IEOR E8100: Scheduling Algorithms

Lecture 13

Yunjie Sun

In the last lecture, we talked about the problem of minimize energy while meeting deadlines. In today's lecture, we are going to discuss another type of problem related to energy, which is to minimize flow time (min $\sum F_j$) subject to an energy budget (E).

Let's first consider a warm-up, simple problem with unit work jobs, one machine, $p(s) = s^\alpha$, and $r_i = 0$. Let

- x_i = time spent on job i,
- C_i = completion time of job i,
- F_i = flow time of job i,
- E = energy budget.

Furthermore, we can calculate the energy spent on job i in the following way:

$$\text{energy on i} = s_i^\alpha x_i = (\frac{1}{x_i})^\alpha x_i = x_i^{1-\alpha}. \tag{1}$$

Therefore, we can formulate the problem as:

$$\min \sum C_i \tag{2}$$

$$\text{s.t. } C_i = C_{i-1} + x_i \tag{3}$$

$$\sum_i x_i^{1-\alpha} \leq E. \tag{4}$$

The above formulation is a convex programming program so it can be solved in polynomial time.

Suppose $r_i \neq 0$, we can modify the above formulation to:

$$\min \sum(C_i - r_i) \tag{5}$$

$$\text{s.t. } C_i \geq C_{i-1} + x_i \tag{6}$$

$$C_i \geq r_i + x_i \tag{7}$$

$$\sum_i x_i^{1-\alpha} \leq E. \tag{8}$$

If we only have two jobs, idealy, we will use more energy on the first job. As discussed in section 3.3 in Pruhs et al. [2008], the first job is more important than the second one in this case which leads to spending more energy on the first job in the optimal solution. So 1) speed decreases over time in optimal solution; 2) speed is proportional to the number of released but unfinished jobs.

Following is the identity in scheduling which we will use later in analysis:

$$\sum F_j = \int_t (\text{Number of released but unfinished jobs } (t))dt \tag{9}$$

The following figure describes the correctness of (9). The total area is Figure 1 is the sum of flow

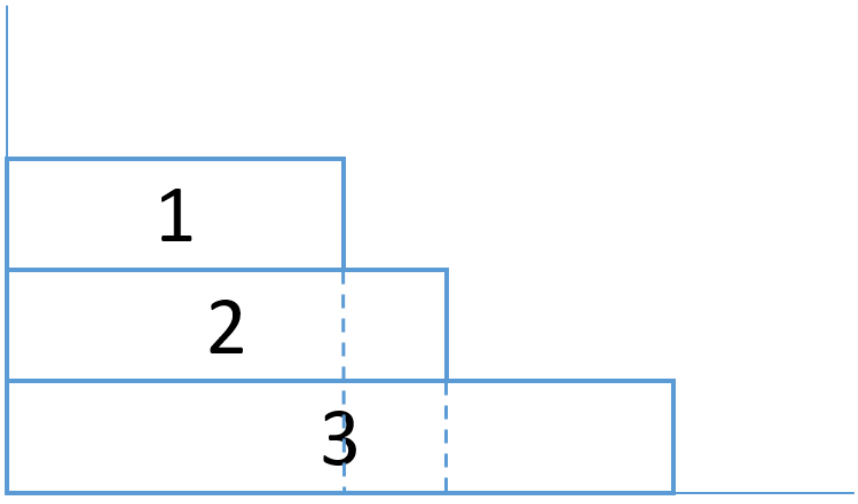

Figure 1: Example

time of all jobs. The left hand side of (9) calculates the summation in horizontal while the right hand side calculates the same area by adding it vertically (corresponding to the dotted lines).

The above formulations and analysis are for the case of offline. However, we want to do something online. For the online problem, we have the same assumptions as before but with release time constraints, r_j. Moreover, in the online version of the problem, we have jobs arrive over time and we don't know about a job until it arrive.

We now argue that there is no good online algorithm for minimizing flow time subject to an energy budget. Suppose at time 0, 3 jobs arrive, then we have to use at least $\frac{E}{c}$ energy, where c is some constant, because those might be the only jobs that arrive. If at time t_1, 3 more jobs arrive, again we have to use at least some fraction of the remaining energy. Suppose at some time t, we finished L jobs with ϵ energy left, then if L^3 number of jobs are released, then clearly we are in trouble. Therefore, there is no "good" online algorithm for the problem of minimizing $\sum F_j$.

We now look at the problem of minimizing $\sum F_j + E$, more specifically, $1|r_j, \text{pmtn}| \sum F_j + E$. There exists a constant competitive online algorithm for this problem.

For this problem, we allow more general model of $p(s)$. We need $p(s)$ to have the following properties:

1. $p(s)$ is continuous and differentiable $\forall s \in (0, \infty)$
2. $p(0) = 0$
3. $p(s)$ is strictly increasing in s
4. $p(s)$ is strictly convex
5. $p(\infty)$ is unbounded

The following figure shows how it could be between our algorithm (ALG) and the optimal (OPT). There are different ways in analyzing online algorithms,

- $\forall t,\ A(t) \leq c \cdot OPT(t)$
- $\forall t,\ \int_{s=0}^{t} A(s)ds \leq c \cdot \int_{s=0}^{t} OPT(s)ds$

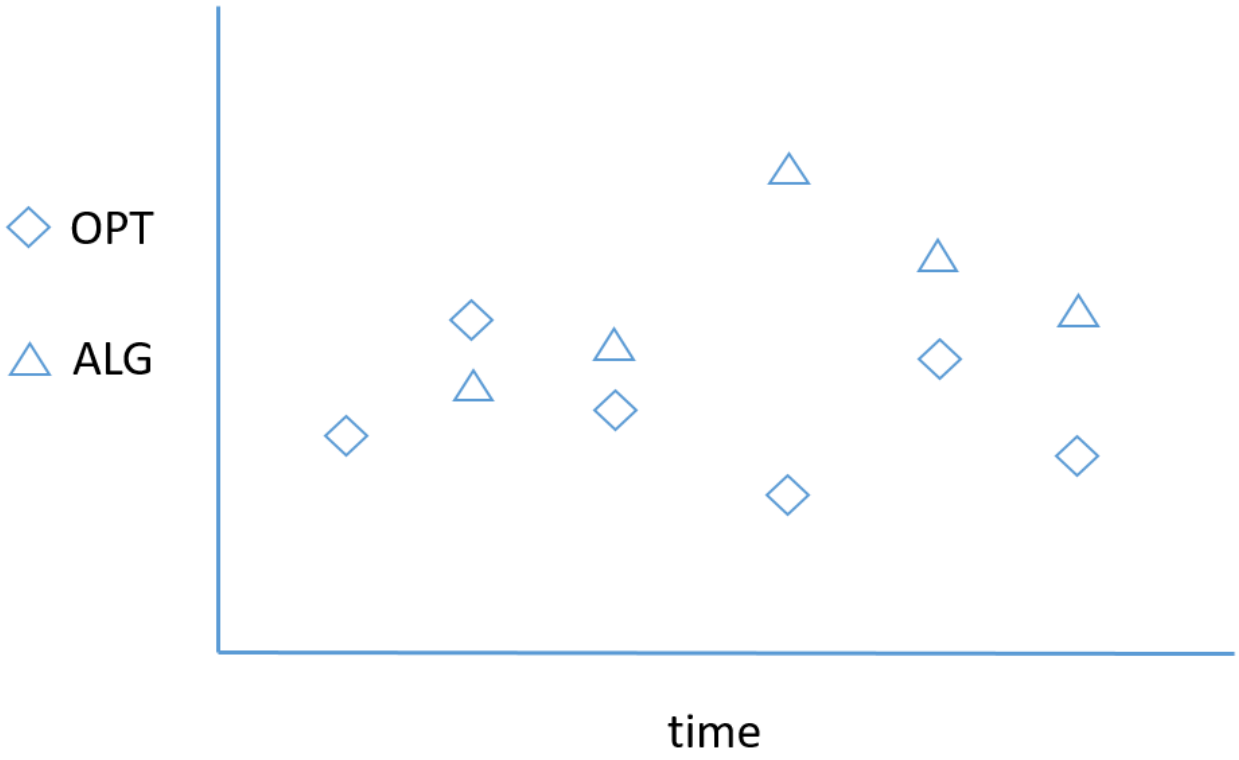

Figure 2: ALG vs. OPT

Here, we will use potential function to do the analysis instead.

Let G be the objective, i.e., $G_A = \sum F_j + E = F_A + E_A$, where $E_A = \int p(s_A(t))dt$, and $F_A = \int n_A(t)dt$. In the previous equation, $n_A(t)$ represents the number of released but unfinished jobs at time t by algorithm A. Therefore, $G_A(t)$ represents the increase in objective G with algorithm A at time t, and could be written as $G_A(t) = p(s_A(t)) + n_A(t)$.

Now, we define the term *Amortized local competitiveness* (A.L.C) in the following sense: Given a $\phi(t)$ s.t.: (1) $\phi(0) = 0$, $\phi(T) \geq 0$; (2) $G_A(t) + \frac{d\phi(t)}{dt} \leq c \cdot G_{OPT}(t)$. From A.L.C, we can get the competitive argument by the following,

$$\int_0^T G_A(t) + \int_0^T \frac{d\phi(t)}{t} \leq \int_0^T c \cdot G_{OPT}(t) \tag{10}$$

$$\Rightarrow G_A + \phi(T) - \phi(0) \leq c \cdot G_{OPT} \tag{11}$$

$$\Rightarrow G_A \leq c \cdot G_{OPT} \text{ by property of } \phi(\cdot) \tag{12}$$

Therefore, our plan is to first choose an algorithm (choose jobs and speed), then choose ϕ, and last show A.L.C. holds.

Our idea here to to set two terms, E_A, and F_A equal. Suppose $p(s) \approx s^2$, let $p(s) = n_A(t)$ gives us $s^2 = n_A(t)$, results in $s = \sqrt{n_A(t)}$. This is a 4-approx.

In general,

$$p(s_A(t)) = \begin{cases} p^{-1}(n_A(t)+1) & \text{if } n_A(t) \geq 1, \\ 0 & \text{if } n_A(t) = 0. \end{cases} \tag{13}$$

In this case, $p(s_A(t)) = p(p^{-1}(n_A(t)+1)) = n_A(t)+1$.

In the algorithm, we will use shortest remaining process time (SRPT) to decide which job to run. In our analysis, let's assume $p(s) = s^\alpha$. Let $\phi(t) = \max\{0, n_A(t) - n_{OPT}(t)\}$, and $n_A^{\geq q}(t)$ represent the number of released but unfinished jobs at time t in algorithm A of remaining size $\geq q$. So, we have $n^{\geq q}(t) = \max\{0, n_A^{\geq q}(t) - n_{OPT}^{\geq q}(t)\}$. Also, we define $\phi(t) = 3\int_0^\infty f(n^{\geq q}(t))dq$, where $f(0) = 0$, and $f(i) - f(i-1) = p'(p^{-1}(i))$.

Now, we discuss the intuition to get $\phi(\cdot)$. We need the following equation to hold,

$$G_A(t) + \frac{d(\phi(t))}{dt} \leq c \cdot G_{OPT}(t) \tag{14}$$

which is equivalent to

$$p_A(t) + n_A(t) + \frac{d\phi}{dt} \leq c \cdot (p_{OPT}(t) + n_{OPT}) \tag{15}$$

By choosing s_A which makes $F_A \approx E_A$ and setting $c = 2$, we get

$$2p_A(t) + \frac{d\phi}{dt} \leq 2(p_{OPT}(t) + n_{OPT}). \tag{16}$$

We want ϕ to be a function of $N = n_A - n_{OPT}$. Notice that upon arrival, A and OPT do the same thing, so the worst case is $n_{OPT} = 0$, $n_A = N$. Under this case, we can write $\frac{d\phi}{dt} = \frac{d\phi}{dN} \cdot \frac{dN}{dt}$. Intuitively, $\frac{dN}{dt} = s_{OPT} - s_A$. Plug this into (16) gives us

$$2p_A(t) + \frac{d\phi}{dN}(s_{OPT} - s_A) \leq 2p_{OPT} \text{ because } n_{OPT}(t) = 0 \tag{17}$$

From (17), we can express $\frac{d\phi}{dN}$ as

$$\frac{d\phi}{dN} = 2\left(\frac{p_{OPT} - p_A}{s_{OPT} - s_A}\right) \approx 2\frac{dp(s_A)}{ds} \approx \frac{2dp(p^{-1}(N))ds}{ds}, \tag{18}$$

which results in

$$d\phi \approx 2 \cdot p'(p^{-1}(N))dN \tag{19}$$

For verifying this ϕ function satisfies A.L.S, we refer to the proof in Section 3 in Bansal et al. [2009]

References

Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms (TALG)*, 4(3):38, 2008.

Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. In *Proceedings of the twentieth annual ACM-SIAM symposium on discrete algorithms*, pages 693–701. Society for Industrial and Applied Mathematics, 2009.