IOE 691: Approximation & Online Algorithms

Lecture Notes: Online Potential Functions

Instructor: Viswanath Nagarajan

Any online algorithm can be viewed as an alternating sequence $\sigma_1, a_1, \sigma_2, a_2, \cdots \sigma_t, a_t, \cdots$ where in each step t, the incremental information σ_t is observed and the algorithm makes incremental decision a_t. One can also split the optimal solution into a sequence $o_1, o_2, \cdots o_t, \cdots$ where o_t is the incremental decision taken by the optimum in each step t. Let $\Delta_t\mathsf{ALG} = cost(a_t)$ and $\Delta_t\mathsf{OPT} = cost(o_t)$ denote the incremental costs in step t for the online algorithm and optimum respectively. Then, the overall online cost is $\mathsf{ALG} = \sum_t \Delta_t\mathsf{ALG}$ and the optimal cost $\mathsf{OPT} = \sum_t \Delta_t\mathsf{OPT}$. A natural approach to proving α-competitiveness is to show $\Delta_t\mathsf{ALG} \leq \alpha \cdot \Delta_t\mathsf{OPT}$ at each step t. Unfortunately, for many online algorithms this is not true (or difficult to prove). A more flexible approach is to introduce a "potential function" Φ that helps relate the current online and optimal solutions. Then, we prove the relation $\Delta_t\mathsf{ALG} + \Delta_t\Phi \leq \alpha \cdot \Delta_t\mathsf{OPT}$ at each step t. Adding this relation over all steps, we get $\mathsf{ALG} + \Phi_{final} - \Phi_{initial} \leq \alpha \cdot \mathsf{OPT}$, which implies α-competitiveness as long as $\Phi \geq 0$ (non-negative potential function) and $\Phi_{initial} = 0$ (zero potential at the start).

1 Flowtime Scheduling

Consider a single machine that processes n jobs arriving online. Each job j arrives at some time r_j at which point the online algorithm is made aware of the job's processing time p_j. The algorithm has no information about job j before its arrival time. The machine is allowed to *preempt* jobs, i.e., pause the processing of any job and resume it later on. Job j is completed when the machine works on j for a total of p_j time. Let C_j denote the completion time of job j; note that $C_j \geq r_j + p_j$. Then, the *flowtime* of job j is $F_j := C_j - r_j$, the time spent by j in the system. The objective is to minimize the total flowtime of all jobs.

Any job that has been released but not completed is called an *active* job. A very natural algorithm is *shortest remaining processing time* (SRPT), which always runs the active job with the smallest *remaining* processing time.

Theorem 1.1 *SRPT is 1-competitive for single-machine flowtime minimization.*

Proof: Let OPT denote an optimal schedule and ALG the SRPT schedule. Let N_t (resp. N_t^*) denote the number of active jobs in ALG (resp. OPT) at any time t. Note that $\Delta_t\mathsf{ALG} = N_t$ and $\Delta_t\mathsf{OPT} = N_t^*$. We will show:

$$N_t \leq N_t^*, \quad \forall t, \tag{1}$$

which when addded would imply $\mathsf{ALG} = \sum_t \Delta_t\mathsf{ALG} \leq \sum_t \Delta_t\mathsf{OPT} = \mathsf{OPT}$ as needed.

In order to prove (1) we use an "exchange argument". Specifically, we will maintain a schedule σ (initially equals OPT) that is modified iteratively until $\sigma = \mathsf{ALG}$. Crucially, we will ensure that there is no increase in the number of active jobs in schedule σ (at any time). As $\sigma = \mathsf{OPT}$ initially and $\sigma = \mathsf{ALG}$ finally, this would imply (1).

Modifying schedule σ. If the current schedule $\sigma \neq \mathsf{ALG}$ then let t denote the earliest time when σ

disagrees with ALG. Let j (resp. i) denote the job processed at time t in ALG (resp. σ). Note that $j \neq i$ as ALG and σ differ at this time. Let p'_j and p'_i denote the remaining processing time of i and j at time t (in both schedules ALG and σ). Note that $p'_j \leq p'_i$ by the SRPT rule. We now modify schedule σ into σ' by only changing how i and j are processed. Let I denote all time steps beyond t where σ runs either job i or j: these times need not be contiguous. Note that there are $p'_i + p'_j$ time steps in I. The new schedule σ' is identical with σ on all time steps other than I. For the time steps in I, schedule σ' runs (1) job j in the first p'_j time steps of I and (2) job i in the last p'_i time steps of I. As $p'_i \geq p'_j$ it is easy to see that the number of active jobs in σ' is at most that in σ (at any time). Finally, note that the new schedule σ' agrees with ALG on more time steps than σ: so we only modify the schedule σ a finite number of times before it equals ALG. ■

2 Non-clairvoyant Flowtime Scheduling

We now consider the flowtime minimization problem in the "non clairvoyant" setting, where the online algorithm does not even know the processing time of the jobs after their arrival. As before, each job $j \in [n]$ arrives at time r_j: at this point the algorithm only knows that a new job has arrived. The job's processing time p_j remains unknown to the algorithm. When the algorithm has performed p_j units of work on job j, it gets to know that job j has completed (and left the system). The goal is to minimize the total flowtime (as before). Again, the algorithm is allowed to preempt jobs. Note that SRPT is not a valid algorithm in the non-clairvoyant setting as it uses information about job processing times. It turns out that no online algorithm can achieve any reasonable competitive ratio:

Theorem 2.1 ([1]) *Every deterministic online algorithm for non-clairvoyant flowtime minimization has $\Omega(n^{1/3})$ competitive ratio.*

To get around this impossibility, we allow for *resource augmentation*, where the online algorithm can run the jobs at a faster speed than the optimum. In particular, an s-speed c-competitive online algorithm is one that runs at speed $s \geq 1$ and finds a solution with flowtime at most c times the optimum (that only has speed 1). Note that the above result says that any 1-speed online algorithm has competitive ratio $c = \Omega(n^{1/3})$. In light of this, a natural question is: can we get an $O(1)$-speed $O(1)$-competitive algorithm?

A natural non-clairvoyant algorithm is *round robin* (RR), which divides its speed s equally among all active jobs. Formally, if N_t denotes the set of active jobs at time t then RR allocates speed $\frac{s}{|N_t|}$ to each job $j \in N_t$. We will show:

Theorem 2.2 *For any $\epsilon > 0$, RR is a $(2+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive algorithm for non-clairvoyant flowtime minimization.*

We will use a potential function in the analysis. For any time t, let N_t (resp. N^*_t) denote the number of active jobs under RR (resp. OPT). We will define a (non negative) potential function Φ that depends on the current "state" of RR and OPT so that:

$$\frac{\partial}{\partial t}\mathsf{ALG} \; + \; \frac{\partial}{\partial t}\Phi \quad \leq \quad \left(2 + \frac{1}{\epsilon}\right) \cdot \frac{\partial}{\partial t}\mathsf{OPT}. \tag{2}$$

This would imply Theorem 2.2.

The potential. Fix any time t. Let a_j (resp. o_j) denote the remaining work for job j in the RR (resp. OPT) schedule. Note that $a_j > 0$ for all $j \in N_t$ (active jobs in RR). Let

$$z_j \;=\; (a_j - o_j)^+ \;=\; \max\{a_j - o_j\,,\,0\}$$

denote the amount by which RR is behind OPT in processing job j. Let $M_t = \{j \in N_t : z_j > 0\}$ be the active jobs (in RR) for which RR is lagging behind OPT. We re-index jobs in M_t in decreasing order of z_j. Then, the potential at time t is:

$$\Phi \;:=\; \frac{1}{\epsilon} \cdot \sum_{j \in M_t} j \cdot z_j,$$

which corresponds to the total amount by which RR is behind OPT (weighted appropriately).

We observe that there is no discontinuous change in the potential at any time. In particular, consider the following discrete events:

1. *A new job j arrives.* There is no change in Φ as $a_j = o_j$ and so $z_j = 0$.
2. *Job j completes in RR.* There is no change in Φ as $a_j \le o_j$ and so $z_j = 0$.
3. *Job j completes in* OPT. There is no change in Φ as z_j remains the same: $z_j = a_j$ (if $j \in N_t$) or $z_j = 0$ (if $j \notin N_t$).
4. *The ordering of jobs in M_t changes.* This can only happen there is a tie in the z_j values: so there is no change in Φ. If a new job enters M its z_j value must be infinitesimal, so again Φ doesn't change.

We are now ready to prove (2). Consider any time t. It suffices to show:

$$N_t \;+\; \frac{\partial}{\partial t}\Phi \quad\le\quad \left(2 + \frac{1}{\epsilon}\right) \cdot N_t^*. \tag{3}$$

We drop the subscript t for notational ease. We now prove (3) for an infinitesimal time period $[t, t+\partial t)$. Note that the number of active jobs in OPT is $N^* \ge N - M$. So it suffices to show:

$$N \;+\; \frac{\partial}{\partial t}\Phi \;-\; \left(2 + \frac{1}{\epsilon}\right) \cdot (N - M) \quad\le\quad 0. \tag{4}$$

To bound the rate of increase in Φ, we consider separately the cases that RR and OPT work.

RR works. Suppose that RR is allowed to run at speed $2(1+\epsilon)$. Then, it runs each active job $j \in N$ at speed $\beta = \frac{2(1+\epsilon)}{N}$. So whenever $z_j > 0$, it increases at rate $-\beta$. Hence, the rate of potential increase

$$\frac{\partial}{\partial t}\Phi \le -\frac{\beta}{\epsilon}(1 + 2 + \ldots + M) = -(1 + \frac{1}{\epsilon}) \cdot \frac{M(M+1)}{N} \le -(1 + \frac{1}{\epsilon})\frac{M^2}{N}.$$

OPT works. We can assume OPT works on a *single* job j during the infinitesimal time. If $j \notin M$ then there is no change in Φ. Now suppose $j \in M$; recall that we re-index these jobs as above. The rate of potential increase $\frac{\partial}{\partial t}\Phi \le \frac{1}{\epsilon} \cdot j \le \frac{1}{\epsilon} \cdot M$.

Adding the increase in Φ, the left-hand-side of (4) is at most:

$$\begin{aligned} & N - \left(1+\frac{1}{\epsilon}\right)\frac{M^2}{N} + \frac{M}{\epsilon} - \left(2+\frac{1}{\epsilon}\right)\cdot(N-M) \\ = \; & -\left(1+\frac{1}{\epsilon}\right)\frac{1}{N}\cdot(M^2-2MN) - \left(1+\frac{1}{\epsilon}\right)N \\ = \; & -\left(1+\frac{1}{\epsilon}\right)\frac{1}{N}\cdot(M-N)^2 \quad \leq \quad 0. \end{aligned}$$

This completes the proof of (4) and hence Theorem 2.2.

3 Fractional Set Cover

We consider the unweighted set cover problem. Given m sets $\{S_i\}_{i=1}^m$ with unit costs and n elements arriving online, we wish to cover each element immediately upon arrival and minimize the number of sets used. Consider the LP relaxation:

$$\begin{aligned} & \min \sum_{i=1}^m x_i \\ & \sum_{i:S_i \ni e} x_i \geq 1, \quad \forall e \in [n] \\ & x \geq 0. \end{aligned}$$

We consider the fractional online algorithm for this LP. When element e arrives, repeat the following continuously:

$$\frac{\partial}{\partial t} x_i = x_i + \delta \;\; \text{for all } i : S_i \ni e, \tag{5}$$

until $\sum_{i:S_i \ni e} x_i \geq 1$. We set $\delta = \frac{1}{m}$.

Let $o^* \in [0,1]^m$ denote the optimal LP solution that will also be viewed as increasing over time (similar to the online solution x). So $\mathsf{OPT} = \sum_{i=1}^m o_i^*$ and $\mathsf{ALG} = \sum_{i=1}^m x_i$. Consider the following *potential function*:

$$\Phi \;=\; 2 \cdot \sum_{i=1}^m o_i^* \cdot \ln\left(\frac{1}{x_i+\delta}\right).$$

Note that this is a function of the current online solution x as well as the optimal solution o^*.

We will show that

$$\frac{\partial}{\partial t}\mathsf{ALG} \;+\; \frac{\partial}{\partial t}\Phi \quad \leq \quad 2\ln(1/\delta)\cdot\frac{\partial}{\partial t}\mathsf{OPT}. \tag{6}$$

We consider separately the situations where the online/optimal solution increases. We first increase o^* and then increase x.

Optimum increases. Suppose o^* increases and x stays the same. We view o_i^* as increasing at rate one (for the appropriate time). So, $\frac{\partial \mathsf{OPT}}{\partial t} = 1$ and

$$\frac{\partial}{\partial t}\Phi = 2\cdot\ln\left(\frac{1}{x_i+\delta}\right) \leq 2\cdot\ln(1/\delta) = 2\,\ln(1/\delta)\cdot\frac{\partial}{\partial t}\mathsf{OPT}.$$

Moreover, $\frac{\partial \mathsf{ALG}}{\partial t} = 0$, which implies (6).

Online increase. Suppose now that x increases (due to arrival of element e) and o^* stays the same. By the update in (5) above,

$$\frac{\partial \mathsf{ALG}}{\partial t} = \sum_{i:S_i \ni e} (x_i + \delta) \leq 1 + \delta m = 2.$$

Moreover, the rate of increase in the potential is,

$$\frac{\partial \Phi}{\partial t} = 2 \cdot \sum_{i:S_i \ni e} o^*_i \cdot \frac{\partial}{\partial t}(-\ln(x_i + \delta)) = -2 \sum_{i:S_i \ni e} \frac{o^*_i}{x_i + \delta} \frac{\partial x_i}{\partial t} = -2 \sum_{i:S_i \ni e} o^*_i \leq -2,$$

where we used the update equation (5) and the fact that the optimal solution o^* also covers e. Adding the rate of increase in ALG and Φ, and using the fact that $\frac{\partial \mathsf{OPT}}{\partial t} = 0$ here, we get (6).

Wrapping up. Integrating (or adding) (6) over all time,

$$\mathsf{ALG} + \Phi_{final} - \Phi_{initial} \leq 2\ln(1/\delta) \cdot \mathsf{OPT}.$$

Note that $\Phi_{initial} = 0$ when $x = o^* = 0$. So we proved the following (again).

Theorem 3.1 *There is a $2\ln(m)$-competitive algorithm for fractional set cover.*

4 Load Balancing on Unrelated Machines

There are m machines that are fixed upfront. A sequence of n jobs arrives online. Upon the arrival of each job j, the algorithm gets to know the processing times $\{p_{ij}\}_{i=1}^m$ of this job on each of the m machines. The online algorithm needs to assign the job j to some machine $a(j)$ immediately. The objective is to minimize the maximum load on any machine, i.e.,

$$\min_{a:[n]\to[m]} \quad \max_{i=1}^{m} \quad \sum_{j:a(j)=i} p_{ij}.$$

This objective is also known as the *makespan*.

We will provide an online algorithm for the following variant. Given a target makespan T, find an online assignment $a : [n] \to [m]$ such that:

- the makespan of a is at most $\alpha \cdot T$, or
- the optimal value is more than T.

Given such an algorithm, one can also obtain an $O(\alpha)$-competitive online algorithm for the load balancing problem (defined above).

Note that we can assume that the target $T = 1$ by scaling all processing times. So, our goal is to come up with an online algorithm that has makespan $\leq \alpha$ *assuming* that the optimal makespan is at most one. We will never assign job j to a machine i with $p_{ij} > 1$. (Note that no solution with makespan ≤ 1 can use this assignment.)

If L_i denotes the load on each machine i then the makespan is $\max_{i=1}^m L_i$. It is difficult to analyze the max objective as it is not smooth. Instead, we will work with a "soft max" function defined as follows:

$$\Phi(L_1, \cdots L_m) = \sum_{i=1}^m c^{L_i},$$

where $c \in (1,2)$ is some constant fixed later.

The algorithm is the natural greedy algorithm w.r.t. the soft-max objective. In other words, when job j arrives, it gets assigned to the machine:

$$\arg \min_{i \in [m]: p_{ij} \le 1} \left(c^{L_i + p_{ij}} - c^{L_i} \right), \tag{7}$$

where $\{L_i\}_{i=1}^m$ denotes the current load on each machine.

Let $\{A_i\}_{i=1}^m$ denote the algorithm's final load on the machines. Also, let $o : [n] \to [m]$ denote the optimal assignment, which is assumed to have makespan at most one.

Lemma 4.1 *The potential increase when job j arrives,*

$$\Delta_j \Phi \;\le\; (c-1) c^{A_{o(j)}} \cdot p_{o(j),j}.$$

Proof: Let $\{L_i\}_{i=1}^m$ denote the current loads when job j arrives. We know that $p_{o(j),j} \le 1$ as the optimal makespan is at most one. So machine $o(j)$ will be considered as an option for job j in (7). It follows that the potential increase $\Delta_j \Phi$ is at most:

$$c^{L_{o(j)} + p_{o(j),j}} - c^{L_{o(j)}} \;=\; c^{L_{o(j)}} \left(c^{p_{o(j),j}} - 1 \right) \;\le\; c^{L_{o(j)}} \cdot (c-1) p_{o(j),j} \;\le\; (c-1) c^{A_{o(j)}} \cdot p_{o(j),j}.$$

Above, the first inequality uses the fact $c^x \le 1 + (c-1)x$ for all $c > 1$ and $x \in [0,1]$. The last inequality uses the fact that loads are monotonically increasing: so $L_{o(j)} \le A_{o(j)}$. ∎

Lemma 4.2 *The net potential increase in the algorithm $\Phi_{final} - \Phi_{initial} \le (c-1) \cdot \Phi_{final}$.*

Proof: Adding the potential increases (Lemma 4.1) over all jobs, the net increase $\Phi_{final} - \Phi_{initial}$ equals:

$$\sum_{j=1}^n \Delta_j \Phi \le (c-1) \sum_{j=1}^n c^{A_{o(j)}} \cdot p_{o(j),j} = (c-1) \sum_{i=1}^m c^{A_i} \sum_{j: o(j)=i} p_{ij} \le (c-1) \sum_{i=1}^m c^{A_i} = (c-1) \cdot \Phi_{final},$$

where the last inequality uses that the optimal makespan is at most one. ∎

Theorem 4.1 *Fix any $\epsilon > 0$. There is an online algorithm that given target T, finds an assignment of makespan $\left(\frac{\ln m + \ln(1/\epsilon)}{\ln(2-\epsilon)} \right) \cdot T$ if the optimal makespan is at most T.*

Proof: Note that $\Phi_{initial} = m$ as all loads are initially zero. By Lemma 4.2 it then follows that $\Phi_{final} \le \frac{m}{2-c}$. As $\Phi_{final} \ge c^{A_i}$ for all machines i, we have:

$$\max_{i=1}^m A_i \quad \le \quad \frac{1}{\ln c} \cdot \ln \Phi_{final} \quad \le \quad \frac{1}{\ln c} \cdot \ln \left(\frac{m}{c-2} \right).$$

Setting $c = 2 - \epsilon$ proves the theorem. ∎

Note that as $\epsilon \to 0$, we get a competitive ratio of $\approx 1.44 \ln(m) + O(1)$ where m is the number of machines. There is also a nearly-matching $\ln m$ lower bound on the competitive ratio, even for randomized algorithms.

5 Experts and Online Learning

Consider an online prediction setting over a horizon of T steps. In each step, an algorithm needs to predict a binary outcome (say Yes or No). There are n "experts" each of which offers their prediction in each step. Can we combine these expert predictions to maximize our prediction accuracy?

In each step t, the online algorithm can use the predictions of all experts for step t (as well as prior steps) to come up with its own prediction. The algorithm also observes the actual outcome in step t, *after making its own prediction*. So, before moving to step $t+1$, the algorithm gets to know which of the experts were correct/incorrect in step t.

An important question is: what benchmark should we compare the online algorithm to? So far, our focus has been competitive analysis, where we compared the online algorithm to an omniscient optimal solution (that known all future information). In the online learning setting, this benchmark turns out to be too strong: so no algorithm can achieve a good competitive ratio. Instead, we will consider a weaker benchmark: the *best single expert*. In particular, we will obtain online algorithms where the number of mistakes is not much more than the best expert.

Perfect Expert. As a warmup, we first consider the case when there is some expert who makes no mistake. A natural algorithm in this case is the following. Eliminate any expert who makes a mistake (at any step), and use the "majority" prediction from the remaining experts. Let $E \subseteq [n]$ denote the set of remaining experts. Initially $E = [n]$. Whenever the online algorithm makes a mistake, we eliminate at least half of E, so the size $|E|$ reduces by at least a factor of 2. Moreover, the perfect expert will never be eliminated. So the online algorithm never makes a mistake after $|E|$ drops to 1. It follows that the number of mistakes is at most $\log_2 n$. To summarize:

Theorem 5.1 *Assuming that there is some expert who makes no mistake, there is an online experts algorithm that makes at most* $\log_2 n$ *mistakes, where* n *is the number of experts.*

Imperfect experts. We now consider the general case where every expert may make mistakes. For each $i \in [n]$, let $m_i \in [0, T]$ denote the number of mistakes made by expert i. Our benchmark is $M^* := \min_{i=1}^{n} m_i$ the number of mistakes by the best single expert. We will show:

Theorem 5.2 *For any* $\epsilon \in (0, \frac{1}{2})$*, there is an online experts algorithm that makes at most*

$$(2 + 2\epsilon) \cdot M^* + \frac{2 \ln n}{\epsilon}$$

many mistakes.

The online algorithm maintains a weight for each expert (corresponding to their performance so far) and uses their *weighted majority* as its prediction. Formally, let $w_t(i)$ denote the weight of expert i at the start of step t. Initially $w_1(i) = 1$ for all $i \in [n]$. For each step $t = 1, 2, \cdots T$:

1. Let W_{yes} (resp. W_{no}) be the total weight of experts predicting Yes (resp. No).
2. If $W_{yes} \geq W_{no}$ output "Yes", else output "No".
3. Observe the actual outcome.

4. For each $i \in [n]$, update its weight as follows:

$$w_{t+1}(i) = \begin{cases} (1-\epsilon) \cdot w_t(i) & \text{if expert } i \text{ made a mistake in step } t \\ w_t(i) & \text{otherwise} \end{cases} \tag{8}$$

Proof of Theorem 5.2. Let $\Phi_t := \sum_{i=1}^n w_t(i)$ denote the total weight in step t. Let W'' (resp. W') denote the total weight of experts who predicted correctly (resp. incorrectly) in step t. By the updates in (8), we have $\Phi_{t+1} - \Phi_t = -\epsilon \cdot W'$. Moreover, if the algorithm makes a mistake in step t then it must be that $W' \geq W''$, i.e., $W' \geq \frac{1}{2}\Phi_t$. This implies $\Phi_{t+1} - \Phi_t \leq -\frac{\epsilon}{2} \cdot \Phi_t$, i.e.,

$$\Phi_{t+1} \leq (1 - \epsilon/2) \cdot \Phi_t, \quad \text{if online made a mistake in step } t.$$

So, if ALG denotes the total number of mistakes in the online algorithm, we have:

$$\Phi_{T+1} \leq (1-\epsilon/2)^{\mathsf{ALG}} \cdot \Phi_1 = (1-\epsilon/2)^{\mathsf{ALG}} \cdot n.$$

Notice that for each $i \in [n]$ we have $w_{T+1}(i) = (1-\epsilon)^{m_i}$. So,

$$\Phi_{T+1} \geq \max_{i=1}^n (1-\epsilon)^{m_i} = (1-\epsilon)^{M^*}.$$

Combining the two inequalities above, $(1-\epsilon)^{M^*} \leq (1-\epsilon/2)^{\mathsf{ALG}} \cdot n$. Taking logarithms and using Fact 5.1 below, we get:

$$-(\epsilon + \epsilon^2) \cdot M^* \leq \ln(1-\epsilon) \cdot M^* \leq \ln(1 - \frac{\epsilon}{2}) \cdot \mathsf{ALG} + \ln n \leq -\frac{\epsilon}{2} \cdot \mathsf{ALG} + \ln n.$$

Rearranging, we get $\mathsf{ALG} \leq 2(1+\epsilon) \cdot M^* + \frac{2 \ln n}{\epsilon}$, which proves Theorem 5.2.

Fact 5.1 *For any $\delta \in (-\frac{1}{2}, \frac{1}{2})$, we have $\delta - \delta^2 \leq \ln(1+\delta) \leq \delta$.*

Note that the above weighted majority algorithm is deterministic. It turns out that no deterministic online algorithm can do much better. Consider the case of $n = 2$ experts with all-Yes and all-No predictions. Let $\mathcal{A}$ be any deterministic algorithm. The actual outcomes $\sigma_1, \sigma_2 \cdots \sigma_T$ are constructed adversarially: if $\mathcal{A}$ outputs Yes in step t then $\sigma_t = No$ and vice-versa. Clearly, the number of mistakes made by $\mathcal{A}$ is T. However, the best single expert makes $M^* \leq T/2$ mistakes. So, we must have $\mathsf{ALG} \geq 2 \cdot M^*$ for any deterministic algorithm. This implies that the factor $2(1+\epsilon)$ in Theorem 5.2 is nearly best possible.

5.1 Randomized experts

We will now see that randomization allows us to obtain a better performance. We assume that the actual outcomes $\sigma_1, \sigma_2 \cdots \sigma_T$ are fixed upfront, but unknown to the online algorithm. Then, the algorithm makes its randomized predictions. Finally, we bound the expected number of mistakes in the online algorithm. Again, our benchmark is $M^* = \min_{i=1}^n m_i$ the best single expert.

Theorem 5.3 *For any $\epsilon \in (0, \frac{1}{2})$, there is a randomized experts algorithm that makes at most*

$$(1+\epsilon) \cdot M^* + \frac{\ln n}{\epsilon}$$

many mistakes in expectation.

The "randomized weighted majority" algorithm also maintains weights $w_t(i)$ on experts exactly as in the deterministic algorithm (8). However, instead of using the majority prediction, in each step t, it chooses the prediction of one expert according to the probabilities:

$$p_t(i) = \frac{w_t(i)}{\Phi_t}, \quad \forall i \in [n].$$

Recall that $\Phi_t = \sum_{i=1}^{n} w_t(i)$ is the total weight in step t. Note that the weights w_t evolve deterministically even in this randomized algorithm, and are identical to those in the deterministic algorithm.

Proof of Theorem 5.3. Consider any step t. Let W'' (resp. W') denote the total weight of experts who predicted correctly (resp. incorrectly) in step t. Then,

$$\mathsf{ALG}_t := \Pr[\mathsf{ALG} \text{ makes a mistake in step } t] = \frac{W'}{\Phi_t}.$$

This means $\Phi_{t+1} - \Phi_t = -\epsilon \cdot W' = -\epsilon \Phi_t \cdot \mathsf{ALG}_t$. Rearranging,

$$\frac{\Phi_{t+1}}{\Phi_t} = 1 - \epsilon \cdot \mathsf{ALG}_t.$$

Taking the product over all t and using $\mathsf{ALG} = \sum_{t=1}^{T} \mathsf{ALG}_t$, we get:

$$\frac{\Phi_{T+1}}{\Phi_1} = \prod_{t=1}^{T} (1 - \epsilon \cdot \mathsf{ALG}_t) \leq e^{-\epsilon \cdot \mathsf{ALG}}.$$

As before, we have $\Phi_1 = n$ and $\Phi_{T+1} \geq (1-\epsilon)^{M^*}$. Combined with the above and taking logs,

$$-\epsilon \cdot \mathsf{ALG} \geq \ln\left(\frac{\Phi_{T+1}}{\Phi_1}\right) \geq \ln(1-\epsilon) \cdot M^* - \ln n \geq -(\epsilon + \epsilon^2) \cdot M^* - \ln n.$$

This implies $\mathsf{ALG} \leq (1+\epsilon) \cdot M^* + \frac{\ln n}{\epsilon}$, which proves Theorem 5.3.

Regret. Theorem 5.3 gives a nearly *additive* error bound. This can be re-stated in the following way. Define the *regret* of an algorithm as the difference between its (expected) mistakes and that of the best single expert. So,

$$\textit{regret} = \mathsf{ALG} - M^* \leq \epsilon \cdot M^* + \frac{\ln n}{\epsilon}.$$

Setting $\epsilon = \sqrt{\frac{\ln n}{M^*}}$, it follows that

$$\textit{regret} \leq 2\sqrt{M^* \ln n} \leq 2\sqrt{T \ln n}.$$

This shows that the regret of the randomized experts algorithm grows *sublinearly* in the horizon T. In words, the regret-per-step tends to zero as $T \to \infty$.

References

[1] Rajeev Motwani, Steven J. Phillips, and Eric Torng. Non-clairvoyant scheduling. *Theor. Comput. Sci.*, 130(1):17–47, 1994.