IEOR 8100: Scheduling Lecture Guest

Optimal Online Preemptive Scheduling

Lecturer: Jir Sgall *Scribe: Michael Hamilton*

1 Introduction

In this lecture we'll study online preemptive scheduling on m machines of arbitrary speeds. We'll first review the problem and it's associated literature, then give an optimal deterministic algorithm known as "RatioStretch". RatioStretch uses the optimal competitive ratio $R > 1$ as an input, and achieves R whenever such an R is obtainable. The exact optimal R maybe computed via a linear program, however it varies based on the problem input parameters and in general is unknown (as an absolute constant over all possible machines).

2 Warm-Up

We'll warm up by studying the cow path problem. An extremely simple problem with a surprisingly deep solution.

2.1 Cow Path Problem

1. **Setup:** You're a cow and you start at the origin 0, of an infinite length line. As in figure 1 for some unknown $n > 0$, a tasty treat is buried at either n or $-n$ on the line. As a cow, your keen sense of smell can detect the treat but only when you walk directly over it.

2. **Objective 1:** Give deterministic algorithm $\mathcal{A}$ that minimizes worst case total travel time to find the treat.

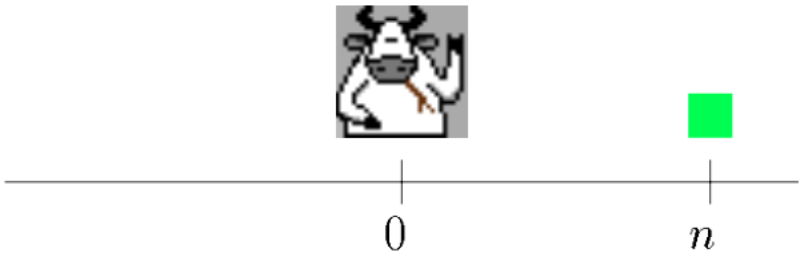

Figure 1: Credit: Dror Rawitz, circa 2002

If n was known to us, it's clear the best algorithm would be to walk n steps to the left and, if the treat is not found, 2n steps back to the right. Thus the worst case when n is known is $3n$ steps. For n unknown we'd like an algorithm with worst case distance traveled to be something like Cn where C is a constant.

A natural algorithm to consider is one where the cow "sweeps" out the line in stages. At stage i for all i odd, the cow travels distance $d(i)$ to the left and, if it doesn't find the treat, returns to the origin. At stage $i+1$, the cow travels $d(i+1)$ to the right, and if it does not find the treat, returns to the origin. As long as $d(\cdot)$ is an increasing function this strategy will eventually find the treat. The question is what is the best function $d(\cdot)$?

Theorem 1 *The algorithm that sweeps out the line according to* $\mathcal{A} : d(i) = 2^i$ *has worst case walking distance* $9n$ *for the cow path problem. Further the factor 9 is tight and optimal for any* **deterministic** *strategy.*

Proof Let $j \in \mathbb{N}$ be s.t. $2^j < n \leq 2^{j+1}$, then no matter what the algorithm takes first $\sum_{i=1}^{j} 2 * 2^i$ steps since at each stage up to j we can't find the treat. Now suppose for the worst case, at stage j+1 we walk in the wrong direction adding distance $2 * 2^{j+1}$. Finally we walk distance n the opposite way to the treat. Adding it all together:

$$\text{Distance } \mathcal{A} = \sum_{i=1}^{j+1} 2 * 2^i + n = 2(2^{j+2} - 1) + n \leq 8n - 2 + n \leq 9n$$

where second to final inequality follows since $j + 1 \leq \log_2(n)$ □

The optimality of this algorithm is more difficult to prove, see [2] for details. In the following section we'll show we can improve this constant (i.e 9) in expectation using randomization.

2.2 Randomized Algorithms for the Cow Path Problem

In this section we'll improve our "sweeping" algorithm $\mathcal{A}$ by randomizing the direction in which walk and by randomizing the step length. In expectation we'll achieve constants as low as 4.915 (see [1]), almost half the distance in the expectation that would have been required from a deterministic algorithm.

The first idea is to choose the alternation of Left then Right or Right then Left uniformly at random. That is at Stage 1, flip a fair coin: if Heads go left at odd numbered stages and right on even stages, otherwise do the opposite. We'll call this simple randomized algorithm $\mathcal{A}_{coinflip}$ and see it reduces the factor in expectation down to 7.

Theorem 2 *The randomized algorithm* $\mathcal{A}_{coinflip}$ *has expected worst case distance traveled* $E[\mathcal{A}_{coinflip}] = 7n$

[0] or $d(i) = 2d(i-1), d(1) = 1$, we'll examine the recursive form when we design a randomized algorithm.

Proof Let $j \in \mathbb{N}$ be s.t. $2^j < n \leq 2^{j+1}$, then no matter what the algorithm takes first $\sum_{i=1}^{j} 2 * 2^i$ steps since at each stage up up to j we can't find the treat. Now at stage $j+1$, we go the correct direction with probability 1/2 and incorrect direction with probability 1/2. So the total distance is:

$$E[\text{Distance } \mathcal{A}_{coinflip}] = \sum_{i=1}^{j+1} 2 * 2^i + (1/2)n + 1/2(2^{j+1} + n) \leq 4n + n/2 + 2n + n/2 = 7n$$

□

We can further reduce the expected walking distance by randomizing our distance update rule. That is, if we initialize $d(1) = 2^{\mathcal{X}}$ where $\mathcal{X}$ is a uni[0,1] and afterwards proceed as normal with $d(i) = 2d(i-1)$. Finally, we can further reduce the expected walking time by increasing the base from 2 to some real number $r > 2$. The following algorithm summarizes:

Algorithm $\mathcal{A}_{rOPT}$

1. **Initialization 1:** Choose a starting direction Left or Right, uniformly at random. Let K = 0 if left, and 1 if right.
2. **Initialization 2:** Fix r = 3.5912
3. **Initialization 3:** Draw $\mathcal{X}$ from $uni[0,1]$ and initialize $d(1) = r^{\mathcal{X}}, d(0) = 1$
4. Repeat until we find the treat:
   (a) Let d(i) = $d(i-1) * r$
   (b) Explore to $(-1)^{i+K} d(i)$, if the treat is found stop, otherwise return to the origin
   (c) i := i+1

Theorem 3 *$\mathcal{A}_{rOPT}$ is randomized 4.5911-competitive and further is optimal over all randomized algorithms for the cowpath problem.*

Proof See [1], section 3 for the details. □

That completes our study of the cowpath problem. Now we'll return to scheduling.

3 Online Preemptive Scheduling

The Problem: In the online preemptive scheduling problem we start with m machines with differing speeds: $s_i, i \in [m]$. Wlog we'll assume $s_1 \geq s_2 \geq \ldots \geq s_m$. In the online problem, jobs with processing times p_j arrive one at a time and must be scheduled on some machine(s) as soon as they arrive. They may be scheduled preemptively, but each job may only run on one machine at a time and must be scheduled completely at the time of arrival. The goal is to design an online algorithm that minimizes the makespan of the online algorithm, compared to the optimal offline algorithm.

3.1 What's Known

3.1.1 The Offline Problem

Recall in class we studied the offline preemptive scheduling problem on parallel machines and showed the optimal makespan was:

$$OPT = \max\{P/S, P_1, S_1, \ldots, P_{m-1}/S_{m-1}\}$$

where P_i is the sum of the i longest processing times, and $S_k = \sum_{i=1}^{k} s_i$, the sum of the speeds of the k fastest machines. Further the optimal preemptive scheduling can be computed efficiently and in polynomial time. We'll need this result in our analysis of RatioStretch.

3.1.2 Approximation Algorithms for the Offline Problem

The offline problem also has a number of approximation results. For identical machines, Graham (1966) showed List Scheduling is a 2 approximation. Woeginger (2000) showed non-preemptive offline optimum is at most twice the makespan of the preemptive offline optimum.

3.1.3 Known Algorithms for the Online Problem

For m identical speed machines Chen et al (1995) gave a $\frac{1}{1-(1-1/m)^m} \to \frac{e}{e-1}$ competitive algorithm. For just two machines and for machines with non-decreasing speed ratios (that is: $s_1/s_2 \leq s_2/s_3 \leq \ldots$), Epstein et al gave an exact optimal competitive ratio online algorithms (the ratio is a function of the speeds $\vec{s}$).

For general parallel machines, Ebenlender (2004) show it's possible to build an optimal online schedule as long as the value of OPT is known to algorithm. Further they gave a 4-competitive deterministic algorithm and e competitive randomized algorithm for the problem. On the lower bound side, Epstein (2000) showed no algorithm could have absolute competitive ratio less than 2 for all inputs.

In the remainder of this section we study an optimal algorithm *RatioStretch* and it's consequences.

3.2 What's new: RatioStretch

RatioStretch is an optimal online algorithm for the online preemptive scheduling problem that works by scheduling incoming jobs preemptively on adjacent machines. To describe the algorithm formally lets introduce some notation. For a set of machines with speeds $s_1, s_2, \ldots, s_m$ let R be the optimal possible competitive ratio. Let OPT_i be the makespan of the optimal offline preemptive schedule on just the first i jobs. Finally we'll need to introduce the notion of a *virtual machine*.

Before any jobs are scheduled, the virtual machines correspond exactly to the real machines. Once jobs are scheduled with preemption, the virtual machine denoted V_i is a variable speed machine that runs the job via the fastest machine available, without overlap between the machines. So V_1, the first virtual machine runs with the fastest open machine at each time t, then V_2 runs as fast as possible with what's left and so on. RatioStretch will schedule arriving jobs on adjacent virtual machines (which can be easily translated back to our notion of regular machines with preemption) upon arrival. Figure 2 and it's explanation demonstrates the concept.

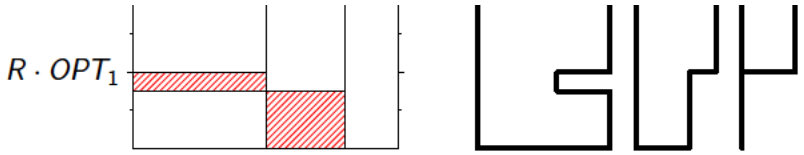

Figure 2: In this picture, the width of the column corresponds to the speed of the machine. The area of the job corresponds to it's processing time. Here, the red job arrives and is scheduled preemptively on the first and second machines, according to the left graphic. The right graphic shows the resultant virtual machines after the red job is scheduled. V_1 can use machine 1, until it hits the time where the red job is scheduled on the first machine, and it's forced to resort to the second machine. The second virtual machine V_2 is forced to use the third fastest machine since the red job is scheduled on the second machine and the first virtual machine has claimed the first machine, until the red job preempts over to the first machine. Then V_2 can use the second machine again. Etc.

RatioStretch

1. Input: $s_1, s_2, \ldots, s_n$ and optimal ratio R (R may be computed via a linear program, which we'll describe later).
2. On arrival of job j, compute OPT_j via the optimal offline algorithm.
3. Schedule the arriving job j on the slowest possible two adjacent virtual machines V_k, V_{k+1} s.t. the job completes in *exactly* time $= R * OPT_j$ (this is no difficulty, the scheduler can simply try all n combinations). This is the Stretch part of Ratio Stretch.
4. Update the virtual machines.

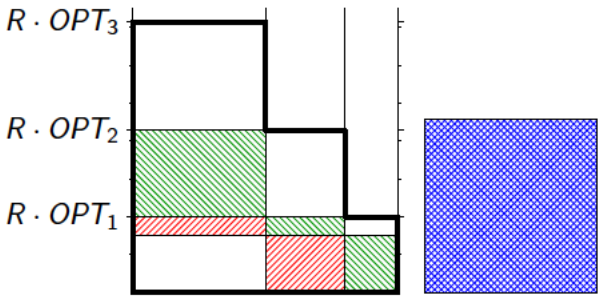

Figure 3: Here two jobs are scheduled preemptive by RatioStretch. The red job arrived first, and was scheduled on two adjacent machines so it finished exactly at time $R * OPT_1$. The free machines were then "collapsed" into virtual machines, and the arriving green job was scheduled across two *virtual machines* (here three actual machines) so it finished exactly at time $R * OPT_2$. The blue box has just arrived, OPT_3 is computed. Exercise: What exactly are the three variable speed virtual machines that must be considered?

Figure 3 shows an snapshot of the algorithm. For a proof that RatioStretch can always, for every job j, find a pair of adjacent virtual machines such that the completion time for $j = ROPT_j$, see [3]. The idea is to show that for every job j:

$$p_1 + p_2 + \ldots + p_j \leq R(s_1 OPT_{n-1} + s_2 OPT_{n-2} + \ldots + s_m OPT_{n-m}) \quad (1)$$

which follows from the fact that R is a valid competitive ratio and the fact that no machines run after OPT_n, at most one machine runs after OPT_{n-1}, at most two machines run after OPT_{n-2} and so on (by the definition of OPT_i).

3.3 On Finding the Optimal R

As mentioned, we can find the optimal competitive ratio for the input machine speeds via solving a linear programming like in Figure 4, which essentially searches over all possible processing times while maintaining Eq. 1. For more details see [3], section 3.

3.4 Conclusions and Extensions

RatioStretch is an optimal deterministic algorithm for the online preemptive scheduling problem. For each instance, the algorithm computes the optimal competitive ratio R - however a general R is not known. In [3], an example is given with $m > 100$ jobs, for which the competitive ratio is $\approx 2.02 \geq 2$, which is best known lower bound. Since RatioStretch is optimal over the class of both deterministic and randomized algorithms, the e competitive ratio shown previous is the best upper instance independent upper bound on the competitive ratio.

$$
\begin{array}{lrcl}
\text{Variables:} & q_1, q_2, q_3, O_1, O_2, O_3 \geq 0 & & \\
\text{Maximize:} & R & = & q_1 + q_2 + q_3 \\
\text{Subject to:} & 1 & = & s_1 O_3 + s_2 O_2 + s_3 O_1 \\
q_1 & & \leq & (s_1 + s_2 + s_3) O_1 \\
q_1 + q_2 & & \leq & (s_1 + s_2 + s_3) O_2 \\
q_2 & & \leq & s_1 O_2 \\
q_1 + q_2 + q_3 & & \leq & (s_1 + s_2 + s_3) O_3 \\
q_2 + q_3 & & \leq & (s_1 + s_2) O_3 \\
q_3 & & \leq & s_1 O_3 \\
q_2 & & \leq & q_3
\end{array}
$$

Figure 4: Linear Program to find R for three machines with speeds s_1, s_2, s_3.

References

[1] Kao, Reif, Tate 1996 *Searching In A Unknown Environment: An Optimal Randomized Algorithm for the Cow Path Problem* Information and Computation 131.1 (1996): 63-79.

[2] Baeza-Yates, R.A., et al 1993. *Searching in the Plane.* Information and Computation 16 234-252.

[3] Ebenlendr, Jawor, and Ji Sgall. *Preemptive Online Scheduling: Optimal Algorithms for All Speeds.* Algorithmica 2009.