DIMACS Technical Report 2001-50
December 2001

Online scheduling with release times and set-ups

by

Srikrishnan Divakaran [1]
Department of Computer Science
Hofstra University
West Hempstead, NY
cscsyd@hofstra.edu

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

[1] Supported in Part by NSF grant CCR-9700239 and DIMACS.
[2] Supported in Part by NSF grant CCR-9700239.

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs, NEC Research Institute and Telcordia Technologies (formerly Bellcore).

DIMACS was founded as an NSF Science and Technology Center, and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

In this paper, we address the problem of online scheduling of n independent jobs with release times and sequence independent set-ups on a single machine system with the objective of minimizing the maximum flow time. For this problem, we present an $O(1)$-competitive online algorithm. We also present the proof of NP-completeness of the offline problem.

1 Introduction

The *problem of scheduling with set-ups* [1,2,3,4,6,8,9,10,14,17,22,23,26,27,28,33,34] occurs naturally in systems that involve processing jobs of different types on common facilities where a set-up is incurred when there is a switch from processing a job of one type to a job of another type. The set-up time in general depends on both the type of the most recently serviced job as well as the type of the currently serviced job. This general case is referred to as *sequence dependent set-ups*. The special case where the set-up time depends only on the type of the currently serviced job is referred to as *sequence independent set-ups*.

Some specific examples of *scheduling with set-ups* were described in Monma and Potts [23], Baker [2,3] and Schutten et al[28,29]. The first example involves the production of different colors of paint on the same machine. A set-up time for cleaning the machine is incurred whenever the machine switches from manufacturing a paint of one color to another. The amount of time required to clean the machine depends both on the color being removed and the color for which the machine is being prepared. In this model, a set-up time is required at the start of production and on each occasion when the machine switches from manufacturing one color paint to another. In this scenario, a batch is a maximal set of jobs that are scheduled contiguously on a machine and share a set-up. Large batches have the advantage of high machine utilization because the number of set-ups is small. On the other hand, processing in large batches may cause some jobs to incur a large delay. So, in this manufacturing system the system designer needs to trade-off between the time spent on set-ups and the maximum delay of a job. Note that if all processing costs are zero this problem reduces to an instance of the well known travelling salesperson problem. A second example is the problem of scheduling tasks in computer systems. In this problem, a collection of tasks is to be processed on a computer where each task has a requirement for a particular compiler to be resident in the computer's memory. If the appropriate compiler is resident, then the task may start processing immediately; otherwise a setup time is incurred to bring the relevant compiler into memory. In this example, the setup time depends only on the time to load the compiler for the current job, and does not depend on the previous job. A third example is a manufacturing system where the labor force is a limiting resource and workers must be switched from machine to machine, incurring a set-up time, in order to complete all tasks.

For scheduling problems with sequence independent set-ups, the *maximum flow time*, *total weighted completion time* and *maximum lateness* are some of the natural optimality criteria for evaluating solutions. When considering maximum flow time, the objective is to minimize the maximum amount of time any job spends in the system. When considering the *total weighted completion time* criterion, each job is associated with a penalty or non-negative weight for waiting, and the objective is to minimize the total penalty for waiting. For the maximum lateness criterion, each job has a due date and the objective is to minimize the maximum amount of time by which the completion time of any job deviates from its due

date.

In this paper, we focus on the problem of scheduling a sequence of n jobs with release times and sequence independent set-ups on a single machine to minimize the maximum flow time (i.e. maximum amount of time any job waits in the system) and we refer to this problem as the *maximum flow time problem*.

Related Work: For the maximum flow time problem, when the release times of jobs are the same, then any schedule where the jobs within each type are processed in a single batch is optimal. We can also observe that when the set-up times are zero, an optimal solution can be obtained in $O(nlogn)$ time by scheduling jobs in non-decreasing value of release times. So, we consider the maximum flow time problem when the release times and set-up times are non-trivial. For this problem, Monma and Potts[23] provide an offline dynamic programming algorithm that obtains an optimal solution in $O(n^F)$ time, where n is the number of jobs and F is the number of job types. This algorithm is polynomial time if F is fixed, but exponential otherwise. However, when the number of job types is arbitrary, we show this problem to be NP-hard even when there are at most three distinct release times and at most two jobs within each job type.

From the perspective of heuristics, Schutten et al[27,28] present a branch and bound algorithm for the *maximum lateness problem* [2,3,9,10,17,23, 26,27,28,33,34], a generalization of the maximum flow time problem. Though their branch and bound algorithm performs well for small problem instances, it does not provide any approximation guarantees. In addition, their algorithm can be used only in solving the offline version of the maximum flow time problem.

From the perspective of approximation algorithms, Zdrzalka[33,34] presented a constant approximation algorithm to the *maximum lateness problem* when all jobs have a negative due dates. However, when the release times are non-trivial, this algorithm cannot be used in solving our problem of minimizing the maximum flow time.

Notice that the maximum flow time problem is a special case of the maximum lateness problem when the due dates of jobs equals their release times. For details on related problems, we refer the reader to the papers of Baker [2,3], Bruno and Downey[6], Divakaran and Saks[9], Monma and Potts[23] and Potts and Kovalov[26], Schutten et al[27,28] and Zdrzalka[33,34].

Our Results : In this paper, we present an $O(1)$-competitive online algorithm for the problem of scheduling a sequence of n jobs with release times and sequence independent set-ups on a single machine to minimize the maximum flow time. We also present the proof of NP-completeness for the offline version of this problem.

2 Problem Description

Model Description : We consider a system consisting of a single machine M that is capable of sequentially serving one job at a time. Each job belongs to one of F pre-determined job types $1, 2, ..., F$. The machine services a job in two stages : the set-up stage and the processing stage. In the set-up stage, the machine checks whether the most recently serviced job is of the same type as the job that is currently serviced. If so, the machine proceeds immediately to the processing stage. Otherwise, the machine is configured (or set-up) to process jobs of that type. The set-up for a job is needed only if at the start of servicing, the most recently serviced job is not of the same type as the current job. In the processing stage, the machine processes the job without interruption. The *set-up time* for a given job depends only on the type of the currently serviced job and is referred to as *sequence independent set-ups*.

The incoming job sequence is typically denoted by $J = (J_1, J_2, ..., J_N)$. Each job $J_i \in J$ is of type $\tau_i \in [1..F]$, is available for processing at time r_i and requires a processing time p_i, and prior to processing J_i the machine requires a minimum setup time of 0 if the most recently processed job is of type τ_i and s_{τ_i} otherwise.

Preliminaries : A *schedule* σ for a job sequence J specifies for each job $J_i \in J$, the time A_i^σ at which the set-up of J_i starts, the time B_i^σ at which processing of J_i starts and the time C_i^σ at which J_i completes. A schedule σ for job sequence J is said to be *feasible* if

- for any pair of jobs $J_i, J_j \in J$, the time intervals $[A_i^\sigma, C_i^\sigma]$ and $[A_j^\sigma, C_j^\sigma]$ are disjoint (i.e. the machine M can serve only one job at a time).
- for $i \in [1..n]$, $A_i^\sigma \geq r_i$ (i.e. a job cannot be assigned before it is released);
- for $i \in [1..n]$, $B_i^\sigma - A_i^\sigma$ is greater than or equal to the set-up time for J_i. Note that if the type of the most recently serviced job is of the same type as this job, then its set-up time is considered to be zero.
- for $i \in [1..n]$, $C_i^\sigma - B_i^\sigma \geq p_i$ (i.e. the time interval between the completion and the start of processing of a job is greater than or equal to its processing time).

We denote the set of feasible schedules for J by $\Phi(J)$. The objective function (or optimality criteria) that we use in evaluating a feasible schedule in $\Phi(J)$ is *Maximum Flow Time* $F_{max}^\sigma = \max_{i \in [1..n]} F_i^\sigma = C_i^\sigma - r_i$. This objective function is a function of set of job completion times, so their general form is $Z = f(C_1, C_2, ..., C_n)$. Furthermore, these objective functions are *regular*, that is Z can increase only if the completion time of at least one of the jobs increase.

For a given objective function f, a schedule $\sigma \in \Phi(J)$ is said to be optimal if $f(C_1^\sigma, ..., C_n^\sigma) \leq f(C_1^\phi, ..., C_n^\phi)$ for any other schedule $\phi \in \Phi(J)$. In the future, for ease of notation, we will

often denote $f(C_1^\sigma, ..., C_n^\sigma)$ by f^σ.

A scheduling problem $\mathcal{P}$ consists of a set $\mathcal{J}$ of possible input sequences and a regular objective function f. Associated with every input sequence J is a set of feasible schedules $\Phi(J)$, and associated with each feasible schedule $\sigma \in \Phi(J)$ is a real number f^σ representing the value of objective function for the schedule σ.

Given any input sequence J, a scheduling algorithm G for a scheduling problem $\mathcal{P}$ assigns to each job sequence J a schedule $G(J) \in \Phi(J)$. The value of the objective function associated with the schedule $G(J)$ is denoted by $f^{G(J)}$. An *optimal algorithm* OPT is such that for all sequences J, $f^{OPT(J)} = \min_{\sigma \in \Phi(J)} f^\sigma$. It is easy to show that for a regular objective function every sequence has an optimal schedule.

If each job in the sequence of jobs is revealed to G at the time it is released and at all times the algorithm 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 an online scheduling algorithm to J and $\hat{J}$ will be identical up to time t. When we want to emphasize that an algorithm is not necessarily online we say it is offline.

An on-line algorithm that is a c-approximation algorithm is said to be *c-competitive*. If an algorithm G is c-competitive [5] we sometimes say that G has a competitive ratio of c. Notice that the competitive ratio is at least 1. An algorithm is called *competitive* if it attains a "constant" competitive ratio c. Although c may be a function of the problem parameters, it must be independent of the size of the input sequence J. For example, in a scheduling problem involving m machines, we might have an algorithm with a competitive ratio c that depends on m, but must be independent of the number and type of jobs in the input sequence. The infimum over the set of all values c such that G is c-competitive is called the *competitive ratio* of G and is denoted by $R(G)$.

Problem Definition : In this paper, we present an $O(1)$-competitive online algorithm for the *maximum flow time problem* which is defined as follows :

> **Maximum Flow Time Problem** $(1|s_f, r_i|F_{max})$ **:** Given a single machine system with sequence independent set-ups and a job sequence $J = (J_1 J_2, ..., J_n)$, where each job J_i belongs to one of F job types $\{1, ..., F\}$, is available for processing at time r_i and requires a processing time p_i, determine a schedule $\sigma \in \Phi(J)$ that minimizes the maximum flow time of jobs in J.

3 Paper Outline

The technical results in this paper are organized as follows : In Section 4, we present the NP-completeness proof of the maximum flow time problem, in section 5, we present three natural but non-competitive algorithms for the maximum flow time problem, in section 6, we present an $O(1)$-competitive online algorithm for the maximum flow time problem. We present the analysis of our algorithm in section 7.

4 Proof of NP-Hardness 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 [6].

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 1: 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 $F^{\bar{\sigma}}_{max} > 5u_{N+1} + b$. □

5 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. The algorithms we consider have the following 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 $f \in \{1, ..., F\}$), then it is easy to observe that **Earliest-First** does not have a finite competitive ratio.

We can 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). It is also easy to observe that for the maximum flow time problem Avoid-Setup does not have finite competitive ratio. We observe that putting highest priority on avoiding set-up can lead to increasing the flow time of some jobs (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 shows 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.

6 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.

6.1 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

$$\begin{aligned} isolated^{<f>}(t) &= \{\ J_i \in Q^{<f>}(t), \text{ such that there is no job of type } f \text{ that was} \\ &\quad \text{released within } \tau_1 \text{ time units after the release of } J_i \text{ and } r_i + \tau_1 < t\}; \\ firstgap^{<f>}(t) &= \text{the release time of the job in } isolated^{<f>}(t) \text{ with the earliest} \\ &\quad \text{release time. If } isolated^{<f>}(t) = \emptyset \text{ then } firstgap^{<f>}(t) = \infty; \end{aligned}$$

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

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

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. Later, during the analysis , we show that by choosing $\tau_1 < \tau_2$ our

algorithm penalizes the non-isolated job types more than the isolated job types and this helps in obtaining constant competitiveness.

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 2: Let $\lambda = \max_{f\in\{1,\ldots,F\}} s_f$, $\tau_1 = (\Lambda + \lambda)$ and $\tau_2 = 3(\Lambda + \lambda)$, then for any $J \in J_\Lambda$, $F_{max}^{G_{(\tau_1,\tau_2)}(J)} \leq \tau_1 + 2\tau_2 \leq 7\Lambda + 7\lambda$

The proof of Theorem 2 is given in Section 6.3.

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

Using Theorem 2, 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 help define algorithm H.

DEFINITION 6.1: 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 $J^{\bar{(t_1)}} = J$;
Let $\Lambda_1 = \lambda$, $\tau_1^1 = \Lambda_1 + \lambda$ and $\tau_2^1 = 3(\Lambda_1 + \lambda)$;
while $(J^{\bar{(t_i)}} \neq \emptyset)$
{
  Execute algorithm $G_{(\tau_1^i,\tau_2^i)}$ for sequence $J^{\bar{(t_i)}}$;
  Let $i = i + 1$;
  Let t_i be the most recent time at which some job in J completed;
  Let $J^{\bar{(t_i)}}$ be the adjusted residual job sequence at time t_i;

Let $\Lambda_i = 2^{i-1}\lambda$, $\tau_1^i = \Lambda_i + \lambda$ and $\tau_2^i = 3(\Lambda_i + \lambda)$;
}

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 3: The algorithm H is 28 competitive.

Proof . Let h be the stage during which H completes servicing all jobs in J. If $h = 1$, then clearly $F_{max}^{H(J)} \leq 7(\Lambda_1+\lambda) \leq 7F_{max}^{OPT(J)}+7\lambda$. 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^* denote the optimal schedule $OPT(K)$ and $\bar{\sigma}^*$ denote the optimal schedule $OPT(\bar{K})$. Notice that $\bar{\sigma}^*$ is a shifted version of σ^*. 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 2, 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 $\bar{J}^{(i)} \leq \tau_1^i + 2\tau_2^i \leq 7(\Lambda_i + \lambda)$. 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 $\tau_1^i + 2\tau_2^i \leq 7(\Lambda_i + \lambda)$. Therefore, $F_{max}^{H(J)} \leq 7(\Lambda_h + \lambda) + 7\sum_{j\in[1,h-1]}(\Lambda_j + \lambda) \leq 7(\Lambda_{h+1} + \lambda)$. This implies that $F_{max}^{H(J)} \leq 7(\Lambda_{h+1} + \lambda) \leq 28F_{max}^{OPT(J)} + 7\lambda$. □ .

6.2 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 = $\max_{i\in\{1,...,N\}}\{C_i^\sigma - r_i\}$ 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;
$P^\sigma(J)$ = $\sum_{i\in\{1,...,N\}} p_i$ 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 $J = (J_1, ..., J_N) \in J_\Lambda$ is fixed.

6.3 Proof of Theorem 2

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 were completely processed during the time interval $[\hat{t} - \tau_2, t - (\tau_1 + \tau_2) + \Lambda]$ in the optimal schedule σ^*. 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 4: 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 4, which proves the theorem. □

6.4 Proof of Lemma 4

Proof . We partition 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 $f \in \{1, ..., F\}$,

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

Now, we will first introduce certain definitions and lemmas that will be used in proving the claim.

DEFINITIONS **6.2:** Let $\bar{f} \in [1..f]$ be any arbitrary job type and $\alpha = \alpha_{\bar{f}}$ be the number of set-ups performed in σ while serving jobs in $K^{<\bar{f}>}$.

For the online schedule σ, let $t_1 < t_2 < ... < t_\alpha$ be the respective times at which M performs set-up while serving jobs in $K^{<\bar{f}>}$ (i.e. jobs of type $\bar{f}$ in K), and $\pi(h)$, $1 \leq h \leq \alpha$, be the index 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, we will introduce two lemmas that will help in establishing a lower bound for the total set-up time in the optimal schedule σ^* while processing jobs of type $\bar{f}$ in K.

Lemma 5: If $(\bar{f} \neq f^*)$ then $S^{\sigma^*}(D^{<\bar{f}>}(0)) \geq s_{\bar{f}}$

Proof Without loss of generality we assume that in the optimal schedule σ^* all jobs of the same type are processed in non-decreasing order of their release times. From the 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 $S^{\sigma^*}(D^{<\bar{f}>}(0)) \geq s_{\bar{f}}$. □

Lemma 6: For any $h \in \{1, ..., \alpha - 1\}$, $S^{\sigma^*}(D^{<\bar{f}>}(h)) + I^{\sigma^*}(D^{<\bar{f}>}(h)) \geq s_{\bar{f}}$

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

We prove this lemma in two steps. In the first step, we show that either one of the following conditions holds.

(Condition 1) there exists a job $J_l \in D^{<\bar{f}>}(h)$ such that $r_l \in [r_{\pi(h)}, r_{\pi(h+1)} - \tau_1]$ and there are no other jobs of type $\bar{f}$ with release time in $(r_l, r_l + \tau_1)$.

(Condition 2) there exists a job in $D(h)$ of type $f \neq \bar{f}$ with release time in $[r_{\pi(h)} + \tau_1, r_{\pi(h+1)} - \tau_1]$.

Then, in step 2, we show that if Condition 1 is true then either $S^{\sigma^*}(D^{<\bar{f}>}(h)) \geq s_{\bar{f}}$ or $I^{\sigma^*}(D^{<\bar{f}>}(h)) \geq s_{\bar{f}}$, otherwise we show that $S^{\sigma^*}(D^{<\bar{f}>}(h)) \geq s_{\bar{f}}$.

Now, we start the first step by analyzing the online schedule σ during the time interval (t_h, t_{h+1}) and showing that either Condition 1 or Condition 2 is true.

From definition of $\pi(h)$ and $\pi(h+1)$, we observe that $J_{\pi(h)}$ and $J_{\pi(h+1)}$ are of type $\bar{f}$ and in σ, M starts serving them at times t_h and t_{h+1} respectively. From the definition of algorithm $G_{(\tau_1,\tau_2)}$, we know that in σ , $J_{\pi(h)}$ is processed before any job in $D^{<\bar{f}>}(h)$ and jobs in $D^{<\bar{f}>}(h)$ are processed in the sequence in which they appear in $D^{<\bar{f}>}(h)$ (i.e. non-decreasing value of job release times). Therefore, from the above observation and the definition of t_h and t_{h+1}, we can observe that in σ, once M starts processing $J_{\pi(h)}$ it continues to process all jobs of type $\bar{f}$ in $D^{<\bar{f}>}(h)$ except $J_{\pi(h+1)}$ before switching to a job of type $f \neq \bar{f} \in [1..F]$ at some time $t \in (t_h, t_{h+1})$. Let $\hat{t}$ be the earliest time in (t_h, t_{h+1}) when M switches to a job of type $\hat{f} \neq f \in [1..F]$. Now, in σ the following situations are possible :

case 1 : $\bar{f}$ is isolated at time t_h
From definition of Case 1, we know that at time t_h, there exists an unserviced job J_l of type $\bar{f}$ such that $r_l = firstgap^{<\bar{f}>}(t_h) \geq r_{\pi(h)}$. From the definition of case 1 and algorithm $G_{(\tau_1,\tau_2)}$, we observe that at time t_h^- (immediately prior to the start of servicing of $J_{\pi(h)}$),

$$priority^{<\bar{f}>}(t_h^-) = t_h - (r_l + \tau_1) > priority^{<f>}(t_h^-) \quad \forall f \neq \bar{f} \tag{6}$$

We can also observe that at time t_h, $\bar{f}$ becomes active and as a result of this $\bar{f}$'s priority increases by at least τ_1, where as the priority of all other job types remain the same. More formally, at time t_h

$$priority^{<\bar{f}>}(t_h) = t_h - r_{\pi(h)} \geq t_h - r_l$$
$$priority^{<f>}(t_h) \leq t_h - (r_l + \tau_1) \tag{7}$$

Now, in order to show that Condition 1 is true, we will first establish that in σ, once M starts processing $J_{\pi(h)}$, it will switch to $\hat{f}$ only after processing J_l. Suppose, on the contrary, M switches to $\hat{f}$ prior to processing J_l. From Equations (7), we can observe that at time t_h, since the priority of $\bar{f}$ is greater than the priority of all other job types, $firstgap^{<\hat{f}>}(t_h) > r_l$ and $oldest^{<\hat{f}>}(t_h) > r_l + \tau_1 - \tau_2$. In addition, we also know that in σ, M continues to service only jobs of type $\bar{f}$ in the time interval $(t_h, \hat{t})$. From this we get, $firstgap^{<\hat{f}>}(\hat{t}) > r_l$ and $oldest^{<\hat{f}>}(\hat{t}) > r_l + \tau_1 - \tau_2$. This implies that at time $\hat{t}$,

$$priority^{<\hat{f}>}(\hat{t}) < \hat{t} - (r_l + \tau_1) \leq priority^{<\bar{f}>}(\hat{t}) \tag{8}$$

Equation (8) contradicts Equation (7). Therefore, in σ, once M starts processing $J_{\pi(h)}$, it will switch to $\hat{f}$ only after processing J_l.

Now, we will show that $r_l \in [r_{\pi(h)}, r_{\pi(h+1)} - \tau_1]$. From definition of J_l and $\pi(h)$ and $\pi(h+1)$, we know that $J_{\pi(h)}, J_l$ and $J_{\pi(h+1)}$ are of type $\bar{f}$, and $r_{\pi(h)} \leq r_l \leq r_{\pi(h+1)}$. From the definition of case 1, we also know that there exists no job of type $\bar{f}$ in $(r_l, r_l + \tau_1)$. Therefore, $r_{\pi(h+1)} \geq r_l + \tau_1$. Hence, $r_l \in [r_{\pi(h)}, r_{\pi(h+1)} - \tau_1]$ and Condition 1 is true.

case 2 : $\bar{f}$ is non-isolated at time t_h and $\hat{f}$ is isolated at time $\hat{t}$
From definition of Case 2, we know that at time $\hat{t}$, there exists an unserviced job J_m of type $\hat{f}$ such that $r_m = firstgap^{<\hat{f}>}(t_h) \geq r_{\pi(h)}$. From the definition of Case 2 and algorithm $G_{(\tau_1,\tau_2)}$, we observe that at time t_h^- (immediately prior to the start of servicing of $J_{\pi(h)}$),

$$priority^{<\bar{f}>}(t_h^-) = t_h - (r_{\pi(h)} + \tau_2) > priority^{<f>}(t_h^-) \quad \forall f \neq \bar{f} \tag{9}$$

From Equation (9) and the definition of case 2, we can observe that $r_m \geq r_{\pi(h)} + \tau_2 - \tau_1$. Suppose, on the contrary $r_m < r_{\pi(h)} + \tau_2 - \tau_1$, then from the definition of Case 2 and the above assumption, we can observe that $\hat{f}$ would have been isolated at time t_h. This implies that at time t_h^- (i.e. immediately prior to start of servicing of $J_{\pi(h)}$),

$$priority^{<\hat{f}>}(t_h^-) = t_h - (r_m + \tau_1) > t_h - (r_{\pi(h)} + \tau_2 - \tau_1 + \tau_1) = t_h - (r_{\pi(h)} + \tau_2) = priority^{<\bar{f}>}(t_h^-) \tag{10}$$

But, Equation (10) contradicts Equation (9). Therefore,

$$r_m \geq r_{\pi(h)} + \tau_2 - \tau_1 > r_{\pi(h)} + \tau_1 \tag{11}$$

From definition of case 2, we can observe that at time $\hat{t}^-$ (i.e. immediately prior to switching to $\hat{f}$), $\bar{f}$ was active and

$$priority^{<\hat{f}>}(\hat{t}^-) = \hat{t} - (r_{m)} + \tau_1) > priority^{<\bar{f}>}(\hat{t}^-) \geq \hat{t} - r_{\pi(h+1)} \tag{12}$$

From Equation (12), we get

$$r_m \leq r_{\pi(h+1)} - \tau_1 \tag{13}$$

Therefore, from Equations (11) and (13), we get $r_m \in [r_{\pi(h)} + \tau_1, r_{\pi(h+1)} - \tau_1]$. This implies Condition 2 is true.

case 3 **:** $\bar{f}$ is non-isolated at time t_h and $\hat{f}$ is non-isolated at time $\hat{t}$
From definition of Case 3, we know that at time $\hat{t}$ there exists an unserviced job J_m of type $\hat{f}$ such that $r_m = firstgap^{<\hat{f}>}(t_h) \geq r_{\pi(h)}$. From the definition of Case 3 and algorithm $G_{(\tau_1,\tau_2)}$, we observe that at time t_h^- (i.e. immediately prior to the start of servicing of $J_{\pi(h)}$),

$$priority^{<\bar{f}>}(t_h^-) = t_h - (r_{\pi(h)} + \tau_2) > priority^{<f>}(t_h^-) \quad for \quad allf \neq \bar{f} \tag{14}$$

From Equation (14), and definition of case 3, we can observe that $r_m \geq r_{\pi(h)}$. From definition of case 3, we can observe that at time $\hat{t}^-$ (i.e. immediately prior to switching to $\hat{f}$), $\bar{f}$ was active and

$$priority^{<\hat{f}>}(\hat{t}^-) = \hat{t} - (r_{m)} + \tau_2) > priority^{<\bar{f}>}(\hat{t}^-) \geq \hat{t} - r_{\pi(h+1)} \tag{15}$$

From Equation (15), we get $r_m \leq r_{\pi(h+1)} - \tau_2$.

From the definition of case 3 and non-isolated job types, we know that there exists at least one job of type $\hat{f}$ in $[r_m + \tau_1, r_m + 2\tau_1]$, otherwise $\hat{f}$ would have been isolated at time $\hat{t}$. In addition, $r_m + \tau_1 > r_{\pi(h)} + \tau_1$ and $r_m + 2\tau_1 = r_m + \tau_2 - \tau_1 \leq r_{\pi(h+1)} - \tau_1$. Therefore, there exists a job of type $\hat{f}$ with release time in $[r_{\pi(h)} + \tau_1, r_{\pi(h+1)} - \tau_1]$.

Now, we will show that if Condition (1) is true, then in σ^* either $S^{\sigma^*}(D^{<\bar{f}>}(h)) \geq s_{\bar{f}}$ or $I^{\sigma^*}(D^{<\bar{f}>}(h)) \geq s_{\bar{f}}$. If Condition 1 is true, then from definition we know that there exists a job $J_l \in D^{<\bar{f}>}(h)$ such that $r_l \in [r_{\pi(h)}, r_{\pi(h+1)} - \tau_1]$ and there are no other jobs of type $\bar{f}$ in J with release time in $(r_l, r_l + \tau_1]$. In 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 $J_{\pi(h+1)}$. 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, in 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.

Now, we will show that if condition 2 is true then in σ^*, $S^{\sigma^*}(D^{<\bar{f}>}(h)) \geq s_{\bar{f}}$.

Now, if condition (2) is true then we know that there exists a job J_n of type $\hat{f} \neq \bar{f}$ such that $r_n \in [r_{\pi(h)} + \tau_1, r_{\pi(h+1)} - \tau_1]$. Now, we will show that in σ^*, $J_{\pi(h)}$ is processed before J_n and J_n is processed before $J_{\pi(h+1)}$. Since $r_n \in [r_{\pi(h)} + \tau_1, r_{\pi(h+1)} - \tau_1]$, we observe that in σ^*, the starting time of job J_n 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_n by at least τ_1 time units. More formally,

$$B_n^{\sigma^*} \;\geq\; r_n \;\geq\; r_{\pi(h)} + \tau_1 \;\; and \;\; B_{\pi(h+1)}^{\sigma^*} \;\geq\; r_{\pi(h+1)} \;\geq\; r_n + \tau_1$$

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

$$C_{\pi(h)}^{\sigma^*} \;\leq\; r_{\pi(h)} + \Lambda \;<\; r_{\pi(h)} + \tau_1 \; and \; C_n^{\sigma^*} \;\leq\; r_n + \Lambda \;<\; r_n + \tau_1$$

From the above two observations, we get

$$C_{\pi(h)}^{\sigma^*} \leq B_n^{\sigma^*} \leq C_n^{\sigma^*} \leq B_{\pi(h+1)}^{\sigma^*} \tag{16}$$

From (16) , we know that in 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 $S^{\sigma^*}(D^{<\bar{f}>}(h)) \geq S_{\pi(h+1)}^{\sigma^*} \geq s_{\bar{f}}$. □

Now, we are ready to prove the Claim.

6.5 Proof of Claim

Proof 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,

$$\begin{aligned} S^{\sigma^*}(K^{*<\bar{f}>}) + I^{\sigma^*}(K^{*<\bar{f}>}) \;&=\; \sum_{h\in[0,\alpha]} S^{\sigma^*}(D^{<\bar{f}>}(h)) + \sum_{h\in[0,\alpha]} I^{\sigma^*}(D^{<\bar{f}>}(h)) \\ &\geq S^{\sigma^*}(D^{<\bar{f}>}(0)) + \sum_{h\in[1,\alpha-1]} (\; S^{\sigma^*}(D^{<\bar{f}>}(h)) + I^{\sigma^*}(D^{<\bar{f}>}(h)) \;) \end{aligned} \tag{17}$$

Now, we bound $S^{\sigma^*}(D^{<\bar{f}>}(0))$ by using Lemma 5 For $h \in \{1, ..., \alpha - 1\}$, we bound $S^{\sigma^*}(D^{<\bar{f}>}(h)) + I^{\sigma^*}(D^{<\bar{f}>}(h))$ by using Lemma 6. Now, evaluating (17) using Lemma 5 and Lemma 6, the claim follows immediately. □

7 Acknowledgments

The first author would like to thank Professors Eric Allender, Michael Grigoriadis, Howard Karloff, Liviu Iftode and Bill Steiger for their time and help in reviewing this paper, Professors Ken Baker, J.M.J.Schutten and Dr. Dunstall for their time and generous help in providing me copies of their papers and relevant research articles in scheduling with set-ups. We would also like to thank the National Science Foundation and DIMACS for supporting part of this research through their grants CCR-9700239, CCR-9988526 and CCR91-19999.

References

[1] Ahn, B.H., Hyun, J.H., 1990. Single facility multi class job scheduling, Computers and Operations Research, 265-272(17)

[2] Baker., K.R. and Webster, S., 1995. Scheduling groups of jobs on a single machine, Operations Research, 692-703 (43-4).

[3] Baker, K.R. and Magazine, M.J., 2000. Minimizing Maximum Lateness with Job Families, European Journal of Operational Research, 126-139(127).

[4] Baker, K.R. and Magazine, M.J., 2000. Minimizing Maximum Lateness with Job Families, European Journal of Operational Research, 126-139(127).

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

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

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

[8] Crauwels, H.A.J., Potts, C.N., Van Wassenhove, L.N. 1997. Local search heuristics for single machine scheduling with batch set-up times to minimize the total weighted completion time, Annals of Operations Research, 261-279(70)

[9] Divakaran, S., Saks, M., 2001, Approximation algorithms for Offline scheduling with set-ups, Dimacs Technical report, 2001.

[10] 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.

[11] Fiat, A., Karp, R.M., Luby, M., McGeoch, L.A., Sleator, D.D., Young, N.E., 1988. Competitive paging algorithms, Journal of Algorithms, 12 : 685-699.

[12] 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.

[13] Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling, Annals of Discrete Math, 5:287-326.

[14] Gupta, J.N.D., 1988. Single facility scheduling with multiple job classes, European Journal of Operational Research, 42-45 (33)

[15] Hall, L.A., Shmoys, D.B., Wein, J. 1996. Scheduling to minimize average completion time : off-line and on-line algorithms. Proceedings of the sixth ACM-SIAM Symposium on Discrete Algorithms.

[16] Hall, L.A. 1999. Approximation algorithms for scheduling, survey in Approximation algorithms for NP-hard problems, edited by Dorit S. Hochbaum.

[17] 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.

[18] Jackson, J.R., 1955., Scheduling a production line to minimize maximum tardiness, Research report, UCLA.

[19] Kalyanasundaram, B., Pruhs., K, 1995., Speed is as powerful as clairvoyance, Proceedings of 36th Annual Symposium on Foundations of Computer Science, 214-225.

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

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

[22] Mason, A.J., Anderson, E.J., 1991. Minimizing flow time on a single machine with job classes and set-up times. Naval research logistics 335-350(38).

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

[24] Phillips, C.A., Stein, C., Torng, E., Wein, J., 1998. Optimal Time-Critical Scheduling via resource augmentation, personal communications.

[25] Phillips, C., Stein, C., Wein., 1995. Scheduling jobs that arrive over time, Proceedings of 4th WADS, Lecture notes in computer science, 955, Springer, Berlin, 86-97..

[26] Potts, C.N., Kovalyov, M.Y., 2000. Scheduling with batching : a review, European journal of operational research, 228-249 (120).

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

[28] 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

[29] Sleator, D.D. and Tarjan, R.E., 1985. Amortized efficiency of list update and paging rules, Communications of the ACM, 28(2):202-208.

[30] Smith, W., 1956. Various optimizers for single stage production, Naval Research Logistics Quarterly, 59-66 (3)

[31] Torng, E., 1998. A unified analysis of paging and caching, Algorithmica, 20: 175-200

[32] Young, N.E., 1991. Competitive paging and dual guided online weighted caching and matching algorithms, Ph.D. dissertation. Department of Computer Science, Princeton University.

[33] 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.

[34] 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.