l and J_l before $J_{\pi(h+1)}$. From the condition of case 2 and case 2a, we observe that $r_l \geq r_{\pi(h)} + \tau_2 - \tau_1$. Otherwise, at time t_h the job type $\hat{f}$ would have priority $priority^{\hat{f}}(t_h) = t_h - (r_l + \tau_1) > t_h - (r_{\pi(h)} + \tau_2) = priority^{\bar{f}}(t_h)$ the priority of job type $\bar{f}$ and M would have serviced J_l instead of $J_{\pi(h)}$. Similarly, from the definition of case 2 and case 2a we observe that $r_l \leq r_{\pi(h+1)} - \tau_1$. Otherwise, at time $\hat{t}$ the job type $\bar{f}$ would have priority $priority^{\bar{f}}(\hat{t}) = \hat{t} - r_{\pi(h+1)} > \hat{t} - (r_l + \tau_1) = priority^{\hat{f}}(\hat{t})$ the priority of job type $\hat{f}$ and M would have continued to service jobs of type $\bar{f}$ instead of switching to service jobs of type $\hat{f}$. Therefore, we conclude that the release time of job J_l is in the time interval $[r_{\pi(h)} + \tau_2 - \tau_1, r_{\pi(h+1)} - \tau_1]$

Now, we will make some observations about the optimal schedule σ^*. Since $r_l \in [r_{\pi(h)} + \tau_2 - \tau_1, r_{\pi(h+1)} - \tau_1]$, we observe that for σ^*, the starting time of job J_l is greater than the release time of job $J_{\pi(h)}$ by at least τ_1 time units and the starting time of job $J_{\pi(h+1)}$ is greater than the release time of job J_l by at least τ_1 time units. More formally,

$$B_l^{\sigma^*} \geq r_l \geq r_{\pi(h)} + \tau_2 - \tau_1 \geq r_{\pi(h)} + \tau_1 \ \ and \ \ B_{\pi(h+1)}^{\sigma^*} \geq r_{\pi(h+1)} \geq r_l + \tau_1$$

From the definition of Λ, we know that for σ^*, the completion time of job $J_{\pi(h)}$ is less than $r_{\pi(h)}+\tau_1$ and the completion time of job J_l is less than $r_l+\tau_1$. More formally,

$$C^{\bar{\sigma}^*}_{\pi(h)} \ \leq \ r_{\pi(h)}+\Lambda \ < \ r_{\pi(h)}+\tau_1 \ and \ C^{\bar{\sigma}^*}_{l} \ \leq \ r_l+\Lambda \ < \ r_l+\tau_1$$

From the above two observations, we get

$$C^{\sigma^*}_{\pi(h)} \leq B^{\sigma^*}_{l} \leq C^{\sigma^*}_{l} \leq B^{\sigma^*}_{\pi(h+1)} \tag{7}$$

From (8), we know that for σ^*, M processes a job of type $\hat{f} \neq \bar{f}$ during the time interval between the completion of job $J_{\pi(h)}$ and the start of processing of job $J_{\pi(h+1)}$. Therefore, M must have performed at least one set-up while servicing jobs in $D^{<\bar{f}>}(h)$. Hence $S^{\sigma^*}(D^{<\bar{f}>}(h)) \geq S^{\sigma^*}_{\pi(h+1)} \geq s_{\bar{f}}$.

case 2b : $\hat{f}$ is non-isolated at time $\hat{t}$

From the definition of $\hat{t}$, we know that for σ, at time $\hat{t}$ M switches to a job of type $\hat{f} \neq \bar{f}$. For σ, at time $\hat{t}$, let J_m be the first unserviced job of type $\hat{f}$ with release time r_m. From the definition of $\hat{t}$ and σ, we know that M will process $J_{\pi(h)}$ before all jobs of type $\hat{f}$ that were released in the interval $[r_m, r_m+\tau_2]$ and all jobs of type $\hat{f}$ that were released in the interval $[r_m, r_m+\tau_2]$ before $J_{\pi(h+1)}$. We can observe that $r_m > r_{\pi(h)}$, otherwise for σ, at time t_h, $\hat{f}$ would have higher priority than $\bar{f}$ and M would have serviced J_m before $J_{\pi(h)}$. We can also observe that $r_m < r_{\pi(h+1)} - \tau_1$, otherwise for σ, at time $\hat{t}$, $\bar{f}$ would have higher priority than $\hat{f}$ and M would have serviced $J_{\pi(h+1)}$ before J_m. From the condition of case $2b$, we can observe that at least one job of type $\hat{f}$ is released in the intervals $[r_m, r_m+\tau_1]$ and $[r_m+\tau_1, r_m+\tau_2]$. From the previous observations, we observe that there exists a job $J_l \in J$ of type $\hat{f}$ with release time in the interval $[r_{\pi(h)}+\tau_1, r_{\pi(h+1)}-\tau_1]$. From the definition of σ and J_l, we can observe that for σ M will process $J_{\pi(h)}$ before J_l and J_l before $J_{\pi(h+1)}$.

Now, we will make some observations about σ^*. Since $r_l \in [r_{\pi(h)}+\tau_1, r_{\pi(h+1)}-\tau_1]$, we observe that for σ^*, the starting time of job J_l is greater than the release time of job $J_{\pi(h)}$ by at least τ_1 time units and the starting time of job $J_{\pi(h+1)}$ is greater than the release time of job J_l by at least τ_1 time units. More formally,

$$B^{\sigma^*}_{l} \ \geq \ r_l \ \geq \ \ r_{\pi(h)}+\tau_1 \ \ and \ \ B^{\sigma^*}_{\pi(h+1)} \ \geq \ r_{\pi(h+1)} \ \geq \ r_l+\tau_1$$

From the definition of Λ, we know that for σ^*, the completion time of job $J_{\pi(h)}$ is less than $r_{\pi(h)}+\tau_1$ and the completion time of job J_l is less than $r_l+\tau_1$. More formally,

$$C^{\bar{\sigma}^*}_{\pi(h)} \ \leq \ r_{\pi(h)}+\Lambda \ < \ r_{\pi(h)}+\tau_1 \ and \ C^{\bar{\sigma}^*}_{l} \ \leq \ r_l+\Lambda \ < \ r_l+\tau_1$$

From the above two observations, we get

$$C^{\sigma^*}_{\pi(h)} \leq B^{\sigma^*}_{l} \leq C^{\sigma^*}_{l} \leq B^{\sigma^*}_{\pi(h+1)} \tag{8}$$

From (8), we know that for σ^*, M processes a job of type $\hat{f} \neq \bar{f}$ during the time interval between the completion of job $J_{\pi(h)}$ and the start of processing of job $J_{\pi(h+1)}$. Therefore, M must have performed at least one set-up while servicing jobs in $D^{<\bar{f}>}(h)$. Hence $S^{\sigma^*}(D^{<\bar{f}>}(h)) \geq S^{\sigma^*}_{\pi(h+1)} \geq s_{\bar{f}}$.

□

6 *NP*-Hardness proof for the maximum flow time problem

We reduce the knapsack problem to the maximum flow time problem. This reduction is similar to the reduction in Bruno and Downey [3].

Knapsack Problem : Let $u_1, u_2, ..., u_N$ be positive integers. Does there exist a subset $I \subseteq \{1, ..., N\}$ such that $\sum_{i \in I} u_i = b$.

We denote an instance of the knapsack problem by $KN(\{u_1, u_2, ..., u_N\}, b)$. Given an instance of the Knapsack problem, we will construct an instance MF of the maximum flow time problem as follows :

Let $u_{N+1} = \sum_{i \in \{1,...,N\}} u_i > b$ and $u_{N+2} = 0$. MF consists of a sequence consisting of $2N+3$ jobs belonging to $N+2$ job types $\{1, 2, ..., N+2\}$. For $1 \leq i \leq N+2$, the set-up time for job of type i is $s_i = u_i$. The job sequence consists of two batches of jobs, A and B. The batch A consists of $N+2$ jobs $A_1, A_2, ..., A_{N+2}$ and the batch B consists of $N+1$ jobs $B_1, B_2, ..., B_{N+1}$. The jobs in A are released at time 0 and the jobs in B are released at time $3u_{N+1} - 2b$. For $1 \leq i \leq N+1$, the job A_i is of type i and has a processing time $p_{i,1} = u_i$. The job A_{N+1} is of type $N+2$ and has a processing time $p_{N+2,1} = u_{N+1}$. For $1 \leq i \leq N+1$, the job B_i is of type i and has a processing time $p_{i,2} = u_i$.

Theorem 6: For the instance MF, there exists a schedule σ with maximum flow time $F^{\sigma}_{max} \leq 5u_{N+1} + b$ if and only if $KN(\{u_1, u_2, ..., u_N\}, b)$ has a solution.

Proof Let us suppose that the instance $KN(\{u_1, u_2, ..., u_N\}, b)$ does have a solution. Then there exists a subset $K \subseteq \{1, ..., N\}$ such that $\sum_{i \in K} u_i = b$. Let $A = \{A_{N+1}, A_{N+2}\} \cup A' \cup A''$, where $A' = \{A_i : i \in K\}$ and let $B = \{B_{N+1}\} \cup B' \cup B''$, where $B' = \{B_i : i \in K\}$. Now, for the instance MF, we will construct the schedule $\bar{\sigma}$ in phases, in the following manner :

Phase 1 : Starting at time 0, service the job A_{N+2};
Phase 2 : Starting at time u_{N+1}, service the jobs in A'' in the sequence in which they appear in A;
Phase 3 : Starting at time $3u_{N+1} - 2b$, service the jobs in $A' \cup B'$, such that, for $i \in K$, job B_i is serviced immediately after A_i;
Phase 4 : Starting at time $3u_{N+1} + b$, service the job A_{N+1} immediately followed by B_{N+1};
Phase 5 : Starting at time $6u_{N+1} + b$, service the jobs in B'' in the sequence in which they appear in B;

We can observe that $\bar{\sigma}$ is a valid schedule. From the construction of the instance MF, we know that all jobs in A are released at time 0 and all jobs in B are released at time $3u_{N+1} - 2b$. From the construction of the schedule $\bar{\sigma}$, we observe that for $\bar{\sigma}$, all jobs in A are completely processed by time $5u_{N+1} + b$ and all jobs in B are completely processed by time $8u_{N+1} - b$. Therefore, the maximum flow time of any job in A or B is at most $5u_{N+1} + b$. Thus, $F^{\bar{\sigma}}_{max} \leq 5u_{N+1} + b$.

Now, let us suppose that $KN(\{u_1, u_2, ..., u_N\}, b)$ does not have a solution. Let $\bar{\sigma}$ be any arbitrary schedule for MF. For $\bar{\sigma}$, let $\bar{B}$ be the subset of jobs in B that are serviced before some job in A is serviced and $P(\bar{B})$ be the total processing time of jobs in $\bar{B}$. Since $KN(\{u_1, ..., u_N\})$ does not have a solution, $P(\bar{B}) \neq b$. Suppose $P(\bar{B}) > b$, then M spends at least $b+1$ time units processing jobs in B before all jobs in A are serviced. So, the earliest time at which all jobs in A are serviced is $5u_{N+1}+b+1$. Therefore $F^{\bar{\sigma}}_{max} > 5u_{N+1}+b$. Now suppose $P(\bar{B}) < b$, then the servicing of job B_1 begins only after all jobs in A have been serviced. For $\bar{\sigma}$, let A_i be the last job in A to be serviced. If in schedule $\bar{\sigma}$, the job B_1 is serviced immediately after A_i then the total set-up time of jobs in B is at least $P(B-\bar{B}) \geq u_{N+1}-b+1$. Otherwise, the total set-up time of jobs in B is at least u_{N+1}. Therefore, for the schedule $\bar{\sigma}$, the earliest time by which all jobs in A and B will be serviced is $8u_{N+1}-b+1$. This implies $F^{\bar{\sigma}}_{max} > 5u_{N+1}+b$. $\square$

References

[1] Baker, K.R. and Magazine, M.J., 1999. Minimizing Maximum Lateness with Job Families, Personal communication with Professor Baker.

[2] Borodin, A. and El-Yaniv, R., 1998. Online Computation and Competitive Analysis, Cambridge University Press.

[3] Bruno, J. and P.Downey. 1978. Complexity of task sequencing with deadlines, setup times and changeover costs. SIAM Journal of Computing 7, 393-404.

[4] Charikar, M , Motwani, R, Raghavan, P, Silverstein, C. Constrained TSP and Low-Power Computing, 5th International Workshop, WADS' 97, LNCS 1272, 104-115

[5] Dunstall, S., Wirth, A., Baker,K.R., 1999, Lower bounds and algorithms for flow minimization on a single machine with setup times, personal communication with Professor Baker and Dr.Dunstall.

[6] Monma, C.L. and C.N.Potts. 1989. On the complexity of scheduling with batch set-up times. Operations Research. 37, 798-804.

[7] Garey, M.R. and Johnson, D.S. 1979. Computers and Intractability A Guide to the Theory of NP-Completeness. W.H.Freeman and Company, San Francisco.

[8] Hariri, A.M.A. and Potts, C.N. 1997. Single machine scheduling with batch set-up times to minimize maximum lateness, Annals of Operations Research 70, 75-92.

[9] Lenstra, J.K., 1977. Sequencing by enumerative methods. Mathematical centre Tracts Vol 69, North Holland Publications, ISBN 90 6196 125 4.

[10] Lenstra, J.K., A.H.G.Rinnooy Kan and P.Brucker. 1977. Complexity of machine scheduling problems. Annals of Discrete Math 1, 343-362.

[11] Schutten, J.M.J, 1996 Shop floor scheduling with set-up times. PhD Thesis, personal communication with Professor Schutten.

[12] Schutten, J.M.J, Van de Velde, S.L., Zijm, W.H.M. 1996. Single machine scheduling with release dates, due dates and family setup times. Management Science. 42(8), 1165-1174.

[13] Webster, S. and Baker,K.R. 1995. Scheduling group of jobs on a single machine. Operations Research 43, 692-703.

[14] Zdrzalka,S. 1991. Approximation algorithms for single machine sequencing with delivery times and unit batch set-up times, European Journal of Operational Research 51, 199-209.

[15] Zdrzalka,S. 1995. Analysis of approximation algorithms for single machine sequencing with delivery times and sequence independent batch set-up times, European Journal of Operational Research 51, 371-380.