Online Algorithms 18.1.2012

Lecture Notes 12: Scheduling - Cont.

Professor: Yossi Azar *Scribe:Inna Kalp*

1 Introduction

In this Lecture we discuss 2 scheduling models. We review the "scheduling over time model" that was introduced in the previous lecture and prove that SRPT algorithm is optimal with respect to this model. We also discuss another scheduling model where each job also has size, release time, deadline and value. In this model we'll present:

- EDF for unit size and unit value jobs
- EDF for jobs of different size
- Max value first for unit size jobs with arbitrary value

2 Reminder

2.1 Interval Scheduling

In the previous lecture we completed analyzing the Interval Scheduling Problem (assuming the interval is of length T).

Table 1: val =1

	deterministic alg.	random alg.
no preemption	T	logT
with preemption	logT	Const

Table 2: val =length

	deterministic alg.	random alg.
no preemption	T	logT
with preemption	$\sqrt{5}+2$	Const

2.2 Scheduling Over Time

The Model:

- One (or more) processors
- Jobs are released over time. A job i has release time r_i and size w_i.
- No more than one job is executing in any given time.
- Preemption is allowed. Preemption in this model means that a processor can stop the execution of a job and continue executing it later.
- Each job has Completion Time c_i, and Flow Time $f_i = c_i - r_i$.

We consider 2 possible goal functions:

1. Minimize maxc_i: we saw a reduction from release time to batch - we collect several jobs and then process the batch, and so on. The reduction is 2 competitive. We also showed that greedy algorithm is 2 competitive for identical machines.

2. A more reasonable goal function is to minimize $\sum f_i$, meaning the total (or average) processing time. If all jobs arrive in time 0, we proved that SPT (=Shortest Process Time) algorithm is optimal.

3 Scheduling over time - General Case

Consider the general case where job i arrives at time r_i.
Theorem: SPRT (Shortest Remaining Processing Time) is optimal in the general case.
Proof: Assume by contradiction that the optimal schedule is not SRPT. Look at the first time t_0 that a job A should run according to SRPT, but instead, job B is executing in the optimal schedule. The sum of all the segments in which A or B execute after time t_0 equals to the sum of their remaining sizes from time t_0. Define t_2 as the time in which the first job from {A,B} ends its execution. Also define t_1 as the time in which the second job ends. If we'll change the execution order only in the segments in which jobs A or B execute after time t_0, the end of execution of all the other jobs will remain unchanged. Notice that our goal is to minimize $\sum f_i = \sum c_i - \underbrace{\sum r_i}_{constant}$, and we can alternatively minimize $\sum c_i$. (The two sums are equivalent for exact solution. This would not work if we were attempting to find an approximation algorithm). Therefore, the optimal schedule brings to a minimum $C_A + C_B$. In every schedule t_1= max$\{C_A, C_B\}$. If we will execute job A "Continuously" (only in the slots that are originally of A or B) , time t_2 will decrease, and by definition $C_A + C_B$ will decrease. We decreased $\sum C_i$ in contradiction to the assumption that the initial schedule is optimal.

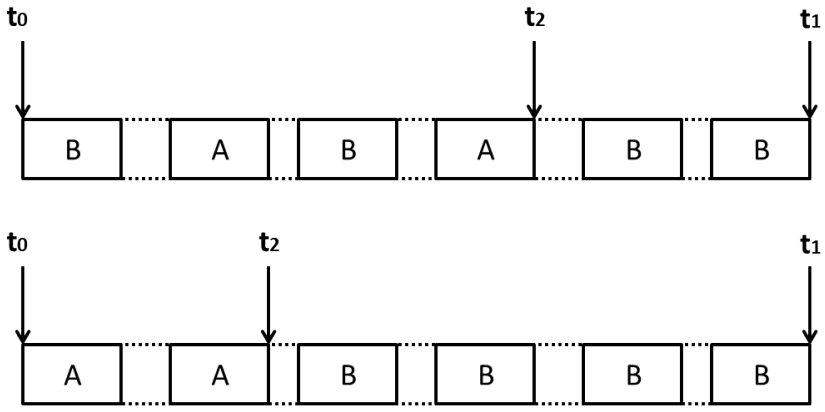

Figure 1: Upper figure - The slots of jobs A and B in the assumed optimal schedule. Lower figure: the schedule after the reordering of the execution in the slots of A and B.

4 Scheduling with deadlines

The Model:

- One (or more) processor(s)
- job i comes with $\left\{ \underbrace{p_i}_{size}, \underbrace{r_i}_{release-time}, \underbrace{d_i}_{deadline}, \underbrace{b_i}_{benefit} \right\}$
- At most one job is executing in any given time.
- Preemption is allowed.

The Goal is to maximize $\sum_{i \in A} b_i$ where A is the set of all jobs that ended before their deadline.

Definition :EDF (= Earliest Deadline First) algorithm : at any given time t, among the "live jobs" (release time before t, deadline after t and its execution hasn't ended yet), execute the job with the minimal feasible deadline (feasible means that if we process this job continuously it will be completed before its deadline).

Note that EDF is idle only when there are no available feasible jobs

4.1 EDF for unit size jobs

For the unit size jobs, i.e., $\forall i : p_i = 1$ we assume that all release time and deadlines are integers (this is called the slotted case).

Theorem 1: If there exists a schedule that can execute all jobs, then EDF also executes all jobs.

Proof: Assume there is a schedule that can execute all jobs. Let's take two jobs that are executing in the opposite order accoring to EDF, and switch them. The new schedule is still feasible (since both jobs end before their deadlines). Repeat this until the new schedule is EDF.

4.2 EDF for unit size and unit value jobs

In case the value of all unit jobs is equal (can be normalized to be 1), i.e., $\forall i : b_i = p_i = 1$, we have the following theorem which does NOT assume that there exists a schedule that can execute all jobs. Note that we are still in the slotted case.

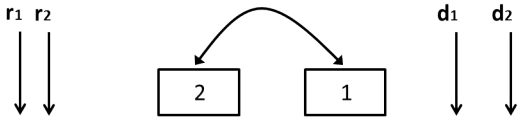

Figure 2: turning a schedule that executes all jobs to EDF schedule.

Theorem 2: EFD is optimal for unit size and unit value jobs.

Proof: Consider the optimal schedule - OPT. WLOG assume that the jobs that are performed in OPT are ordered by EDF (otherwise we can reorder them using theorem 1). Consider the first job that was rejected "unrightfully by EDF" at time t, meaning OPT is executing job B at time t, but by EDF we should have executed job A. Therefore we can

deduce that job A was rejected by OPT (since OPT is ordered by EDF). We will switch jobs A and B (run A instead of B) and the schedule will remain optimal, because the number of executed jobs is unchanged and the schedule is feasible. Repeat switching jobs. Since with every iteration, the prefix of the schedule that equal to EDF increases, we get EDF schedule.

4.3 EDF for jobs of different size

Note: In the general case, EDF can be preemptive. For example, assume that during an execution of a job 1with deadline d_1, job 2 arrives with deadline $d_2 < d_1$.

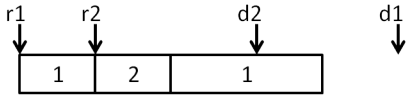

figure 3: example - EDF can be preemptive

Theorem 3: In the general case (jobs of different sizes), if there exists a schedule that executes all jobs, then EDF executes all jobs.

Proof: Consider the first time such a schedule is not EDF. We have a job A that should execute according to EDF, but job B executes instead ($d_B > d_A$). Let's reorder the slots where jobs A and B are executing. We'll execute job A in those slots until it finishes, and then execute job B. In any reordering of these slots, B will finish before d_B, and in our reordering, A will finish before its original completion time and therefore before d_A, so the schedule will remain feasible. We will repeat the reordering until we get EDF schedule, since with every iteration, the prefix of the schedule that equal to EDF increases.

Note: If there is no schedule that can execute all jobs EDF can be very bad, even when all jobs have unit value.

Examples:

1. Define n$\in \mathbb{N}$:1 x (n-1,0,n-1,1) , n x (1,0,n,1): EDF will perform only 2 jobs - one job from the first group,and another one from the second group. OPT will perform n jobs from group 2. Ratio of $\frac{n}{2} = \Omega(n)$.

4.4 Unit size jobs with arbitrary value

Even for unit size jobs EDF can be very bad

1. Define M , n$\in \mathbb{N}$, (n-1) x (1,0,n-1,1) , n x (1,0,n, M): EDF will execute the n-1 jobs from the first group and one job from the second group. EDF = n-1+M. OPT will perform n jobs from the second group. OPT = nM. Ratio of $\frac{nM}{n-1+M} = \Omega(n)$ for large enough M.

Definition: MVF = Max Value First algorithm executes the job with the maximum value among the "living jobs".

Theorem: MVF is 2 competitive, when p_i is 1 and ,b_i is arbitrary.

Proof: Let's define RA =the group of jobs that OPT executed and MVF rejected. We will show that

$$(*)\sum_{i\in RA} b_i \leq MVF = \sum_{i\in A} b_i.$$

After we'll prove (*) , we can conclude :

$$OPT(\sigma) \leq \sum_{i\in A} b_i + \sum_{i\in RA} b_i \leq 2\sum_{i\in A} b_i = 2MVF(\sigma)$$

and the competitve ratio follows. We still have to prove (*): Let's take $i \in RA$. Suppose OPT executes at time t. There are several reasons why MVF doesn't execute i at time A:

1. MVF already executed it - can't be since $i \in RA$.
2. i is not alive (t$<r_i$or t$>d_i$) - can't be since OPT executes i at time t.
3. MVF executes in time t a job with an higher value.

since option 3 is the only viable option, we found a one-to-one function between RA to A, such that every job in RA corresponds to a job in A with a higher value. Therefore, $\sum_{i\in RA} b_i = \sum_{i\in A} b_i \leq MVF$ follows.
Note: The competitive ratio is tight.
Example: (1,0,1,1) , (1,0,2,1+ε): MVF will execute the second job. OPT Will execute 2 jobs: it will execute the first job and then the second one. MVF = 1+ε. OPT = 1+1+ε. The ratio is 2.