Scheduling Parallelizable Jobs Online to Maximize Throughput

Kunal Agrawal[1], Jing Li[2], Kefu Lu[1], and Benjamin Moseley[1]

[1] Washington University in St. Louis, St. Louis, MO 63130, USA,
kefulu@wustl.edu
[2] New Jersey Institute of Technology, Newark, NJ 07102, USA

Abstract. In this paper, we consider scheduling parallelizable jobs online to maximize the throughput or profit of the schedule. In particular, a set of n jobs arrive online and each job J_i arriving at time r_i has an associated function $p_i(t)$ which is the profit obtained for finishing job J_i at time $t + r_i$. Each job can have its own arbitrary non-increasing profit function. We consider the case where each job is a parallel job that can be represented as a directed acyclic graph (DAG). We give the first non-trivial results for the profit scheduling problem for DAG jobs and show $O(1)$-competitive algorithms using resource augmentation.

1 Introduction

Scheduling preemptive jobs online to meet deadlines is a fundamental problem and, consequently, the area has been extensively studied. In a typical setting, there are n jobs that arrive over time. Each job J_i arrives at time r_i, has a deadline d_i, relative deadline $D_i = d_i - r_i$ and a profit or weight p_i that is obtained if the job is completed by its deadline. The *throughput* of a schedule is the total profit of the jobs completed by their deadlines and a popular scheduling objective is to maximize the total throughput of the schedule.

In a generalization of the throughput problem, each job J_i is associated with a function $p_i(t)$ which specifies the profit obtained for finishing job J_i at $r_i + t$. It is assumed that p_i can be different for each job J_i and that the functions are arbitrary non-increasing functions. We call this problem the *general profit* scheduling problem.

In this work, we consider the throughput and general profit scheduling problems in the preemptive online setting for parallel jobs. In this setting, the *online* scheduler is only aware of the job at the time it arrives in the system, and a job is *preemptive* if it can be started, stopped, and resumed from the previous position later. We model parallel jobs as a directed acyclic graph (DAG) where each job J_i is represented as an *independent* DAG. Each node in the DAG is a sequence of instructions that are to be executed and the edges in DAG represent dependencies. A node can be executed if and only if all of its predecessors have been completed. Therefore, two nodes can potentially be executed in parallel if neither precedes the other in the DAG. In this setting, each job J_i arrives as a single independent DAG and a profit of p_i is obtained if *all* nodes of the DAG are completed by job J_i's deadline. The DAG model can represent parallel programs written in many widely used parallel languages and libraries, such as OpenMP [1], Cilk Plus [2], Intel TBB [3] and Microsoft Parallel Programming Library [4].

Both the throughput and general profit scheduling problem have been studied extensively for sequential jobs. In the simplest setting, each job J_i has work or processing time W_i to be processed on a single machine (processor). It is known that there exists a deterministic algorithm which is $O(\delta)$-competitive, where δ is the ratio of the maximum to minimum density of a job [5,6,7,8]. The *density* of job J_i is $\frac{p_i}{W_i}$ (the ratio of its profit to its work). In addition, this is the best possible result for any deterministic online algorithm even in the case where all jobs have unit profit and the goal is to complete as many jobs as possible by their deadline. In the case where the algorithm can be randomized, $\Theta(\min\{\log \delta, \log \Delta\})$ is the optimal competitive ratio [9,10]. Here Δ is the ratio of the maximum to minimum job processing time.

These strong lower bounds on the competitive ratio on any online algorithm makes it difficult to differentiate between algorithms and to discover the key algorithmic ideas that work well in practice. To overcome this challenge, the now standard form of analysis in scheduling theory is a *resource augmentation* analysis [11,12]. In a resource augmentation analysis, the algorithm is given extra resources over the adversary and the competitive ratio is bounded. A s-speed c-competitive algorithm is given a processor s times faster than the optimal solution and achieves a competitive ratio of c. The seminal scheduling resource augmentation paper considered the throughput scheduling problem and gave the best possible $(1 + \epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive algorithm for any fixed $\epsilon > 0$ [12].

Since this work, there has been an effort to understand and develop algorithms for more general scheduling environments and objectives. In the identical machine setting where the jobs can be scheduled on m identical parallel machines (processors), a $(1+\epsilon)$-speed $O(1)$-competitive algorithm is known for fixed $\epsilon > 0$ [13]. This has been extended to the case where the machines have speed scalable processors and the scheduler is energy aware [14]. In the related machines and unrelated machines settings, similar results have been obtained as well [15]. In [16] similar results were obtained in a distributed model.

None of this prior work consider parallel jobs. Parallel jobs modeled as DAGs have been widely considered in scheduling theory for other objectives [17,18,19,20,21,22,23,24]. There has been an extensive study in the real-time system community on how to schedule parallelizable DAG jobs by their deadlines. See [25,26,27,28,29,17,18,30,31] for pointers to relevant work. These works consider different (yet similar) objectives, focusing on tests to determine if a given set of reoccurring jobs can *all* be completed by their deadline, in contrast to optimizing throughput or profit.

Results: In this paper, we give the *first* non-trivial results for scheduling parallelizable DAG jobs online to maximize throughput and then we generalize these results to the general profit problem. Two important parameters in the DAG setting are the critical-path length L_i of job J_i (its execution time on an infinite number of processors) and its total work W_i (its uninterrupted execution time on a single processor). The value of $\max\{L_i, W_i/m\}$ is a lower bound on the amount of time any 1-speed scheduler takes to complete job J_i on m cores. We will focus on schedulers that are aware of the values of L_i and W_i when the job arrives, but are unaware of the internal structure of the job's DAG. That is, besides L_i and W_i, the only other information a scheduler has on a job's DAG is which nodes are currently available to execute. We call such an algorithm *semi-non-clairvoyant* — for DAG tasks, this is a reasonable model for the real world programs written in languages mentioned above since the DAG generally unfolds dynamically as the program executes. We first state a simple theorem about these schedulers.

Theorem 1. *There exists inputs where any semi-non-clairvoyant scheduler requires speed augmentation of* $2 - 1/m$ *to be* $O(1)$*-competitive for maximizing throughput.*

Roughly speaking, scheduling even a single DAG job in time smaller than $\frac{W_i - L_i}{m} + L_i$ is a hard problem even offline when the entire job structure is known in advance. This is captured by the classic problem of scheduling a precedence constrained jobs to minimize the makespan. For this problem, there is no $2 - \epsilon$ approximation assuming a variant of the unique games conjecture [32]. In particular, in Section 4, we will give an example DAG where any semi-non-clairvoyant scheduler will take roughly $\frac{W_i - L_i}{m} + L_i$ time to complete, while a fully clairvoyant scheduler can finish in time W_i/m. By setting the relative deadline to be $D_i = W_i/m = L_i$, every semi-non-clairvoyant scheduler will require a speed augmentation of $2 - 1/m$ to have bounded competitiveness.

With the previous theorem in place, we cannot hope for a $(1+\epsilon)$-speed $O(1)$-competitive algorithm. To circumvent this hurdle, one could hope to show $O(1)$-competitiveness by either using more resource augmentation or by making an assumption on the input. Intuitively, the hardness comes from having a relative deadline D_i close to $\max\{L_i, W_i/m\}$. In practice, this is unlikely to be the case. We show that so long as $D_i \geq (1+\epsilon)(\frac{W_i - L_i}{m} + L_i)$ then there is a $O(\frac{1}{\epsilon^6})$-competitive algorithm.

Theorem 2. *If for every job* J_i *it is the case that* $(1+\epsilon)(\frac{W_i - L_i}{m} + L_i) \leq D_i$*, then there is a* $O(\frac{1}{\epsilon^6})$*-competitive algorithm for maximizing throughput.*

We note that this immediately implies the following corollary without any assumptions on the input.

Corollary 1. *There is a* $(2+\epsilon)$*-speed* $O(\frac{1}{\epsilon^6})$*-competitive algorithm for maximizing throughput.*

Proof. No schedule can finish a job J_i if its relative deadline is smaller than $\max\{L_i, \frac{W_i}{m}\}$ and we may assume that no such job exists. Using this, we have that $(\frac{W_i}{m} + L_i) \leq 2D_i$. Consider transforming the problem instance giving the algorithm *and* the optimal solution together $2+\epsilon$ speed. In this case, the condition of Theorem 2 is met since we can view this as scaling the work in each node of the jobs by $2+\epsilon$. This scales the work and critical-path length by $2+\epsilon$. The corollary follows by observing that in this case we are comparing to an optimal solution with $2+\epsilon$ speed which is only stronger than comparing to an optimal solution with 1 speed. □

We note that the theorem also immediately implies the following corollary for "reasonable jobs."

Corollary 2. *There is a* $(1+\epsilon)$*-speed* $O(\frac{1}{\epsilon^6})$*-competitive for maximizing throughput if* $(W_i - L_i)/m + L_i \leq D_i$ *for all jobs* J_i*.*

This assumption on the deadlines is reasonable since, as we show in Section 4, there exists inputs for which even the optimal semi-non-clairvoyant scheduler has unbounded performance if the deadline is smaller.

We go on to consider the general profit scheduling problem. We first make the following assumption, which is that for all jobs J_i its general profit function satisfies $p_i(d) = p_i(x_i^*)$, where $0 < d \leq x_i^*$ for some $x_i^* \geq (1+\epsilon)(\frac{W_i - L_i}{m} + L_i)$. This assumption states that there is no additional benefit for completing a job J_i before time x_i^*, which is the natural generalization of our assumption in the throughput case. The function is arbitrarily decreasing otherwise. Using this, we show the following.

Theorem 3. *If for every job J_i it is the case that $p_i(d) = p_i(x_i^*)$, where $0 < d \leq x_i^*$ for some value of $x_i^* \geq (1+\epsilon)(\frac{W_i - L_i}{m} + L_i)$ then there is a $O(\frac{1}{\epsilon^6})$-competitive algorithm for the general profit objective.*

This gives the following corollary, just as for throughput.

Corollary 3. *There is a $(2+\epsilon)$-speed $O(\frac{1}{\epsilon^6})$-competitive algorithm for maximizing general profit.*

2 Preliminaries

In the problem considered, there is a set $\mathcal{J}$ of n jobs $\{J_1, J_2, ..., J_n\}$ which arrive online. The jobs are scheduled on m identical processors. Job J_i arrives at time r_i. Let $p_i(t)$ be an arbitrary non-negative non-increasing function for job J_i. The value of $p_i(t)$ is the profit obtained by completing job i at time $r_i + t$. Under some schedule, let t_i be the time it takes to complete J_i after its arrival. The goal is for the scheduler to maximize $\sum_{i \in [n]} p_i(t_i)$.

A special case of this problem is scheduling jobs with deadlines. In this problem, each job J_i has a deadline d_i and obtains a profit of p_i if it is completed by this time. In this case, we let $D_i = d_i - r_i$ be the relative deadline of the job. To make the underlying ideas of our approach clear, we will first focus on proving this case and the more general problem can be found in the Section 5.

Each job is represented by a Directed-Acyclic-Graph (DAG). A node in the DAG is *ready* to execute if all its predecessors have completed. A job is *completed* only when *all* nodes in the job's DAG have been processed. We assume the scheduler knows the ready nodes for a job at any point in time, but does not know the entire DAG structure a priori. Any set of ready nodes can be processed at once, but each processor can only execute one node at a time.

A DAG job has two important parameters. The total *work* W_i is the sum of the processing time of the nodes in job i's DAG. The *span* or *critical-path-length* L_i is the length of the longest path in job i's DAG, where the length of the path is the sum of the processing time of nodes on the path. To show Theorem 2 we assume that $(1+\epsilon)(\frac{W_i - L_i}{m} + L_i) \leq D_i$ for all jobs J_i throughout the remainder of the paper.

3 Jobs with Deadlines

First, we give an algorithm and analysis proving Theorem 2 when jobs have deadlines and profits. To aid the reader, a list of notation can be found in Tables 1, 2 and 3. Throughout the proof, we let C^O denote the jobs that the optimal solution completes by their deadline and let $\left\|C^O\right\|$ denote the total profit obtained by the optimal solution. Our goal is to design a scheduler that achieves profit close to $\left\|C^O\right\|$. Throughout the proof, it will be useful to discuss the aggregate number of processors assigned to a job over all time. We define a *processor step* to be a unit of time on a single processor.

3.1 Algorithm

In this section, we introduce our algorithm S. On every time step, S must decide which jobs to schedule and which ready nodes of each job to schedule. When a job J_i arrives, S calculates n_i — the number of processors "allocated" to J_i. On any time step when S decides to run J_i, it will always allocate n_i processors to J_i. In addition, since S is semi-non-clairvoyant, it is unable to distinguish between ready nodes of J_i; when it decides to allocate n_i nodes to J_i, it arbitrarily picks n_i ready nodes to execute if more than n_i nodes are ready.

We first state some observations regarding work and critical-path length.

Observation 1 *If a job J_i has all of its r ready nodes being executed by a schedule with speed s on m processors, where $r \leq m$, then the remaining critical-path length of J_i decreases at a rate of s.*

As mentioned earlier, we assume that the deadline for each job follows the condition that $(1+\epsilon)(\frac{W_i - L_i}{m} + L_i) \leq D_i$ for some positive constant ϵ.

We define the following **constants**. Let $\delta < \epsilon/2$, $c \geq 1 + \frac{1}{\delta\epsilon}$ and $b = (\frac{1+2\delta}{1+\epsilon})^{1/2} < 1$ be fixed constants. For each job J_i, the algorithm calculates n_i as $\frac{(W_i - L_i)}{\frac{D_i}{1+2\delta} - L_i}$. The value of n_i is the number of processors our algorithm will give to job J_i if we decide to execute J_i on some time step.

Let $x_i := \frac{W_i - L_i}{n_i} + L_i$. By Observation 1 it is the case that if n_i processors are given to job i for x_i units of time then the job will be completed regardless of the order the nodes are executed in. We will consider this to be Observation 2.

Observation 2 *Job J_i can meet its deadline if it is given n_i dedicated processors for x_i time steps in the interval $[r_i, d_i]$.*

We define the ***density*** of a job as $v_i = \frac{p_i}{x_i n_i}$. Note that this is a non-standard definition of density. We define the density as $\frac{p_i}{x_i n_i}$ instead of $\frac{p_i}{W_i}$, because we will think of job i requiring $x_i n_i$ processor steps to complete by Scheduler S. Thus, this definition of density indicates the potential profit per processor step that S can obtain by executing J_i.

The scheduler S maintains jobs that have arrived but are unfinished in two priority queues. A priority queue Q stores all the jobs that have been ***started*** by S. In the priority queue, the jobs are sorted according to the density from high to low. Another priority queue P stores all the jobs that have arrived but have not been started by S. Jobs in P are also sorted according to their densities from high to low.

Job Execution: At each time step t, S picks a set of jobs in Q to execute, in order from highest to lowest density. If a job J_i has been completed or if its absolute deadline d_i has passed ($d_i > t$), S removes the job from Q. When considering job J_i, if the number of unallocated processors is at least n_i the scheduler assigns n_i processors to J_i for execution. Otherwise, it continues on to the next job. S stops this procedure when either all jobs have been considered or when there are no remaining processors to allocate.

We introduce some notations to describe how jobs are moved from queue P to Q. A job J_i is δ-**good** if $D_i \geq (1 + 2\delta)x_i$. A job is δ-**fresh** at time t if $d_i - t \geq (1 + \delta)x_i$. For any set T of jobs, let the set $A(T, v_1, v_2)$ contains all jobs in T with density within the range $[v_1, v_2)$. We define $N(T, v_1, v_2) = \sum_{J_i \in A(T,v_1,v_2)} n_i$. This is the total number of processors that S allocates to jobs in $A(T, v_1, v_2)$. We will say that the set of job $A(T, v_1, v_2)$ *requires* $N(T, v_1, v_2)$ processors.

Adding Jobs to Q: There are two types of events that may cause S to add a job to Q. These events occur when either a job arrives or S completes a job. When a job J_i arrives, S adds it to queue Q if it satisfies the following conditions:

(1) J_i is δ-good;

(2) For all job $J_j \in Q \cup \{J_i\}$ it is the case that $N\left(Q \cup \{J_i\}, v_j, cv_j\right) \leq bm$. In words, the total number of processors required by jobs in $Q \cup \{J_i\}$ with density in the range $[v_j, cv_j)$ is no more than bm.

If these conditions are met, then J_i is inserted into queue Q; otherwise, job J_i is inserted into queue P. When a job is added to Q, we say that the job is *started* by S.

At the completion of a job, S considers the jobs in P from highest to lowest density. S first removes all jobs with absolute deadlines that have already passed. Then S checks if a job J_i in P can be moved to queue Q by checking whether job J_i is δ-fresh and condition (2) from above. If both the conditions are met, then J_i is moved from queue P to queue Q.

Remark: Note that the Scheduler S pre-computes a fixed number of processors n_i assigned to each job, which may seem strange at first glance. This is because that n_i is approximately the minimum number of dedicated cores job J_i requires to complete by $\frac{D_i}{1+2\delta} \to D_i$, without knowing J_i's DAG structure. In addition, as long as J_i can complete by its deadline, it obtains the same profit p_i. Therefore, there is no need to complete J_i earlier by executing J_i on more dedicated cores. Moreover, by carefully assigning n_i, we are able to bound the number of processor steps spent on job J_i as shown in Lemma 3, which is critical for bounding the profit obtained by the optimal solution.

Outline of the Analysis of S: Our goal is to bound the total profit that S obtains. We first discuss some basic properties of S in Section 3.2. In Section 3.3 be bound the total profit of all the jobs S starts by the total profit of jobs that S completes. Then in Section 3.4 we bound the total profit of the jobs the optimal solution completes by the total profit of jobs that S starts. Putting these two together, we are able to bound the performance of S.

3.2 Properties of the Scheduler

We begin by showing some structural properties for S that we will leverage in the proof. We first bound the number of processors n_i that S will allocate to job J_i.

Lemma 1. *For every job J_i we have that $n_i \leq b^2 m$.*

Proof. By assumption we know that $D_i \geq (1+\epsilon)(\frac{W_i - L_i}{m} + L_i)$. The definition of n_i gives the following.

$$n_i = \frac{W_i - L_i}{\frac{D_i}{1+2\delta} - L_i} \leq \frac{W_i - L_i}{\frac{1+\epsilon}{1+2\delta}(\frac{W_i - L_i}{m} + L_i) - L_i} \leq \frac{1+2\delta}{1+\epsilon} m = b^2 m$$

□

Lemma 2. *Every job J_i is δ-good, i.e. $x_i(1+2\delta) \leq D_i$.*

Proof. Note that $L_i \leq \frac{1}{1+\epsilon} D_i$ by definition. Since $n_i = \frac{W_i - L_i}{\frac{D}{1+2\delta} - L_i}$, we have $x_i(1+2\delta) = (\frac{W_i - L_i}{n_i} + L_i)(1+2\delta) = (\frac{D_i}{1+2\delta} - L_i + L_i)(1+2\delta) \leq D_i$. □

The next lemma bounds the total number of processor steps occupied by a job.

Lemma 3. *$x_i n_i \leq a W_i$, where a is $1 + \frac{1+2\delta}{\epsilon - 2\delta}$.*

Proof. By definition we have

$$\begin{aligned} x_i n_i = W_i - L_i + n_i L_i &\leq W_i + \frac{W_i - L_i}{\frac{D_i}{1+2\delta} - L_i} L_i \leq W_i + \frac{W_i - L_i}{\frac{D_i}{1+2\delta} - \frac{D_i}{1+\epsilon}} \Big(\frac{D_i}{1+\epsilon}\Big) \\ &\leq W_i + \frac{(W_i - L_i) D_i (1+2\delta)}{D_i(\epsilon - 2\delta)} \leq W_i + \frac{W_i(1+2\delta)}{\epsilon - 2\delta} \leq W_i \Big(1 + \frac{1+2\delta}{\epsilon - 2\delta}\Big) \end{aligned}$$

□

Observation 3 *At any time and for any $v > 0$, the total number of processors required by all the jobs J_i that are in queue Q and have density $v \leq v_i < cv$ is no more than bm, i.e. $N(Q, v_i, cv_i) \leq bm$.*

Proof. Jobs are only added to queue Q when a new job arrives or a job completes. According to algorithm S, at both times, a job is only added to Q when this condition is satisfied. □

3.3 Bounding the Profit of Jobs S Completes by All Jobs Started by S

In this section, we bound the profit of jobs completed by S compared to the profit of all jobs it ever starts (adds to Q). Let R denote the set of jobs S starts (that is, the set of jobs added to queue Q). Among the jobs in R, let C be the set of jobs it completes and U be the set of jobs that are unfinished. We say job J_i (and its assigned processors) is v**-dense**, if its density $v_i \geq v$. For any set A of jobs, define $\|A\|$ as $\sum_{i \in A} p_i$, the sum of the profits of jobs in the set.

Lemma 4. *For a job $J_i \in U = R \setminus C$ that was added to queue Q but does not complete by its deadline, S must have run cv_i-dense jobs for at least δx_i time steps where J_i is in Q using at least $(1-b)m$ processors at each such time.*

Proof. Since J_i is at least δ-fresh when added to Q and it does not complete by its deadline, there are at least δx_i time steps where S is not executing J_i by Observation 2. In each of these the time steps, all the m processors are executing v_i-dense jobs.

By Observation 3, jobs in Q with density in range $[v_i, cv_i)$ require at most $N(Q, v_i, cv_i) \leq bm$ processors to execute. Therefore, for each of the δx_i time steps, there are at least $(1-b)m$ processors executing cv_i-dense jobs. So the total number processor steps where cv_i-dense jobs are executing is at least $\delta x_i (1-b)m$. □

We now bound the profit of the jobs completed by their deadline under S by those started.

Lemma 5. $\|C\| \geq (\epsilon - \frac{1}{(c-1)\delta}) \|R\|$.

Proof. We use a charging scheme with credit transfers between the jobs. We give each job $J_i \in R$ a bank account B_i. Initially, all completed jobs (in C) are given p_i credits and other jobs (in U) have 0 credit. We will transfer credits between jobs in C and jobs in U. We want to show that after the credit transfer, every job J_i in R will have $B_i \geq (\epsilon - \frac{1}{(c-1)\delta})p_i$. This implies $\|C\| \geq (\epsilon - \frac{1}{(c-1)\delta}) \|R\|$.

Now we explain how credits are transferred. For each time step, a processor executing J_i will transfer $\frac{v_j n_j}{\delta bm}$ credits from B_i to every job J_j in queue Q that has density $v_j \leq \frac{v_i}{c}$.

For every job $J_j \in U$, Lemma 4 implies that there are at least δx_j time steps where at least $(1-b)m$ processors are executing cv_j-dense jobs. By our credit transfer strategy J_j will receive at least $\frac{v_j n_j}{\delta bm}$ credits from each processor in a time step. Therefore, the total credits J_j receives is at least

$$\delta x_j (1-b)m(\frac{v_j n_j}{\delta bm}) = v_j x_j n_j (\frac{1-b}{b}) = p_i(\frac{1-b}{b}).$$

This bounds the total amount of credit each job receives. We now show that not too much credit is transferred out of each job's account. We bound this on a job by job basis. Fix a job $J_i \in R$ and consider how many credits it transfers to other jobs during its execution. By Observation 2, we know that J_i can execute for at most x_i time steps on n_i dedicated processors before its completion.

The job J_i will transfer credit to all jobs in Q with density less than $\frac{v_i}{c}$ at any point in time where J_i is being processed. These are the jobs in $A(Q, 0, \frac{v_i}{c})$. Fix an integer $l \geq 1$ and consider the set of jobs $A(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})$ in Q that have density within the range $[\frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})$. Note that the total number of processors required by them is $N(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l}) \leq bm$ by Observation 3. Knowing that a job J_j in $A(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})$ has density $v_j \leq \frac{v_i}{c^l}$ by definition it is the case that the total credits that J_i gives to jobs in $A(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})$ per processor assigned to J_i during any time step is at most

$$\begin{aligned}\sum_{J_j \in A(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})} \frac{v_j n_j}{\delta bm} &\leq \sum_{J_j \in A(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})} \frac{\frac{v_i}{c^l} n_j}{\delta bm} = \frac{v_i}{\delta bmc^l} \sum_{J_j \in A(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})} n_j \\ &= \frac{v_i}{\delta bmc^l} N(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l}) \leq \frac{v_i}{\delta bmc^l} bm = \frac{v_i}{\delta c^l}.\end{aligned}$$

This bounds the total credit transferred to jobs in $A(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})$ during a time step for each processor assigned to J_i. We sum this quantity over all $l \geq 1$ and all n_i processors assigned to i to bound the total credit transferred from job J_i during a time step. Recall that $c > 1$ by definition.

$$\frac{n_i v_i}{\delta} \sum_{l=1}^{\infty} \frac{1}{c^l} = \left(\frac{n_i v_i}{\delta}\right) \frac{\frac{1}{c}}{1 - \frac{1}{c}} = \left(\frac{n_i v_i}{\delta}\right) \frac{1}{c-1}$$

Therefore, the total credits J_i transfers to all the jobs in $A(Q, 0, \frac{v_i}{c})$ over all times it is executed is at most $(\frac{x_i n_i v_i}{\delta})\frac{1}{c-1} = \frac{p_i}{(c-1)\delta}$ due to the fact that a job will be executed for at most x_i time steps in S's schedule.

Now we put these two observations together. Each job receives at least $p_i \frac{1-b}{b}$ credit and pays at most $\frac{p_i}{(c-1)\delta}$. After the credit transfer, the credits that a job J_i has is at least

$$p_i \frac{1-b}{b} - \frac{p_i}{(c-1)\delta} = p_i(\epsilon - \frac{1}{(c-1)\delta})$$

By our setting of c, this quantity is always positive. Therefore, we conclude that $\|C\| \geq (\epsilon - \frac{1}{(c-1)\delta}) \|R\|$. □

3.4 Bounding the Profit of Jobs OPT Completes by All Jobs Started by S

In this section, we bound the profit of the jobs OPT completes by all of the jobs that S starts. Our high level goal is to first bound the total amount of time OPT spends processing jobs that S does not complete by the time S spends processing jobs. Then using this and properties of S we will be able to bound the total profit of jobs OPT completes. At a high level, this follows since S focuses on processing high density jobs and OPT and S spend a similar amount of time processing jobs. We begin by showing that if not too many processors are executing $\frac{v_i}{c}$-dense jobs then all such jobs must be currently executing.

Lemma 6. *For any density v_i and time, if there are less than $b(1-b)m$ processors executing $\frac{v_i}{c}$-dense jobs, then all $\frac{v_i}{c}$-dense jobs in queue Q are executing and $N(Q, \frac{v_i}{c}, \infty) < b(1-b)m$.*

Proof. By definition, there are at least $m - b(1-b)m > bm - b(1-b)m = b^2m$ processors executing jobs with density less than $\frac{v_i}{c}$. For the sake of contradiction, suppose there is a $\frac{v_i}{c}$-dense job J_j that is not executing by S. By Lemma 1 we know that $n_j \leq b^2m$. Therefore, J_j would have been executed by S on the b^2m processors that are executing lower density jobs, a contradiction.

Now we know all all $\frac{v_i}{c}$-dense jobs in queue Q are executing. By assumption they are using less than $b(1-b)m$ processors and the lemma follows. □

In the next lemma, we show that if not too many processors are running $\frac{v_i}{c}$-dense jobs then when a job arrives or completes, the schedule S will start processing a v_i-dense job that is δ-fresh, for any density v_i (if such a job exists). In particular, the job J_j will pass condition (2) of for adding jobs to Q in the definition of S.

Lemma 7. *Fix a density v_i. At a time where a new job arrives or a job completes if there are less than $b(1-b)m$ processors executing $\frac{v_i}{c}$-dense jobs, then a δ-fresh v_i-dense job J_j (arriving or in queue P) will be added to Q by S assuming such a job J_j exists.*

Proof. By Lemma 6, we know that all $\frac{v_i}{c}$-dense jobs in queue Q are executing on less than $b(1-b)m$ processors. By Lemma 1, we know that $n_j \leq b^2m$. Therefore,

$$N(Q \cup \{J_j\}, \frac{v_i}{c}, \infty) < b(1-b)m + b^2m = bm$$

Consider any δ-fresh job J_j that is also v_i-dense. Consider any job J_k where $J_j \in A(Q \cup \{J_i\}, v_k, cv_k)$. By definition of J_j being v_i-dense it must be the case that $A(Q \cup \{J_i\}, v_k, cv_k) \subseteq A(Q \cup \{J_j\}, \frac{v_i}{c}, \infty)$. The above implies that $N(Q \cup \{J_i\}, v_k, cv_k) \leq N(Q \cup \{J_j\}, \frac{v_i}{c}, \infty) \leq bm$. Thus, the condition (2) in our algorithm is satisfied. □

For an arbitrary set of jobs $\mathcal{E}$ and any $v \geq 0$ let $T_O(v, \mathcal{E})$ denote the total work processed by the optimal schedule for the jobs in $\mathcal{E}$ that are v-dense. We similarly let $T_S(v, \mathcal{E})$ be the total number of processors steps S used for executing jobs in $\mathcal{E}$ that are v-dense over all time. Now we are ready to bound the time that OPT spends on jobs that S never adds to Q.

Lemma 8. *Consider the jobs in $\mathcal{J} \setminus R$, the jobs that are never added to Q. For all $v > 0$, $T_O(v, \mathcal{J} \setminus R) \leq \frac{1+2\delta}{\delta b(1-b)} T_S(\frac{v}{c}, \mathcal{J})$.*

Proof. Let $\{I_k = [s_k, e_k]\}$ be the set of maximal time intervals where at least $b(1-b)m$ processors are running $\frac{v}{c}$-dense jobs in S's schedule. Notice that by definition $\sum_{k=1}^{\infty}(e_k - s_k)b(1-b)m \leq T_S(\frac{v}{c}, \mathcal{J})$.

Consider a job in $J_i \in \mathcal{J} \setminus R$ that is both δ-good and v-dense and additionally arrives during $[s_k, s_{k+1})$. Note that during the intervals $[e_k, s_{k+1}]$, less than $b(1-b)m$ processors are executing $\frac{v}{c}$-dense jobs. Lemma 7 implies that if J_i arrives during $[e_k, s_{k+1}]$ it will be added to Q. This contradicts the assumption that $J_i \in \mathcal{J} \setminus R$. Therefore, J_i must arrive during $[s_k, e_k)$ and is in queue P at time e_k.

Note that at time e_k, the number of processors executing $\frac{v}{c}$-dense jobs decreases, so there must be a job that completes at time e_k. Again, by Lemma 7 if J_i is δ-fresh at time e_k then it will be added to Q at this time. Again, this contradicts $J_i \in \mathcal{J} \setminus R$. Thus, the only reason that S does not add J_i to Q is because J_i is not δ-fresh at time e_k. Knowing that J_i is δ-good at r_i and is not δ-fresh at e_k, we have $e_k - s_k \geq e_k - r_i \geq \delta x_i$.

At time e_k, J_i is not δ-fresh, so $d_i - e_k < (1+\delta)x_i < \frac{1+\delta}{\delta}(e_k - s_k)$.

Let K_k be the set of v-dense jobs that arrive during $[s_k, s_{k+1})$ but are not completed by S. Because OPT can only execute all jobs in K_k during $[s_k, d_i]$ on at most m processors, we get

$$T_O(v, K_k) \leq (d_i - s_k)m = ((d_i - e_k) + (e_k - s_k))m \leq \frac{1+2\delta}{\delta}(e_k - s_k)m$$

This completes the proof, as

$$T_O(v, U) = \sum_{k=1}^{\infty} T_O(v, K_k) \leq \sum_{k=1}^{\infty} (\frac{1+2\delta}{\delta}) m(e_k - s_k) \leq \frac{1+2\delta}{\delta} \frac{1}{b(1-b)} T_S(\frac{v}{c}, \mathcal{J})$$

□

Using the previous lemma, we can bound the profit of jobs completed by OPT by the profit of jobs started by S.

Lemma 9.

$$\left\|C^O\right\| \leq \left(1 + (1 + \frac{1+2\delta}{\epsilon - 2\delta})(1 + \frac{1}{\epsilon\delta})\frac{1+2\delta}{\delta b(1-b)}\right)\|R\|$$

.

Proof. We may assume WLOG that the adversary completes all jobs it starts. First we partition C^O, the jobs that the adversary completes, into C^O_R and C^O_S where $C^O_S = C^O \cap R$, that is, our algorithm started the job at some point. The remaining jobs are placed in C^O_R. Clearly $\left\|C^O_S\right\| \leq \|R\|$. Now it remains to bound $\left\|C^O_R\right\|$.

Consider every job in C^O_R and let the set of densities of these jobs be $\{\mu_1, \mu_2, \ldots, \mu_m\}$ from high to low and for notational simplicity let $\mu_0 = \infty$ and $\mu_{m+1} = 0$. Recall the adversary completed all jobs it started. Thus for each job with density μ_i, the adversary ran the job for a corresponding W_i processor steps. Let β_i denote the number of processor steps our algorithm takes to run jobs with densities within $(\frac{\mu_{i-1}}{c}, \frac{\mu_i}{c}]$.

We have $T_O(v, \mathcal{J} \setminus R) \leq \frac{1+2\delta}{\delta b(1-b)} T_S(\frac{v}{c}, \mathcal{J})$ from Lemma 8 for all densities v. Equivalently for any given density v:

$$T_O(v, \mathcal{J} \setminus R) = \sum_{i=1}^{v} W_i \leq \frac{1+2\delta}{\delta b(1-b)} \sum_{i=1}^{v} \beta_i = \frac{1+2\delta}{\delta b(1-b)} T_S(\frac{v}{c}, \mathcal{J})$$

We then sum over all densities. The subtraction of densities is necessary to insure that each density is only counted a single time.

$$\sum_{v=1}^{m}\left((\mu_v - \mu_{v+1})\sum_{i=1}^{v} W_i\right) \leq \sum_{v=1}^{m}\left((\mu_v - \mu_{v+1})\frac{1+2\delta}{\delta b(1-b)}\sum_{i=1}^{v}\beta_i\right)$$

The LHS can be simplified:

$$\sum_{v=1}^{m}\left((\mu_v - \mu_{v+1})\sum_{i=1}^{v} W_i\right) = \sum_{i=1}^{m} W_i \sum_{v=i}^{m}(\mu_v - \mu_{v+1}) = \sum_{i=1}^{m} W_i(\mu_i - \mu_{m+1}) = \sum_{i=1}^{m} W_i\mu_i$$

The RHS similarly simplifies to $\frac{1+2\delta}{\delta b(1-b)}\sum_{i=1}^{m}\beta_i\mu_i$, leading to the inequality that $\sum_{i=1}^{m} W_i\mu_i \leq \frac{1+2\delta}{\delta b(1-b)}\sum_{i=1}^{m}\beta_i\mu_i$. Recall that densities such as μ_i are defined by $\mu_i = \frac{p_i}{x_i n_i}$ and $x_i n_i \leq aW_i$. Therefore:

$$\sum_{i=1}^{m} W_i\mu_i = \sum_{i=1}^{m}\frac{W_i p_i}{x_i n_i} \geq \sum_{i=1}^{m}\frac{W_i p_i}{aW_i} \geq \sum_{i=1}^{m}\frac{p_i}{(1 + \frac{1+2\delta}{\epsilon-2\delta})} = \frac{1}{(1 + \frac{1+2\delta}{\epsilon-2\delta})}\left\|C^O_R\right\|$$

And also, by the definition of β_i, we know that $\sum_{i=1}^{m}\beta_i\frac{\mu_i}{c} \leq \|R\|$.

Combining these results, we get:

$$\frac{1}{(1 + \frac{1+2\delta}{\epsilon-2\delta})}\left\|C^O_R\right\| \leq \sum_{i=1}^{m} W_i\mu_i \leq \frac{1+2\delta}{\delta b(1-b)}\sum_{i=1}^{m}\beta_i\mu_i \leq \frac{1+2\delta}{\delta b(1-b)} c\,\|R\|$$
$$\Rightarrow \left\|C^O_R\right\| \leq \left(1 + \frac{1+2\delta}{\epsilon-2\delta}\right)\left(\frac{1+2\delta}{\delta b(1-b)}\right) c\,\|R\|$$
$$\Rightarrow \left\|C^O\right\| = \left\|C^O_R\right\| + \left\|C^O_S\right\| \leq \left(1 + (1 + \frac{1+2\delta}{\epsilon-2\delta})(1 + \frac{1}{\epsilon\delta})\frac{1+2\delta}{\delta b(1-b)}\right)\|R\|$$

□

Finally we are ready to complete the proof, bounding the profit OPT obtains by the total profit the algorithm obtains for jobs it completed.

Lemma 10.

$$\left\|C^O\right\| \leq \frac{\left(1 + (1 + \frac{1+2\delta}{\epsilon-2\delta})(1 + \frac{1}{\epsilon\delta})\frac{1+2\delta}{\delta b(1-b)}\right)}{\epsilon - \frac{1}{(c-1)\delta}}\|C\|$$

Proof. This is just by combination of Lemma 5 and Lemma 9. □

Therefore, we prove Theorem 2 by showing that scheduler S is $O(\frac{1}{\epsilon^6})$-competitive for jobs with deadlines and profits, when $(1+\epsilon)(\frac{W_i - L_i}{m} + L_i) \leq D_i$.

4 Examples

In this section, we will give some example DAGs to show why Theorem 2 is close to the best theorem we can hope for using two examples. The first example, shown in Figure 1, shows the limitations of semi-non-clairvoyance. In particular, a semi-non-clairvoyant scheduler does not know the structure of the DAG in advance since the DAG unfolds dynamically. At any time step, the scheduler only knows the ready nodes available for execution. Given this limitation, consider the DAG shown in Figure 1. This job has one sequential chain with length $L = \frac{W}{m}$, where W is the total work of the job and m is the number of processors. The remaining $W - W/m$ work are fully parallelizable in a block and can also be done in parallel with the chain. Therefore, L is the span of the jobs.

Since a semi-non-clairvoyant scheduler cannot distinguish between ready nodes, it may make unlucky choices and execute the entire block of $W - W/m = W - L$ ready nodes first in $(W - L)/m$ time steps and then execute the chain of L nodes sequentially — leading to a total time of $(W - L)/m + L$. On the other hand, a fully clairvoyant scheduler can execute the entire DAG in W/m time. Therefore, a semi-non-clairvoyant scheduler needs at least $2 - 1/m$ speed augmentation to ensure that it can complete the DAG at the same time as OPT.

We now show another example DAG indicating that it would be reasonable to always set deadlines as $D \geq (W - L)/m + L$ if we do not know the structure of the DAG a priori. Figure 2 shows an example DAG, which consists of a chain of $L - \epsilon$ nodes followed by $W - L + \epsilon$ nodes that can run in parallel. Each node in the DAG takes ϵ time to run, so the total work of the DAG is W and the span is L. For such a DAG, even a fully clairvoyant scheduler needs $L - \epsilon + \frac{W-L+\epsilon}{m} = \frac{W-L}{m} + L - \epsilon(1 - \frac{1}{m})$, which approaches to $\frac{W-L}{m} + L$ when $\epsilon \to 0$.

Notation	**Definition**
OPT	optimal schedule and also optimal objective
m	the number of processors
W_i	the total work of job J_i
L_i	the span of job J_i
D_i	relative deadline of job J_i
r_i	the arrival time of J_i
d_i	the absolute deadline of J_i (that is, $r_i + D_i$)
$A(T, v_1, v_2)$	all jobs in T with density within the range $[v_1, v_2)$
$N(T, v_1, v_2)$	$= \sum_{J_i \in A(T,v_1,v_2)} n_i$, the total number of processors required by $A(T, v_1, v_2)$
v-dense	if Job J_i has density $v_i \geq v$
δ	$< \epsilon/2$
c	$\geq 1 + \frac{1}{\epsilon\delta}$
b	$= (\frac{1+2\delta}{1+\epsilon})^{1/2} < 1$
a	$= 1 + \frac{1+2\delta}{\epsilon-2\delta}$

Table 1: Notations and definitions throughout the paper

Notation	**Definition**
p_i	the profit of job J_i
n_i	$= \frac{(W_i - L_i)}{\frac{D_i}{1+2\delta} - L_i}$, the number of processors allocated to J_i
x_i	$= \frac{W_i - L_i}{n_i} + L_i$, the maximum execution time of J_i
v_i	$= \frac{p_i}{x_i n_i}$ the density of J_i
δ-good	job J_i has $D_i \geq (1 + 2\delta)x_i$
δ-fresh	at time t, job J_i has $d_i - t \geq (1 + \delta)x_i$
R	the set of jobs started by S
C	the set of jobs completed by S
U	unfinished jobs by S (that is, $R \setminus C$)
C^O	the set of jobs completed by OPT
$\mathcal{J}$	the set of all jobs
$T_O(v, \mathcal{E})$	the total work processed by the optimal schedule for the jobs in $\mathcal{E}$ that are v-dense
$T_S(v, \mathcal{E})$	the total number of processors steps S used for executing jobs in $\mathcal{E}$ that are v-dense

Table 2: Notations and definitions specific to jobs with deadlines

Notation	**Definition**
$p_i(t)$	the profit of job J_i if the job with arrival time r_i completes by $r_i + t$
n_i	$= \frac{(W_i - L_i)}{\frac{x_i^*}{1+2\delta} - L_i}$, the number of processors allocated to J_i
x_i	$= \frac{W_i - L_i}{n_i} + L_i$, the maximum execution time of J_i
v_i	$= \frac{p_i(D_i)}{x_i n_i}$ the density of J_i

Table 3: Notations and definitions specific to jobs with general profit functions

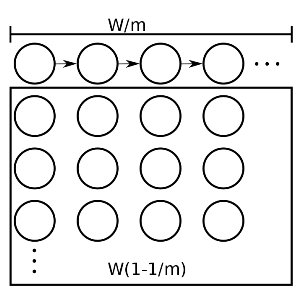

Fig. 1: Example 1

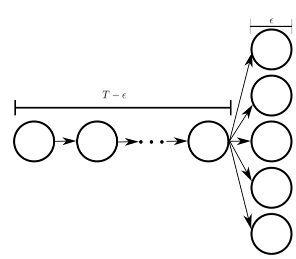

Fig. 2: Example 2

5 Jobs with General Profit Functions

In this section, we focus on a more general case. In particular, each job J_i has a non-negative non-increasing profit function $p_i(t)$ indicating its profit if the job with arrival time r_i completes by $r_i + t$. Our goal is to design a scheduler that maximizes the profit to make it close to what the optimal solution can obtain, denoted as $\|O\|$.

First, we present our scheduler S parameterized using a fixed constant $0 < \epsilon < 1$. Similar to Section 3.1, let $\delta < \epsilon/2$, $c \geq 1 + \frac{1}{\delta\epsilon}$ and $b = (\frac{1+2\delta}{1+\epsilon})^{1/2} < 1$ be fixed constants.

Upon the arrival of a job J_i, the scheduler S assigns a number of allocated cores n_i, a relative deadline D_i and a set of time steps I_i to J_i (according to the assignment procedure described below). In each time step t in I_i, we say that J_i is assigned to t. Scheduler S always executes the highest density jobs that is assigned to t. If S decides to execute J_i in a time step, it will give n_i processors to J_i. Let $x_i := \frac{W_i - L_i}{n_i} + L_i$. We define the ***density*** of a job as $v_i = \frac{p_i(D_i)}{x_i n_i} = \frac{p_i(D_i)}{W_i + (n_i - 1)L_i}$. We now formally specify the algorithm of scheduler S for job assignment and execution.

Assigning cores, deadlines and slots to jobs: When a job J_i arrives, the scheduler will assign a relative deadline D_i and a set of time steps I_i with n_i processors. These time steps are the only time steps in which J_i is allowed to run.

Recall (from Theorem 3) that we assume that the profit function stays the same until some value $x_i^* \geq (\frac{W_i - L_i}{m} + L_i)(1+\epsilon)$. The number of assigned processors n_i is calculated as $n_i = \frac{W_i - L_i}{\frac{x_i^*}{1+2\delta} - L_i}$. The assignment for D_i is determined by searching all the potential deadlines D to find the minimum valid deadline. The set of time steps I_i is determined using the chosen deadline D_i.

For each ***potential relative deadline*** $D > (1+\epsilon)L_i$, scheduler S checks whether it is a valid deadline by the following steps. First, it selects a set of time steps I. Assuming D is assigned to J_i, then the density of J_i is $v = \frac{p_i(D)}{W_i + (n_i - 1)L_i}$. For each time step t from r_i to $r_i + D$, let $\|I(t)\|$ be the number of time steps that have already been added to I before considering time step t. Let $J(t)$ denote the set of jobs that are currently has time t among its assignments. We only add t to the set I if it satisfies the following condition: For every job $J_j \in J(t)$, it is the case that $N\left(J(t) \cup \{J_i\}, v_j, cv_j\right) \leq bm$. In words, the total number of processors required by jobs in $J(t) \cup \{J_i\}$ with density in the range $[v_j, cv_j)$ is no more than bm.

I contains all the time steps during $[r_i, r_i + D_i)$ that can be assigned to J_i. If $\|I\| \geq (1+\delta)\left(\frac{W_i - L_i}{n_i} + L_i\right)$, which is at least δ times longer than the time J_i required to run on n_i processors, then the deadline D is said to be *valid*. Note that a valid assignment always exists by setting the deadline large enough.

Among all the valid assignments, S chooses the smallest valid deadline for J_i, which results in the highest profit. Given this deadline D_i, J_i will be assigned with the corresponding set I_i. Because D_i is the minimum valid deadline, the corresponding set I_i must satisfy $\|I_i\| = (1+\delta)\left(\frac{W_i - L_i}{n_i} + L_i\right)$; otherwise, there must exist a shorter deadline D that is also valid. Intuitively, with this assignment, J_i can complete by its deadline if no other jobs interfere. Note that J_i may not be completed by its deadline as we will allow higher density jobs that arrive after J_i to be scheduled during I_i.

Executing jobs: At each time step t, S picks a set of jobs in $J(t)$ to execute in order from highest to lowest density, where $J(t)$ are the set of jobs that have been assigned to time step t. That is, jobs J_i where $t \in I_i$. When considering job J_i, if the number of unallocated processors is at least n_i, then the scheduler allocates n_i processors to J_i. Otherwise, it continues on to the next job in $J(t)$. S stops this procedure when either all jobs have been considered or when there are no remaining processors to allocate.

Remark: Unlike the scheduler for jobs with deadlines, here we try to complete a job J_i by a calculated deadline D_i that is as close to x_i^* as possible. This is because the obtained profit decreases as the completion time increases but there is no additional benefit for completing a job J_i before time x_i^*. With a carefully designed deadline D_i, we are able to prove the performance bound of the scheduler. Similarly to Section 3, we start by stating the basic properties of the scheduler S, followed by bounding the total profit obtained by S. However, the proofs that bound the profit of jobs that are completed by OPT differ greatly from that for jobs with deadlines. This is because in addition to losing the profit of jobs that do not complete by their assigned deadlines, scheduler S can also loses profit compared to OPT if the completion time of a job under S is later than under OPT. By taking into account all these jobs, we are able to bound the performance of S for jobs with general profit functions.

5.1 Properties of the Scheduler

We begin by showing some structural properties for S that we will leverage in the proof and can be obtained directly from the algorithm of scheduler S. Note that these lemmas are similar to the lemmas shown in Section 3.2 if we replace x_i* with D_i. We state them here again for completeness.

Lemma 11. *For every job J_i we have that $n_i \leq b^2 m$, where $b = (\frac{1+2\delta}{1+\epsilon})^{1/2}$.*

Proof. By definition, we know that $x_i^* \geq (1+\epsilon)(\frac{W_i - L_i}{m} + L_i)$. Therefore, we have

$$n_i = \frac{W_i - L_i}{\frac{x_i^*}{1+2\delta} - L_i} \leq \frac{W_i - L_i}{\frac{1+\epsilon}{1+2\delta}(\frac{W_i - L_i}{m} + L_i) - L_i} \leq \frac{1+2\delta}{1+\epsilon} m = b^2 m$$

□

Lemma 12. *Under scheduler S, we have $x_i n_i \leq a W_i$ and $v_i \geq \frac{p_i(D_i)}{aW_i}$, where $a = 1 + \frac{1+2\delta}{\epsilon - 2\delta}$.*

Proof. By definition, $x_i^* > L_i(1+\epsilon)$. Therefore, we have

$$\begin{aligned} x_i n_i = W_i - L_i + n_i L_i = W_i + \frac{W_i - L_i}{\frac{x_i^*}{1+2\delta} - L_i} L_i &\leq W_i + \frac{W_i - L_i}{\frac{x_i^*}{1+2\delta} - \frac{x_i^*}{1+\epsilon}} \Big(\frac{x_i^*}{1+\epsilon}\Big) \\ &\leq W_i + \frac{(W_i - L_i) x_i^* (1+2\delta)}{x_i^*(\epsilon - 2\delta)} \leq W_i \Big(1 + \frac{1+2\delta}{\epsilon - 2\delta}\Big) \end{aligned}$$

Therefore, we have $v_i = \frac{p_i(D_i)}{x_i n_i} \geq \frac{p_i(D_i)}{aW_i}$. □

Lemma 13. *For every job J_i with the assignment n_i, D_i and I_i, Job J_i can meet its deadline D_i, if it is executed by S for at least x_i time steps in I_i (on n_i dedicated processors).*

Lemma 14. *For every job J_i, $x_i(1+2\delta) \leq x_i^*$.*

Proof. Note that $L_i \leq \frac{1}{1+\epsilon} D_i$ by requirement of potential assignment. Since $n_i = \frac{W_i - L_i}{\frac{x_i^*}{1+\epsilon} - L_i}$, we have $x_i(1+2\delta) = (\frac{W_i - L_i}{n_i} + L_i)(1+2\delta) \leq (\frac{x_i^*}{1+\epsilon} - L_i + L_i)(1+2\delta) = \frac{x_i^*}{1+\epsilon}(1+2\delta) \leq x_i^*$. □

Lemma 15. *At any time step t during the execution and for any density range $[v, cv)$, the total number of cores required by all the jobs $J_i \in J(t)$ (that have been assigned to t) with density $v \leq v_i < cv$ is no more than bm, i.e. $N(J(t), v_i, cv_i) \leq bm$.*

5.2 Bounding the Profit of Jobs S Completes

Similar to Section 3.3, we bound the profit of jobs completed by scheduler S compared to the profit of all jobs. Let $\mathcal{J}$ denote the set of jobs arrived during the execution, C denote the set of jobs that actually complete before their deadlines assigned by S, and $U = \mathcal{J} \setminus C$ be the set of jobs that didn't finish by their deadlines assigned by S. We say job J_i (and its assigned processors during execution) is **v-dense**, if its density $v_i \geq v$. For any set A of jobs, define $\|A\|$ as $\sum_{J_i \in A} p_i(D_i)$, the sum of the profits of jobs in the set under S.

Lemma 16. *For a job $J_i \in \mathcal{J} \setminus C$ that does not complete by its deadline, the number of time steps in I_i where S runs cv_i-dense jobs using at least $(1-b)m$ processors is at least δx_i.*

Proof. From Lemma 13, we know that job J_i can complete if it can execute for x_i time steps by S. Also note that according to the assignment process $(1+\delta)x_i = \|I_i\|$, where $\|I_i\|$ is the number of time steps assigned to J_i during $[r_i, r_i + D_i]$. Since it does not complete by its deadline, there are at least δx_i time steps in I_i where S does not execute J_i. Consider each of these time steps t. According to Lemma 15, jobs in $J(t)$ with density in range $[v_i, cv_i)$ require at most $N(J(t), v_i, cv_i) \leq bm$ processors to execute. Therefore, there must be at least $(1-b)m$ processors executing cv_i-dense jobs. Otherwise, S would execute all jobs in $A(J(t), v_i, cv_i)$, which includes job J_i. □

Lemma 17. $\|C\| \geq (\epsilon - \frac{1}{(c-1)\delta}) \|\mathcal{J}\|$.

The proof is similar to that of Lemma 5 and is omitted for brevity.

5.3 Bounding the Profit of Jobs OPT Completes

Similar to Section 3.4, we will now bound the profit of the jobs OPT completes. We are first going to consider the number of processor steps OPT spends on jobs that S finishes later than OPT. For these jobs, we assume that S makes no profit since the profit function may become 0 as soon as OPT finishes it. Our high level goal is to first bound the total number of processor steps OPT spends on these jobs, which will allow us to bound OPT's profit. This section of the proof differ greatly from the throughput case.

We begin by showing that if not too many processors are executing $\frac{v_i}{c}$-dense jobs then all such jobs must be currently processed under S.

Lemma 18. *Consider a job J_i and a time $t^* < D_i$. For any time step $t \in [r_i, r_i + t^*] \setminus I_i$ (that is not added to I_i by S), the total number of processors required by $\frac{v_i}{c}$-dense jobs in $J(t)$ must be more than $b(1-b)m$, i.e., $N(J(t), \frac{v_i}{c}, \infty) > b(1-b)m$.*

Proof. Because $t \in [r_i, r_i + t^*] \setminus I_i$ and $t^* < D_i$, we know that time step t is before D_i.

Since t is not added to I_i, it must be the case that for some density $v_j \in (\frac{v_i}{c}, v_i]$, the required condition is not true, i.e., $N(J(t) \cup \{J_i\}, v_j, cv_j) > bm$. Note that v_j must be in the range $(\frac{v_i}{c}, v_i]$. This is because without assigning J_i to time step t it is true that $N(J(t), v_j, cv_j) \le bm$ according to S, therefore J_i must have a density within the range of $[v_j, cv_j)$ in order to make impact.

By Lemma 11, we know that $n_i \le b^2 m$. Thus, we have

$$N(J(t), v_j, cv_j) = N(J(t) \cup \{J_i\}, v_j, cv_j) - n_i > bm - b^2 m = b(1-b)m$$

Therefore, we obtain $N(J(t), \frac{v_i}{c}, \infty) \ge N(J(t), v_j, cv_j) > b(1-b)m$. $\square$

Let O be the set of jobs completed by OPT. For each job $J_i \in O$, let d be the difference between J_i's completion time and arrival time under OPT; the profit of J_i under OPT is $p_i(d)$. According to the assumption in Theorem 3, we know that if $d \le x_i^*$, then $p_i(d) = p_i(x_i^*)$ for some $x_i^* \ge (\frac{W_i - L_i}{m} + L_i)(1+\epsilon)$. Therefore, we can assume that OPT assigns a relative deadline D_i^* to J_i, where $D_i^* = \max\{d, x_i^*\}$. Thus, OPT obtains a profit of $p_i(d) = p_i(D_i^*)$.

Lemma 19. *Consider a job J_i such that D_i assigned by scheduler S is larger than the deadline D_i^* assigned by* OPT, *i.e., $D_i > D_i^*$, the number of time steps during $[r_i, r_i + D_i^*)$ where scheduler S is actively executing $\frac{v_i}{c}$-dense jobs on at least $b(1-b)m$ cores is at least $\frac{\delta}{1+2\delta} D_i^*$.*

Proof. By definition of D_i^* and Lemma 14, we know that $D_i^* \ge x_i^*$.

Consider the number of time steps in time interval $[r_i, r_i + D_i^*]$ that are added to I_i, it must be less than $(1+\delta)\left(\frac{W_i - L_i}{n_i} + L_i\right) = (1+\delta)x_i$; otherwise, D_i^* would be a valid deadline under scheduler S with higher profit. Therefore, the number of time steps in $[r_i, r_i + D_i^*] \setminus I_i$ is more than $D_i^* - (1+\delta)x_i \ge D_i^* - \frac{1+\delta}{1+2\delta} x_i^* \ge D_i^* - \frac{1+\delta}{1+2\delta} D_i^* = \frac{\delta}{1+2\delta} D_i^*$.

By Lemma 18, we know that for each time step $t \in [r_i, r_i + D_i^*] \setminus I_i$, the total number of processors required by $\frac{v_i}{c}$-dense jobs in $J(t)$ must be more than $b(1-b)m$. Therefore, there must be at least $b(1-b)m$ cores executing $\frac{v_i}{c}$-dense jobs under scheduler S at time step t and the number of such steps is at least $\frac{\delta}{1+2\delta} D_i^*$. $\square$

Among the jobs in O, let O_1 be the set of jobs that the deadline D_i assigned by scheduler S is no larger than that assigned by OPT, i.e., $D_i \le D_i^* < \infty$. In other words, the obtained profit of these jobs under scheduler S is no less than that under OPT, i.e., $p_i(D_i) \ge p_i(D_i^*)$, since the profit function $p_i(t)$ is non-increasing. Let O_2 be the remaining jobs $O_2 = O \setminus O_1$. Let $\|X\|^*$ be the total profit that OPT obtains from jobs in X and $\|X\|$ be the total profit that S obtains from jobs in X. For jobs in O_1, we have $\|O_1\|^* \le \|O_1\|$.

For an arbitrary set of jobs $\mathcal{E}$ and any $v \ge 0$ let $T_O(v, \mathcal{E})$ denote the total work processed by the optimal schedule for the jobs in $\mathcal{E}$ that are v-dense. Let β_i denote the total number of time steps where S is actively processing job J_i. By definition, we have $\beta_i \le \frac{x_i}{1+\epsilon}$. We similarly let $T_S(v, \mathcal{E})$ be the summation of $\beta_i n_i$ over all jobs i in $\mathcal{E}$ that are v-dense. Note that this counts the total number of processor steps S executes jobs in $\mathcal{E}$ that are v-dense over all time.

Now we are ready to bound the time that OPT spends on jobs O_2 that scheduler S obtains less profit than OPT.

Lemma 20. *Consider a job J_i in O_2, the deadline D_i assigned by scheduler S is longer than deadline D_i^* assigned by* OPT. *For all $v > 0$, $T_O(v, O_2) \le \frac{2(1+2\delta)}{\delta b(1-b)} T_S(\frac{v}{c}, \mathcal{J})$.*

Proof. For any job $J_i \in O_2$, we denote the lifetime of J_i under OPT as the time interval $[r_i, r_i + D_i^*)$, where D_i^* is the deadline assigned by OPT. For any density $v > 0$, let l be the number of time steps of the union of the lifetimes of all jobs in $A(O_2, v, \infty)$. By definition, $T_O(v, O_2) \le lm$, since OPT can execute them on at most m processors.

Let $M \subseteq O_2$ be the minimum subset of O_2 that the union of the lifetimes of jobs in M covers the same time intervals of jobs in O_2. By the minimality of M, we know that at any time t, there are at most two jobs in M that cover time t. Therefore, we can further partition M into two sets M_1 and M_2, where for any two jobs in M_1 or any two jobs in M_2, their lifetimes do not overlap. By definition, either M_1 or M_2 has a union lifetime that is at least $l/2$ and we assume WLOG it is M_1.

Consider $J_i \in M_1$ and let k_i be the number of time steps during its lifetime $[r_i, r_i + D_i^*)$ where scheduler S is actively executing $\frac{v_i}{c}$-dense jobs on at least $b(1-b)m$ cores. By Lemma 19, we know $k \ge \frac{\delta}{1+2\delta} D_i^*$. Therefore, during $[r_i, r_i + D_i^*)$ the number of processor steps where S is processing $\frac{v_i}{c}$-dense jobs is at least $b(1-b)m\frac{\delta}{1+2\delta}D_i^*$.

Let $K = \sum_{M_1} k_i$, be the total number of processor steps where S is processing $\frac{v}{c}$-dense jobs (since $v_i \ge v$) during the intervals in M_1. Thus, by definition,

$$K \ge \frac{\delta b(1-b)}{1+2\delta} m \sum_{J_i \in M_1} D_i^* > \frac{\delta b(1-b)}{1+2\delta} m \times \frac{l}{2} \ge \frac{\delta b(1-b)}{2(1+2\delta)} T_O(v, O_2)$$

Clearly, by adding additional intervals that are not in M_1, we have $T_S(\frac{v}{c}, \mathcal{J}) \ge K > \frac{\delta b(1-b)}{2(1+2\delta)} T_O(v, O_2)$, which gives us the bound. ◻

Lemma 21.

$$\|O\|^* = \|O_1\|^* + \|O_2\|^* \le \left(1 + (1 + \frac{1+2\delta}{\epsilon - 2\delta})(1 + \frac{1}{\epsilon\delta})\frac{2(1+2\delta)}{\delta b(1-b)}\right) \|\mathcal{J}\|$$

Proof. First, by the definition of O_1 and O_2, we have $\|O\|^* = \|O_1\|^* + \|O_2\|^*$ and $\|O_1\|^* \le \|O_1\| \le \|\mathcal{J}\|$. Now it remains to bound $\|O_2\|$.

We have $T_O(v, O_2) \le \frac{2(1+2\delta)}{\delta b(1-b)} T_S(\frac{v}{c}, \mathcal{J})$ from Lemma 20 for all densities v. The remaining proof for the lemma is similar to that in Lemma 9, except for a different constant. Therefore, $\|O_2\|^* \le (1 + \frac{1+2\delta}{\epsilon - 2\delta}) c \frac{2(1+2\delta)}{\delta b(1-b)} \|\mathcal{J}\|$. Taking the summation of $\|O_1\|^* + \|O_2\|^*$ completes the proof. ◻

Finally we are ready to complete the proof, bounding the profit OPT obtains by the total profit the algorithm obtains for jobs it completed.

Lemma 22. $\|C^O\| \le \frac{1 + ac\frac{2(1+2\delta)}{\delta b(1-b)}}{\epsilon - \frac{1}{(c-1)\delta}} \|C\|$.

Proof. This is just by combination of Lemma 17 and Lemma 21. ◻

6 Conclusion

Scheduling jobs online to maximize throughput is a fundamental problem, yet there has been little study of this topic when jobs are parallelizable and represented as DAGs. We give the first non-trivial result showing that a scheduling algorithm is provably good for maximizing throughput. In addition, we extend the result and give an algorithm for the general profit scheduling problem with DAG jobs.

There are several directions for future work. First, we want to design and implement more practical schedulers that have similar theoretical performance but are work-conserving and require fewer preemptions. Second, in this paper we focus on semi-non-clairvoyant algorithms that do not have any knowledge of the internal structure of the DAG. This lets us to provide very general results. However, it is possible that by using the internal structure one could design algorithms with better performance for some special DAG structures. Finally, we are also interested in exploring whether fully non-clairvoyant algorithms can have comparable performance for throughput.

References

1. OpenMP. OpenMP Application Program Interface v4.0, July 2013. `http://http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf`.
2. Intel. Intel CilkPlus, Sep 2013. `https://www.cilkplus.org/`.
3. James Reinders. *Intel threading building blocks: outfitting C++ for multi-core processor parallelism*. O'Reilly Media, 2010.
4. Colin Campbell and Ade Miller. *A Parallel Programming with Microsoft Visual C++: Design Patterns for Decomposition and Coordination on Multicore Architectures*. Microsoft Press, 2011.
5. Sanjoy K. Baruah, Gilad Koren, Decao Mao, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, Dennis Shasha, and Fuxing Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4(2):125–144, 1992.
6. Sanjoy K. Baruah, Gilad Koren, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, and Dennis Shasha. On-line scheduling in the presence of overload. In *Symposium on Foundations of Computer Science*, pages 100–110, 1991.
7. Gilad Koren and Dennis Shasha. Dover: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM J. Comput.*, 24(2):318–339, 1995.
8. Gerhard J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theor. Comput. Sci.*, 130(1):5–16, 1994.
9. Bala Kalyanasundaram and Kirk Pruhs. Fault-tolerant real-time scheduling. *Algorithmica*, 28(1):125–144, 2000.
10. Gilad Koren and Dennis Shasha. MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling. *Theor. Comput. Sci.*, 128(1&2):75–97, 1994.
11. Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
12. Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.
13. Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Competitive algorithms for due date scheduling. *Algorithmica*, 59(4):569–582, 2011.
14. Kirk Pruhs and Clifford Stein. How to schedule when you have to buy your energy. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, Barcelona, Spain, September 1-3, 2010. Proceedings*, pages 352–365, 2010.
15. Sungjin Im and Benjamin Moseley. General profit scheduling and the power of migration on heterogeneous machines. In *Symposium on Parallelism in Algorithms and Architectures*, 2016.
16. Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Efficient online scheduling for deadline-sensitive jobs: extended abstract. In *25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13*, pages 305–314, 2013.
17. Abusayeed Saifullah, David Ferry, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher D. Gill. Parallel real-time scheduling of dags. *IEEE Trans. Parallel Distrib. Syst.*, 25(12):3242–3252, 2014.
18. Jing Li, Jian-Jia Chen, Kunal Agrawal, Chenyang Lu, Christopher D. Gill, and Abusayeed Saifullah. Analysis of federated and global scheduling for parallel real-time tasks. In *ECRTS 2014*, pages 85–96, 2014.
19. Kunal Agrawal, Yuxiong He, Wen Jing Hsu, and Charles E. Leiserson. Adaptive task scheduling with parallelism feedback. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2006.
20. Kunal Agrawal, Yuxiong He, and Charles E. Leiserson. Adaptive work stealing with parallelism feedback. In *Proceedings of the Annual ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, March 2007.
21. Yuxiong He, Wen-Jing Hsu, and Charles E. Leiserson. Provably efficient online non-clairvoyant adaptive scheduling. In *IPDPS*, 2007.
22. Lin Ma, R.D. Chamberlain, and K. Agrawal. Performance modeling for highly-threaded many-core GPUs. In *Proc. of Int'l Conf. on Application-specific Systems, Architectures and Processors (ASAP)*, pages 84–91, June 2014.
23. Kunal Agrawal, Jing Li, Kefu Lu, and Benjamin Moseley. Scheduling parallel DAG jobs online to minimize average flow time. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 176–189, 2016.
24. Julien Robert and Nicolas Schabanel. Non-clairvoyant scheduling with precedence constraints. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 491–500, 2008.
25. Sanjoy Baruah. Improved multiprocessor global schedulability analysis of sporadic DAG task systems. In *26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, July 8-11, 2014*, pages 97–105, 2014.
26. Sanjoy Baruah. Federated scheduling of sporadic DAG task systems. In *2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2015, Hyderabad, India, May 25-29, 2015*, pages 179–186, 2015.
27. Sanjoy Baruah. The federated scheduling of systems of conditional sporadic DAG tasks. In *2015 International Conference on Embedded Software, EMSOFT 2015, Amsterdam, Netherlands, October 4-9, 2015*, pages 1–10, 2015.
28. Sanjoy Baruah, Vincenzo Bonifaci, and Alberto Marchetti-Spaccamela. The global EDF scheduling of systems of conditional sporadic DAG tasks. In *27th Euromicro Conference on Real-Time Systems, ECRTS 2015*, pages 222–231, 2015.
29. Sanjoy Baruah. The federated scheduling of constrained-deadline sporadic DAG task systems. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015*, pages 1323–1328, 2015.
30. Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. Analysis of global edf for parallel tasks. In *ECRTS*, 2013.
31. Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Sebastian Stiller, and Andreas Wiese. Feasibility analysis in the sporadic dag task model. In *ECRTS*, 2013.
32. Ola Svensson. Conditional hardness of precedence constrained scheduling on identical machines. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 745–754, 2010.