IEOR 8100: Scheduling Algorithms
Lecture 12

Instructor: Clifford Stein
Scribe: Benjamin Kuykendall

October 20, 2016

In this lecture, we consider some problems in resource augmented scheduling, reviewing a key lemma, then considering applying simple algorithms to achieve s-speed solutions to hard problems. We define m-machine augmented schedules. We motivate and define energy-aware scheduling, considering objective functions in both time and energy. Then we solve $1|r_j, d_j, \text{pmtn}| \int s^\alpha dt$.

Resource Augmentation

Let us review our main lemma on s-speed augmented schedules.

Lemma. *For any input I, time t, $m \geq 1$, and $1 \leq \beta \leq 2 - 1/m$, let*

$$\alpha = \frac{2 - 1/m}{\beta}.$$

Consider two busy schedules on I: A using m speed α machines, and A' using m speed 1 machines. Then

$$A(I, \beta t) \geq A'(I, t)$$

where $S(I, t)$ denotes the total amount of work done by schedule S on input I by time t.

One interpretation of this lemma is to take A' as the optimal schedule, executed on speed 1 machines. Then A can be any busy schedule, and as long as it runs on speed α machines, it will "keep up" with OPT, completing at least as much work in time βt as OPT does in time t.

Problem $P|r_j, \text{pmtn}|d_j$**.** *parallel machines; release dates and preemption; meet all deadlines.*

Theorem. *Earliest Deadline First is a 2-speed algorithm for $P|r_j, pmtn|d_j$.*

Another interesting solution to this problem uses the concept of laxity. Roughly, a job's laxity is how long can you wait before you must start the job. In our notation, at time t a job j has laxity $\ell_j := (d_j - t) - (p_j - x_j)$ where x_j is the amount of work done on the job so far. This gives the following algorithm.

Theorem. *Least Laxity First is a 2-speed algorithm for $P|r_j, pmtn|d_j$.*

Of course, with preemption, both algorithms require a bit of clarification. Like we have done previously, we must re-evaluate which job to run at some set of times including the release dates, completion times, and at some periodic interval while executing a job. Though this might need more consideration for an actual implementation of the algorithm, what we mean is clear enough.

Although we proved neither theorem, they are proved in [1]. However, finding a s-speed algorithm for $s < 2$ is an interesting open problem.

In addition to augmenting the speed of the available machines, we can increase the number of machines. An m'-machine algorithm is given m' machines instead of the m specified in the instance. In the preemptive case, s-speed algorithms are strictly stronger than (sm)-machine algorithms. Given a (sm)-machine algorithm, we can always simulate it on m machines running at speed s by splitting each time interval to emulate time-sharing. We can show the (sm)-machine is weaker by example. Consider one large job: on a speed s machine, it is done at p_j/s, but on any number of speed 1 machines it takes time p_j. The difference between these two settings motivates us to look for m'-machine solutions to the scheduling problem above.

Theorem. *There are $O(m \log(np_{max}))$-machine and $O(m \log m)$-machine algorithms for $P|r_j, pmtn|d_j$.*

These algorithms are given in [1] and [2] respectively.

Problem $1|r_j, \text{pmtn}|\sum F_j$**.** *1 machine; release dates and preemption; minimize sum of flow times*

Recall we have a $log(n/m)$-approximation for the problem with m machines.

Theorem. *Shortest remaining processing time (SRPT) is a 2-speed algorithm for $1|r_j, pmtn|\sum F_j$.*

This result follows from the lemma. We can show that SRPT_2 "keeps up" with OPT by the lemma, so SRPT_2 completes i jobs no later than OPT completes i jobs. Though we skip the formal proof, we can show optimality with an inductive argument, also in [1].

Energy-Aware Scheduling

The problem of minimizing energy use is well-motivated. In many cases, a machine can run at a range of speeds, and faster speeds require more energy. Finding a schedule that optimizes for an objective that keeps time and energy low does have real life applications. Taking the example of a data center, reducing energy use reduces costs and environmental impact significantly.

We tweak our model slightly to include speed and energy. Each job now has an amount of work w_j associated with it (instead of the t_j in previous problems). At each time, a machine is running at some speed $s(t) \in [0, \infty)$. To complete a job worked on in times T, we must have $\int_T s(t)dt = w_j$. Define power a function of speed. Usually we set $p(s) = s^\alpha$ for some α around 2 or 3. Then our energy use is $E = \int p(s(t))dt$.

This model matches empirical results fairly well. However, we could conciser more general situations. For example, maybe taking $s \in \{s_1, \ldots, s_n\}$ with some arbitrary $p(s)$ would better model cpus, which general have a finite set of speeds. But the model in the paragraph above is both fairly realistic and easy to analyze.

Now consider some possible objectives in both time and energy. Taking a traditional scheduling objective T (e.g. $C_{max}, \sum C_j, \sum F_j, \dots$), there are different ways to combine it with an energy goal.

- Fix some maximum E and optimize for T
- Fix some T and minimize E
- Optimize some function $f(E, T)$

Although these are all reasonable objectives, deadlines also give an even more natural problem.

Problem $1|r_j, d_j, \text{pmtn}| \int s^\alpha dt$**.** *1 machine; release and due dates and preemption; minimize energy*

Supposing $\alpha \geq 1$, we have an efficient algorithm given in [3]. First some preliminaries.

Fact 1. *Let T be a set of times in which work w is completed. Then a function $s(t)$ minimizes the energy $\int_T s(t)^\alpha dt$ when $s(t) = w/|T|$ for all $t \in T$.*

Proof. This fact follows directly from the convexity of x^α. By Jensen's inequality, we have

$$\left(\frac{1}{|T|}\int_T s(t)dt\right)^\alpha \leq \frac{1}{|T|}\int_T s^\alpha(t)dt.$$

The left hand side evaluates to $(w/|T|)^\alpha$. Observe that choosing $s(t) = w/|T|$ achieves the bound:

$$\frac{1}{|T|}\int_T \left(\frac{w}{|T|}\right)^\alpha dt = \left(\frac{w}{|T|}\right)^\alpha.$$

□

Now for any interval $T = [t, t']$, define the set of jobs that must run in T

$$J(T) = \{j \mid r_j, d_j \in T\}.$$

We say this interval has an intensity

$$g(T) = \frac{1}{z - z'}\sum_J w_j.$$

Fact 2. *In any interval T, for jobs $J(T)$ to complete, we must have $s(t) \geq g(T)$ for all $t \in T$.*

With these two facts in mind, we sketch an algorithm:

1. Find the T of maximal $g(T)$.
2. Schedule the jobs $J(T)$ in T by preemptive earliest deadline first at speed $s = g(T)$.
3. Remove the interval T and jobs $J(T)$ from the input and repeat.

We claim this is an efficient algorithm: clearly, the T of maximal intensity will begin on some r_j and end on some d_j, so there are at most n^2 intervals to consider. As each round removes at least 1 job, there are at most n rounds.

We exclude a formal proof of correctness. The following example demonstrates the algorithm.

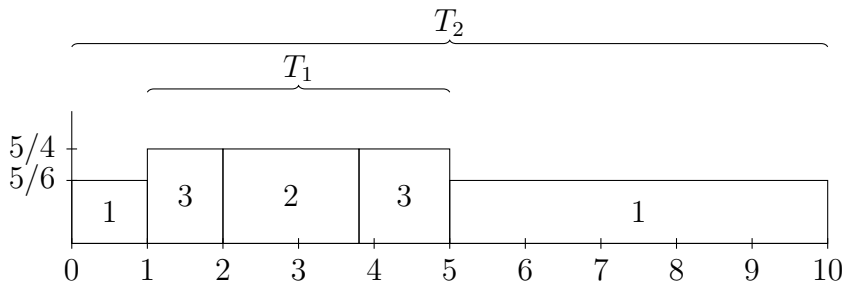

Figure 1: The optimal schedule. Each job runs in the numbered rectangle. The horizontal axis plots time, the vertical machine speed. Intervals of maximal intensity are shown in braces above.

j	r_j	d_j	w_j
1	0	10	5
2	2	3	1
3	1	5	4

In the first round, we have intensities

$$g([0,10]) = \frac{w_1 + w_2 + w_3}{10 - 0} = 1 \qquad g([1,5]) = \frac{w_2 + w_3}{5 - 1} = \frac{5}{4} \qquad g([2,3]) = \frac{w_2}{3 - 2} = 1$$

Thus we select $T_1 = [1,5]$. Let $s = 5/4$, and run job 3 on $[1,2]$ and $[3.8,5]$ and job 2 on $[2,3.8]$.

This leaves only one interval for the second round

$$g([0,1] \cup [5,10]) = \frac{w_1}{(10-5) + (1-0)} = \frac{5}{6}$$

Thus we select $T_2 = [0,1] \cup [5,10]$. Let $s = 5/6$, and run job 1 on $[0,1]$ and $[5,10]$.

Alternatively, we can look at this problem as a convex optimization. Although the greedy algorithm above should be faster than minimizing numerically, the program is still a useful tool for analysis of this and other problems.

Consider only the times $t_1, \ldots, t_k$ in $\{r_j\} \cup \{d_j\}$ numbered in increasing order.
The we denote $I_i = [t_i, t_{i+1}]$ and $J(i)$ the jobs that can execute in I_i (i.e. $r_j \leq t_i \leq t_{i+1} \leq d_j$).

Using this notation, we define the following convex program:

Variables: The w_{ij} are the work done on job j in interval i.

Objective: Minimize the energy $\sum_i \left(\frac{\sum_j w_{ij}}{t_{i+1} - t_i} \right)^{\alpha} (t_{i+1} - t_i)$.

Constraints:

The w_{ij} are all non-negative, but are 0 when j cannot run in i.

For each job $\sum_i w_{ij} \geq w_j$ gives that it must complete.

It is easy to verify this is a convex optimization problem: the objective is convex in the variables, and the feasible region is given by an intersection of half-spaces.

References

[1] Phillips, Stein, Torng, Wein. "Optimal Time-Critical Scheduling Via Resource Augmentation" STOC 97. http://dl.acm.org/citation.cfm?id=258570

[2] Chen, Megow, Schewior. "An O(log m)-Competative Algorithm for Online Machine Minimization" SODA 2016. https://arxiv.org/pdf/1506.05721v1.pdf (pre-print).

[3] Yao, Demers, Shenker. "A Scheduling Model for Reduced CPU Energy" FOCS 95. https://www.computer.org/csdl/proceedings/focs/1995/7183/00/71830374.pdf