7. Online algorithms

- Problem formulation
- Competitive ratio
- Buy-vs-rent problem
- Secretary problem

Online algorithms

- Problem
  - Input: provided as a “stream” of data
  - Objective: make optimal decision at each point in time, based on the data provided so far
  - Two approaches:
  - Probabilistic
    - input from a (parametrized) distribution
    - learn the distribution and then predict next input
  - Worst-case
    - input can be anything, in particular worst-case
  - We focus on worst-case analysis
  - We will see that probabilistic approaches work quite well under worst-case
  - Example: Online bipartite matching [Karp,Vazirani,Vazirani 1990]
    - Any deterministic algorithm has a *competitive ratio* at most 1/2

Online algorithms

- Example: buy-vs-rent
  - Consider buying vs. renting skiing equipment
  - Cost \$500 to buy and \$50 to rent
  - It is optimal to rent if skiing less than 10 times, but we do not know how many times we will ski
  - at each time k we are given a choice to rent or buy, unless we have already bought
  - Input at time k: the fact that we are going on a k-th ski trip
  - Decision: rent or buy
  - Deterministic strategy
    - rent until n and buy
    - completely described by n

  - What is a good strategy?

Online algorithms

- *Competitive analysis* of an online algorithm
  - analyze the performance of an online algorithm by comparing it to the best off-line algorithm that can see the data in advance
  - *Competitive ratio* of an online algorithm
    - Imagine adversary deliberately chooses difficult data for your algorithm
    - compare to the optimal algorithm for that data
    - *competitive ratio* is the ratio between the cost of the online algorithm under adversarial data and the optimal algorithm
    - let $data^* = \arg\max \text{Cost}(ALG_n, data)$

$$\text{competitive ratio}(ALG_n) = \frac{\max_{\text{data}} \text{Cost}(ALG_n, data)}{\min_{ALG} \text{Cost}(ALG, data^*)}$$

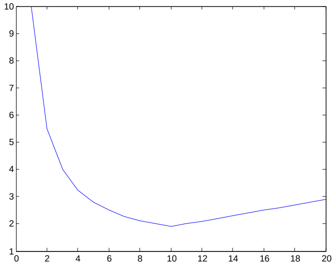
- Competitive ratio of an algorithm for buy-vs-rent problem
  - If $n = 1$, the worst case data is never ski after first trip. Then cost of online algorithm is $500. Cost of the optimal algorithm is $50. Therefore, competitive ratio is 10.
  - Similarly, for general $n \leq 10$, cost of online algorithm is $50*(n-1)+$500. Cost of optimal algorithm is $50*n.

$$\text{Competitive ratio} = 1 + \frac{9}{n}$$

  - what if $n > 10$?

Online algorithms

- Competitive analysis



- Competitive ratio is minimized when $n = 10$
- General rule for buy-vs-rent problem is to keep renting until what we have spent equals the cost of buying
- **Claim.** Competitive ratio of 2 can be always achieved for general buy-vs-rent problem. (Homework)
- Intuitively, "probabilistic" perspective is that once we reach 10, we can predict that there will be another 10 more.

Secretary problem

- Problem
  - There are n candidates and one position
  - Objective: hire the best one
  - Input: each time invite one candidate for an interview. At time k, we know exact ranking of k candidates seen so far.
  - Decision: at time k, can choose to hire the current candidate (k-th) or move on to the next one.
  - How can we maximize the probability of hiring the best one? (We only care whether we got the best one or not)
- Applications
  - Housing, online dating, etc.
  - Googol game [Scientific American, 1960]
    - ⋆ Alice writes n numbers between 1 and Googol(10^{100}) on n sheets of paper
    - ⋆ Bob turns one piece over at a time and stop when he thinks he has the largest among n

Secretary problem

- Optimal strategy
  - Assume candidates are ordered randomly
    - There are n candidates
    - Let 1 be the best candidate, and n the worst candidate
    - They arrive in a random order: $n!$ possible permutations have equal probability
      (equivalently, we can choose to invite candidates in random order)
    - Let c_k be the random variable representing the ranking(1 is best) of the candidate interviewed at time k

$$\begin{aligned}
\mathbb{P}(c_1 = m) &= 1/n \;\; (= (n-1)!/n!\,) \\
\mathbb{P}(c_k = m) &= 1/n \\
\mathbb{P}(c_2 < c_1) &= 1/2 \quad \text{(by symmetry)} \\
\mathbb{P}(c_k < c_1, \ldots, c_{k-1}) &= 1/k \\
\mathbb{P}(1 \in \{c_1, c_2, \ldots, c_k\}) &= k/n \\
\mathbb{P}(\text{best among first } k \text{ candidates is in the fist } \ell) &= \ell/k
\end{aligned}$$

    - Conditional probability: $\mathbb{P}(B|A) = \mathbb{P}(A \text{ and } B)/\mathbb{P}(A)$

$$\mathbb{P}(\text{best among first } k \text{ candidates is in the fist } \ell | c_{k+1} = 1) \;=\; \ell/k$$

    - two events are independent

Secretary problem

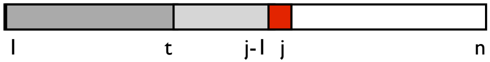
- Consider a strategy of
  - first phase: interview t candidates without hiring any
  - second phase: after time t, continue until you meet someone who is better than everyone seen so far, and hire that person
  - precisely, look at $c_1, \dots, c_t$ and then find the first j such that $c_j < c_1, \dots, c_{j-1}$
- Analyzing the probability of success

$$\mathbb{P}(\text{we succeed}) = \sum_{j=t+1}^{n} \mathbb{P}(\text{we hire } j\text{-th candidate and succeed})$$

$$\begin{aligned}
\mathbb{P}(\text{hire } t+1\text{-th candidate}) &= \mathbb{P}(c_{t+1} < \{c_1, \dots, c_t\}) = \frac{1}{t+1} \\
\mathbb{P}(\text{hire } t+1\text{-th and succeed}) &= \mathbb{P}(c_{t+1} = 1)\,\mathbb{P}(c_{t+1} < \{c_1, \dots, c_t\} | c_{t+1} = 1) = \frac{1}{n} \\
\mathbb{P}(\text{hire } t+2\text{-th and succeed}) &= \mathbb{P}(c_{t+2} = 1) \times \mathbb{P}(t+1 \text{ is not hired} \mid c_{t+2} = 1) \\
&= \frac{1}{n}\mathbb{P}(t+1 \text{ is not hired}) \\
&= \frac{1}{n}\mathbb{P}(\text{best among first } t+1 \text{ candidates is in first } t \\
&= \frac{1}{n}\frac{t}{t+1}
\end{aligned}$$

Secretary problem

- Analyzing the probability of success



$$\begin{aligned}\mathbb{P}(\text{hire } j\text{-th and succeed}) &= \mathbb{P}(c_j = 1) \times \mathbb{P}(t+1, \ldots, j-1 \text{ are not hired} \mid c_j = 1) \\ &= \frac{1}{n}\frac{t}{j-1}\end{aligned}$$

- The success probability is

$$\begin{aligned}\mathbb{P}(\text{we succeed}) &= \sum_{j=t+1}^{n} \mathbb{P}(\text{we hire } j\text{-th candidate and succeed}) \\ &= \sum_{j=t+1}^{n} \frac{1}{n}\frac{t}{j-1} \\ &\simeq \frac{t}{n}(\ln(n) - \ln(t))\end{aligned}$$

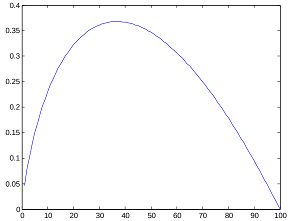
- We used $\ln(n+1) \leq \sum_{i=1}^{n} 1/i \leq 1 + \ln(n)$ and let $\sum_{i=1}^{n} 1/i \simeq \ln(n)$

Secretary problem

- optimal strategy
  - We analyzed the success probability of a strategy that waits until time t

$$\mathbb{P}(\text{success}) = \frac{t}{n}(\ln(n) - \ln(t))$$

  - we want to maximize this probability



  - taking the derivative, (and assuming n to be continuous for simplicity) the success probability is maximized at $t = n/e$
  - the success probability is $1/e$
  - we can make this argument rigorous by maximizing over integer valued n, but more complicated

Paging and caching

- Problem
  - We have two memories:
    - A larger but slower *hard disk*

address	1	2	...	...	n
content	x_1	x_2	...	...	x_n

    - A faster but smaller *RAM*

	1	2	...	k
pointer	i_1	i_2	...	i_k
copied content	x_{i_1}	x_{i_2}	...	x_{i_k}

  - input: sequence of requests to hard disk

$$31, 3, 12, 3, 3, 5, 3, 3, 5, 3, 21, 5, 35, 12, 4, 6, 3, 1, 12, 4, 12, 3$$

  - action:
    - each time a request comes in, we first look up in the RAM whether we have the content of that address stored in the RAM
    - If we have a *hit* accessing RAM takes negligible time
    - If we have a *miss* we need to fetch the entry from the hard disk
    - At no extra cost (time), we (always) store the fetched data in RAM
    - If the RAM is full, we need to delete one of the entries already stored to make room
    - decision: which one should we delete?

Paging and caching

- cost: total number of misses (which is the only non-negligible time)
- optimal off-line algorithm minimizing the total number of misses
  - **Belady's MIN algorithm**
  - Elegant proof using dynamic programming
  - Each time we have a miss, swap out the entry whose next use will occur farthest in the future
  - Example with $k = 3$

requests	31	3	12	3	5	3	5	21	5	31	12	3
RAM	31	31	31	31	31	31	31	31	31	31	31	31
		3	3	3	3	3	3	21	21	21	21	21
			12	12	5	5	5	5	5	5	12	3
miss	o	o	o		o			o			o	o

  - Problem: in practice, requests arrive in an online fashion, and we cannot predict future requests

Paging and caching

- A practical online algorithm
  - **Least Recently Used (LRU) heuristic**
  - Use how much time has passed since last use in the past as an approximation of how soon it will be used in the future

requests	31	3	12	3	5	3	5	21	5	31	12	3
MIN	31	31	31	31	31	31	31	31	31	31	31	31
RAM		3	3	3	3	3	3	21	21	21	21	21
			12	12	5	5	5	5	5	5	12	3
miss	o	o	o		o			o			o	o
LRU	31	31	31	31	5	5	5	5	5	5	5	3
RAM		3	3	3	3	3	3	3	3	31	31	31
			12	12	12	12	12	21	21	21	12	12
miss	o	o	o		o			o		o	o	o

  - **Claim.** For any sequence of requests, let m be the number of misses occurred using the best off-line algorithm with size-k RAM. Then, for the same sequence of requests, LRU with a size-k RAM causes at most km misses.
  - Therefore, competitive ratio is at most k

Paging and caching

- **Proof.**
  - For the proof, we divide the input sequence $i_1, \ldots, i_m$ into phases as follows.
  - Let t be the first time at which we see the $(k+1)$-th new request.
  - Then the first phase is $i_1, \ldots, i_{t-1}$.
  - Next, consider remaining sequence $i_t, \ldots, i_m$ and recursively divide it into phases.
  - For example, if $k = 3$ then the input sequence can be divided into phases as

$$31, 3, 12, 3, 5, 3, 3, 21, 3, 31, 12, 3$$

$$\underbrace{[31, 3, 12, 3]}_{\text{Phase 1}},\ \underbrace{[5, 3, 3, 21, 3]}_{\text{Phase 2}},\ \underbrace{[31, 12, 3]}_{\text{Phase 3}}$$

  - In each phase, LRU algorithm can cause at most k misses, since there are only k distinct requests in each phase
  - Let us not worry about the last phase which might not have k distinct requests, since the effect of the boundary dies out as the sequence gets longer

Paging and caching

- **Proof.** (continued.)
  - Now, move the first item in each phase to the previous phase

    $$[31, 3, 12, 3, \mathbf{5}],\ [3, 3, 21, 3, \mathbf{31}],\ [12, 3]$$

    - ★ each phase now has either k or $k+1$ distinct requests (except for the last phase)
  - We claim that at each phase the MIN algorithm misses at least one request
    - ★ Case 1: a phase has $k+1$ distinct requests
      Then MIN algorithm must miss at least one request, since the RAM size is only k
    - ★ Case 2: a phase has k distinct requests
      This only happens if the last request of the previous phase is not repeated in current phase
      But, the last entry of the previous phase must be stored in current RAM at the beginning of current phase (at the beginning of Phase 2, x_5 which is the content of HARD DISK at address 5 must be stored in RAM). Even in the best case, only $k-1$ distinct requests of Phase 2 can be read from RAM, and at least *one* request has to be fetched from HARD DISK

Paging and caching

- Finally, we know that
    1. LRU causes at most $k \cdot p$ misses, where p is the total number of phases
    2. MIN causes at least p misses
- Therefore, the competitive ratio is at most k for LRU.

Expert advice

- Problem

time	1	2	...	t	...	T
event of interest	$+1$	-1	...	-1	...	$+1$
expert 1	$+1$	-1	...	-1	...	$+1$
expert 2	-1	-1	...	-1	...	-1
expert 3	$+1$	$+1$	...	$+1$	...	$+1$
$\vdots$			$\vdots$		$\vdots$	
expert n	$+1$	$+1$	...	-1	...	-1
prediction	$+1$	-1	...	-1		

  - at time t, make prediction of the "event of interest" at time t based on experts' advice on the event, and their past history
  - once we make a prediction, the true event of interest at time t is revealed immediately
  - Optimal offline algorithm: that chooses *one* expert that makes the smallest number of mistakes (knowing the ground truths)
  - Online algorithm: multiplicative weight

Expert advice

- Multiplicative weight
  - assigns a weight w_i^t to expert i at time t
  - initially: $w_i^0 = 1$
  - update rule: divide by 2 the weight of the experts that were wrong
  - prediction:
    - ★ let x_i^t be the expert i's opinion at time t
    - ★ let $w_+^t = \sum_{i:x_i^t=+} w_i^t$ be the summed weight of experts who said $+$ and $w_-^t = \sum_{i:x_i^t=-} w_i^t$ be the sum of weights of experts who said $-$
    - ★ Predict $+$ if $w_+^t \geq w_-^t$, and $-$ otherwise

time	1	2	3	...	T
event of interest	$+1$	-1	-1	...	$+1$
expert 1	$+1(1)$	$-1(1)$	$-1(1)$	...	$+1$
expert 2	$-1(1)$	$-1(1/2)$	$-1(1/2)$	...	-1
expert 3	$+1(1)$	$+1(1)$	$+1(1/2)$	...	$+1$
expert 4	$+1(1)$	$+1(1)$	$-1(1/2)$	...	-1
prediction	$+1$	$+1$	-1		

Expert advice

- **Definitions.**
  - Let b^t be the ground truth event of interest at time t
  - Define an indicator for mistakes $m_i^t = \begin{cases} 0 & \text{if } x_i^t = b^t \\ 1 & \text{if } x_i^t \neq b^t \end{cases}$
  - Let $m_i = \sum_{t=1}^{T} m_i^t$ be the total number of mistakes by expert i
  - Let m_A^t be the indicator that our algorithm makes a mistake at time t
  - Let $m_A = \sum_{t=1}^{T} m_A^t$ be the total number of mistakes by *multiplicative weight*
  - Let $w^t = \sum_{i=1}^{n} w_i^t$ be the total weight at time t
- **Theorem.** $m_A \leq \frac{1}{\log_2(4/3)}(m_i + \log n)$, for every i (including the best expert).
- This shows that the number of mistakes is a constant times the mistakes of the best expert, plus an extra logarithmic term

Expert advice

- **Proof.**
  - We claim that if we make a mistake at time t, the weight decreases as
$$w^{t+1} \leq (3/4)w^t$$
    - ★ If we make a mistake at t (and let's assume $b^T = +1$), then $w_-^t \geq w_+^t$.
    - ★ Then, $w_-^t \geq (1/2)w^t$ and $w_+^t \leq (1/2)w^t$
    - ★ The weights w_-^t will be decreased in the following step by 1/2
    - ★ $w^{t+1} = w_+^t + (1/2)w_-^t = (1/2)w^t + (1/2)w_+^t \leq (3/4)w^t$
  - By recursion,
$$w^{T+1} \leq \prod_{t=1}^{T} \left(\frac{3}{4}\right)^{m_A^t} w^0 \leq n\left(\frac{3}{4}\right)^{m_A}$$
  - For each expert i, the final weight is 2^{-m_i}, and
$$\frac{1}{2^{m_i}} = w_i^{T+1} \leq w^{T+1}$$
  - Putting these together, we get for every expert i
$$\frac{1}{2^{m_i}} \leq n\left(\frac{3}{4}\right)^{m_A}$$
  - This proves $m_A \leq \frac{1}{\log_2(4/3)}(m_i + \log n)$

Expert advice

- A mixed strategy
  - Action:
    - choose p_i^t
    - at time t follow expert i's advice with probability p_i^t
  - Loss:
    - $m_i^t \in [-1, 1]$ is the loss incurred when following expert i
    - loss is revealed after our action
    - negative loss means *gain*
  - Expected loss: $\sum_{i=1}^n m_i^t p_i^t$
  - Want to minimize the total expected loss

$$\sum_{i=1}^{m} \sum_{t=1}^{T} m_i^t p_i^t$$

  - compare it to the best expert

$$\min_{i \in \{1,\dots,n\}} \sum_{t=1}^{T} m_i^t$$

  - General model: general event (not binary), general loss (cf. mistake/no mistake).
  - Idea: track weight corresponding to the reliability or confidence

Expert advice

- Multiplicative weight (for general model)
  - assigns a weight w_i^t to expert i at time t
  - initially: $w_i^0 = 1$
  - update rule: for some $\epsilon \in (0, 1/2)$,

$$w_i^{t+1} = (1 - \epsilon m_i^t) w_i^t$$

  - action:
    - ★ Let $w^t = \sum_{i=1}^n w_i^t$
    - ★ Choose expert i's advice with probability

$$p_i^t = \frac{w_i^t}{w^t}$$

Expert advice

- Analysis of multiplicative weight
  - Let the expected loss be $m_A^t = \sum_{i=1}^n m_i^t p_i^t$
  - And the total expected loss be $m_A = \sum_{t=1}^T m_A^t$
  - Also, $m_i = \sum_{t=1}^T m_i^t$
- **Theorem.** We can get arbitrarily close to the best expert. Precisely, for any i,

$$m_A \le m_i + \epsilon \sum_{i=1}^{T} |m_i^t| + \frac{\ln n}{\epsilon}$$

- Choosing the value of ϵ as a function of known parameters T and n (m_i's are unknown)
  - Since $m_i \le T$,

$$m_A \le m_i + \epsilon T + \frac{\ln n}{\epsilon}$$

  - The right-hand side is minimum when $T - \ln n/\epsilon^2$.
  - Setting $\epsilon = \sqrt{\ln n/T}$.

$$m_A \le m_i + 2\sqrt{T \ln n}$$

  - For the choice of ϵ, we get an additive error, growing with T and $\ln n$

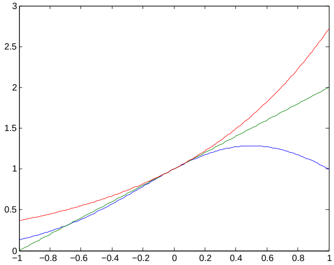
Expert advice

- **Lemma.** For all $\epsilon \in [-1/2, 1/2]$,

$$e^{\epsilon-\epsilon^2} \leq 1+\epsilon \leq e^{\epsilon}$$

  - **Proof.**
  - First, plot on MATLAB.

```
e=[-1:0.01:1];
plot(e,exp(e-e.∧2),e,1+e,e,exp(e));
```



Expert advice

- **Proof.** Upper bound: $1 + \epsilon \leq e^{\epsilon}$
  - Claim $f(\epsilon) = e^{\epsilon} - 1 - \epsilon \geq 0$ for all ϵ.
    - ★ First, notice that $f(\epsilon)$ is a convex function, since
    $$f''(\epsilon) = e^{\epsilon} \geq 0$$
    - ★ Next, since it is a convex function, we can find the global minimum value of the function by setting the derivative to zero.
    $$f'(\epsilon) = e^{\epsilon} - 1 = 0$$
    Hence, the function $f(\epsilon)$ is minimum at $\epsilon = 0$ with $f(0) = 0$
    - ★ Therefore, the function is a non-negative function
  - Convexity of a twice differentiable function: the following are equivalent
    - ★ $f(x)$ is convex
    - ★ $f(ax + (1-a)y) \leq af(x) + (1-a)f(y)$ for all $a \in [0,1]$ and all x, y
    - ★ $f(x) + f'(x)(y - x) \leq f(y)$ for all x, y
    - ★ $f''(x) \geq 0$ for all x

Expert advice

- **Proof.** Lower bound: $e^{\epsilon-\epsilon^2} \leq 1+\epsilon$
  - We first show it for $\epsilon \in [0,1]$.
    - ★ Similarly as before, we can show that
    $$e^x \leq 1 + x + x^2$$
    for $x \in [0, \ln 2]$
    - ★ Let $f(x) = 1 + x + x^2 - e^x$
    $f''(x) = 2 - e^x \geq 0$ implies the function is convex in $x \in [0, \ln 2]$
    $f'(0) = 0$ implies the function is minimum at $x = 0$ with $f(0) = 0$
    Hence, the function is non-negative
    - ★ Then, for $\epsilon \in [0,1]$,
    $$e^{\epsilon-\epsilon^2} \leq 1 + \epsilon - \epsilon^2 + \epsilon^2 - 2\epsilon^3 + \epsilon^4 \leq 1 + \epsilon$$

Expert advice

- **Proof.** Lower bound: $e^{\epsilon-\epsilon^2} \leq 1+\epsilon$
  - Next, we show it for $\epsilon \in [-1/2, 0]$.
    - ★ Similarly, we can show that
      $$e^x \leq 1 + x + (1/2)x^2$$
      for $x \in [-1, 0]$
    - ★ Let $f(x) = 1 + x + (1/2)x^2 - e^x$
      $f''(x) = 1 - e^x \geq 0$ implies the function is convex in $x \in [-1, 0]$
      $f'(0) = 0$ implies the function is minimum at $x = 0$ with $f(0) = 0$
      Hence, the function is non-negative
    - ★ Then, for $\epsilon \in [-1/2, 0]$,
      $$e^{\epsilon-\epsilon^2} \leq 1 + \epsilon - \epsilon^2 + \frac{1}{2}\epsilon^2 - \epsilon^3 + \frac{1}{4}\epsilon^4 \leq 1 + \epsilon$$

Expert advice

- **Proof of the main theorem**
  - For each expert i, the weight of that expert at the end is

$$w_i^{T+1} = \prod_{i=1}^{T} (1 - \epsilon m_i^t)$$

  - We claim that the total weight in the end is

$$w^{T+1} = n \cdot \prod_{t=1}^{T} (1 - \epsilon m_A^t) \tag{1}$$

  - As before, weight of one expert is upper bounded by the sum

$$w_i^{T+1} \leq w^{T+1}$$

  - Putting these together

$$\prod_{t=1}^{T} (1 - \epsilon m_i^t) \leq n \cdot \prod_{t=1}^{T} (1 - \epsilon m_A^t)$$

Expert advice

- **Proof of the main theorem**

$$\prod_{t=1}^{T}(1-\epsilon m_i^t) \le n \cdot \prod_{t=1}^{T}(1-\epsilon m_A^t)$$

$$e^{\epsilon-\epsilon^2} \le 1+\epsilon \le e^{\epsilon}$$

  - We get,

$$\prod_{t=1}^{T}(1-\epsilon m_A^t) \le \prod_{t=1}^{T} e^{-\epsilon m_A^t} \le e^{-\epsilon \sum_{t=1}^{T} m_A^t} \le e^{-\epsilon m_A}$$

$$\prod_{i=1}^{T}(1-\epsilon m_i^t) \ge \prod_{t=1}^{T} e^{-\epsilon m_i^t-(\epsilon m_i^t)^2} \ge e^{-\epsilon m_i-\epsilon^2 \sum_{t=1}^{T}(m_i^t)^2}$$

  - together, using $(m_i^t)^2 \le |m_i^t|$, we get

$$m_A \le m_i + \epsilon \sum_{t=1}^{T} |m_i^t| + \frac{\ln n}{\epsilon}$$

Expert advice

- Proof of claim (1): $w^{T+1} = n \cdot \prod_{t=1}^{T}(1 - \epsilon m_A^t)$
  - Recall that $m_A^t = \sum_{t=1}^{T} p_i^t m_i^t = \sum_{t=1}^{T} \frac{w_i^t}{w^t} m_i^t$

$$\begin{aligned}
w^{t+1} &= \sum_{i=1}^{n} w_i^{t+1} \\
&= \sum_{i=1}^{n} (1 - \epsilon m_i^t) w_i^t \\
&= w^t - \epsilon \sum_{i=1}^{n} m_i^t w_i^t \\
&= w^t - \epsilon w^t \sum_{i=1}^{n} m_i^t \frac{w_i^t}{w^t} \\
&= w^t - \epsilon w^t \sum_{i=1}^{n} m_i^t p_i^t \\
&= w^t - \epsilon w^t m_A^t \;=\; w^t(1 - \epsilon m_A^t)
\end{aligned}$$

Expert advice

- Proof of claim (1) continued
  - By recursion,

$$\begin{aligned} w^{T+1} &= w^{T}(1-\epsilon m_A^t) \\ &= w^{T-1}(1-\epsilon m_A^t)(1-\epsilon m_A^{t-1}) \\ &= w^0 \cdot \prod_{t=1}^{T}(1-\epsilon m_A^t) \end{aligned}$$

  - And $w^0 = n$.

Bin packing

- Data: list of n items $a_1, \ldots, a_n$ of size $a_i \in (0, 1]$.
- Goal: pack these items into minimal number of unit capacity bins
- Items arrive in on-line fashion
- You make assignments each time and cannot change bin-allocation afterwards

- Algorithms
  - Next fit algorithm
    - Start from the first bin, and call it *active*
    - If the next incoming fit the current active bin, then allocate that item in the current active bin
    - Otherwise, create a new bin and call it active
    - Only keep one active bin at a time
  - Best fit algorithm
    - Keep track of all bins
    - Allocate items in a bin that leaves smallest empty space in the bin

Bin packing

- Competitive analysis
  - Let $L = (a_1, \ldots, a_n)$.
  - $NFA(L)$ be the number of bins required by next fit algorithm on data L
  - $OPT(L)$ be the number of bins required by the optimal off-line algorithm on the same data L

$$\text{competitive ratio} = \frac{NFA(L)}{OPT(L)}$$

  - **Claim 1.** Competitive ratio ≤ 2.
  - **Proof 1.**
    - ★ Two adjacent bins must have items of size > 1
    - ★ If not, those items would have been assigned to the same bin.
    - ★ Hence, the total size of the items is $\sum_i a_i > \frac{1}{2} NFA(L)$
    - ★ But the number of bins must be at least $OPT(L) \geq \sum_i a_i$

Bin packing

- Competitive analysis
  - **Claim 2.** Competitive ratio ≥ 2
  - **Proof 2.**
    - ⋆ Consider a sequence of $2m$ items $\{1/2, 1/m, 1/2, 1/m, \ldots\}$
    - ⋆ The Next Fit Algorithm takes m bins
    - ⋆ The optimal offline algorithm uses $(1/2)m + 1$ bins

$$\text{Competitive ratio} = \frac{m}{m/2+1} = \frac{2}{1+2/m}$$

    - ⋆ Taking m large, we get that the competitive ratio can be made as close as to two as we want

Bin packing

- Competitive analysis of Best Fit Algorithm
  - **Claim 3.** Competitive ratio $= \frac{BFA(L)}{OPT(L)} \geq 5/3$
  - **Proof 3.**
    - Consider a sequence of $6m$ items of size 0.15, $6m$ items of size 0.34, and $6m$ items of size 0.51
    - We can pack these into $OPT(L) = 6m$ bins using the optimal offline algorithm
    - BFA requires $m + 3m + 6m = 10m$
    - Competitive ratio $\geq 5/3$

Homework 7

Problem 1.
Prove that Competitive Ratio of 2 can be always achieved for general buy-vs-rent problem using a deterministic algorithm. First explain the deterministic algorithm, and then prove that the competitive ratio is always bounded by two for general cost of buying (\$$B$) and renting (\$$R$).

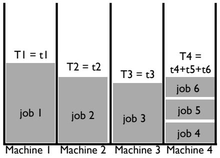
Homework 7

Problem 2.

We consider the following scheduling problem. There are M machines and J jobs. Each job j takes t_j time to finish independent of which machine is used. We want to come up with an assignment for each machine i. Let A_i be the set of jobs assigned to machine i. Then, the completion time for this assignment and this machine is $T_i = \sum_{j \in A_i} t_j$. We want to find assignments such that (i) each job is assigned to one of the machines, and (ii) the maximum completion time is minimized.

$$\text{minimize}_{\{A_i\}_{i \in \{1,\ldots,M\}}} \max_{i \in \{1,\ldots,M\}} T_i$$

We let this minimum value be $OPT(L)$, where $L = \{t_j\}_{j \in \{1,\ldots,J\}}$ is the list of completion times.



Now consider a *greedy* approach to solve this problem. First sort the jobs in an decreasing order such that $t_1 \geq t_2 \geq \ldots \geq t_J$. Then, the greedy algorithm iteratively assigns at iteration k, the current job k with completion time t_k to the machine with the smallest load at current time. Once we assign a job to a machine, we never change the assignment. In this sense you can think of it as an online algorithm, working on a sorted and online input data.

Let $GA(L)$ be the maximum completion time $\max_{i \in \{1,\ldots,M\}} T_i$ for the greedy algorithm. We want to prove that for any input L,

$$\frac{GA(L)}{OPT(L)} \leq \frac{3}{2} \,.$$

Prove this claim step by step in the following.

(a) If $J \leq M$, prove that the greedy algorithm is optimal.

$$GA(L) = OPT(L)$$

(b) Now for any J, let i be one of the machines that is assigned maximum load $GA(L)$ using the greedy algorithm. If i has only one job, then prove that the greedy algorithm is optimal.

$$GA(L) = OPT(L)$$

(c) Again for any J, let i be one of the machines that is assigned maximum load $GA(L)$ using the greedy algorithm. Let j be the last job assigned to machine i. Prove that

$$GA(L) - t_j \le \frac{1}{M} \sum_{i=1}^{M} T_i \ .$$

(d) If $J > M$, prove that, for any input L,

$$t_{M+1} \le \frac{1}{2} OPT(L) \ .$$

(e) Prove that even the optimal algorithm requires the maximum completion time to be at least the average completion time. Precisely, prove

$$\frac{1}{M} \sum_{i=1}^{M} T_i \le OPT(L)$$

(f) Using $(a), \cdots, (e)$, prove that

$$\frac{GA(L)}{OPT(L)} \le \frac{3}{2} \ .$$

(g) **(Optional)** Now, use similar ideas from above to prove that

$$\frac{GA(L)}{OPT(L)} \le \frac{4}{3} \ .$$