References:

Caching:
Irani, Sandy. "Competitive analysis of paging." *Online Algorithms*. Springer Berlin Heidelberg, 1998. 52-73.

Matching: last page of
Karp, Richard M., Umesh V. Vazirani, and Vijay V. Vazirani. "An optimal algorithm for on-line bipartite matching." *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. ACM, 1990.

Set Cover:
Korman, Simon. *On the use of randomization in the online set cover problem*. Diss. Weizmann Institute of Science, 2004.

* See lecture notes by Prof. Plotkin for CS369 at Stanford.

Introduction:

An online problem is one where not all the input is known at the beginning. The input consists of "requests" that arrive one by one. Upon the arrival of each request, we need to process as it is recieved.

Since the algorithm does not know the rest of the input, it may not be able to make optimum decisions.
How do we measure the performance? We compete with the best possible 'offline solution'. Hence, we use the notion of 'competitive ratio'.
Let OPT denote the cost of an optimal offline solution.
Alg := cost of the online algorithm
For minimization problems: $\forall \sigma$: $Alg(\sigma) \leq \boxed{\alpha} OPT(\sigma) + \gamma$
where γ is some constant independent of σ.
For maximization: $\forall \sigma$ $Alg(\sigma) \geq \frac{1}{\alpha} OPT(\sigma)$

An advantage of online problems when proving lower bounds:
You often can prove LBs without any hardness assumptions.

Problem 1: Paging

We have n number of pages in RAM.

A cache of k pages.

At online step i, a page σ_i is requested.

Our cost $\begin{cases} 0 & \text{if } \sigma_i \text{ is in cache} \\ 1 & \text{o.w. In which case we need to replace } \sigma_i \text{ with a page in cache.} \end{cases}$

fault occurs

Initialization: the same for every alg.

Decision: Which page to kick out?

LIFO Last In First Out FIFO, LFU

LRU: Least Recently Used

Quick detour: LRU is k-competitive.

A phase: an interval that LRU faults exactly k times.

We show that OPT faults at least once in a phase.

Two cases: Case 1: Same page kicked twice: $[\sigma_i \; \sigma_j]$ call it σ_k

In between i and j, the k pages have been requested together with σ_k and what replaces it, we have $k+1$

Case 2: k different pages. Let σ_k be the last page before the phase.

It is in both OPT and Alg. $\begin{cases} \text{LRU does not fault on } \sigma_k \Rightarrow \text{OPT has to fault} \\ \text{LRU does fault} \Rightarrow \text{Case 1.} \end{cases}$

LB for deterministic algorithms for paging

Let $n = k+1$.

A: an arbitrary det. Alg.

Input: always request the hole in the cache of A.

$\Rightarrow Cost_A = |\sigma|$

Opt: kick out the page that is going to be requested furthest in the future. $\Rightarrow Cost_{OPT} \leq \frac{|\sigma|}{k+1}$ since there are at least (k) request between a pair.

* Just so you know, randomization helps a lot!

Bipartite

Matching

A bipartite Graph $G = (U \uplus V, E)$. $U = \{u_1, \ldots, u_m\}$

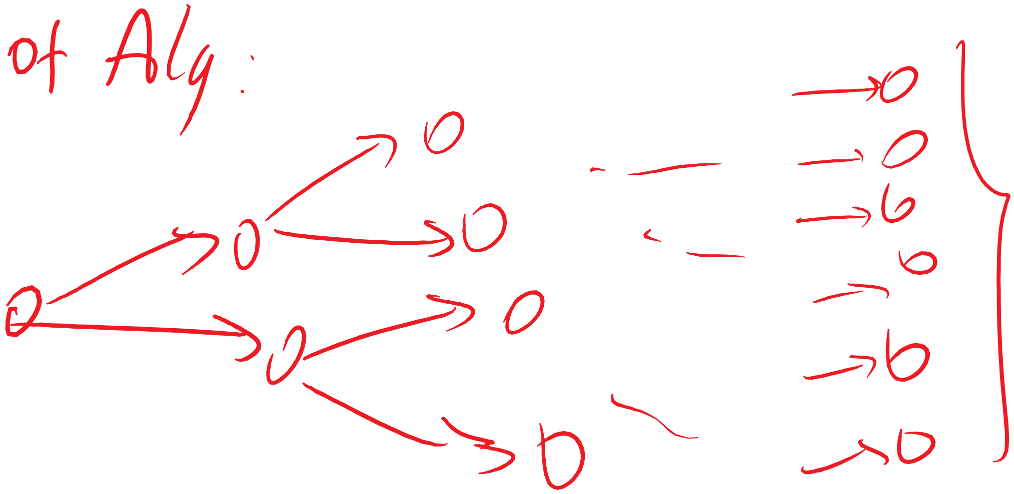
Find the maximum matching! $V = \{v_1, \ldots, v_n\}$

Online Scenario: V is known in advance (just |V|).
At online step i, u_i and its adjacent edges are revealed.

We quickly go over some simple algorithms.
Coming up with lower bounds is not only good for proving impossibility results, but also for shooting down algorithms.
It gives a very good intuition of what makes an instance hard.

How do we prove a LB for deterministic problems?

(i) We assume every Alg makes a sequence of decisions. Since Alg is deterministic, we can anticipate its decisions. We design our hard instance <u>BASED</u> on the deterministic decisions of Alg:



A sequence of possible hard instances for different algorithms.

(ii) Can we prove something stronger?

For (i), our 'hard instance' depends on the algorithm. Can we have a fixed instance that is hard for every deterministic algorithm?

NO: An oblivious hard instance doesn't exist: for every input, there is a perfect algorithm that just outputs OPT of that instance!

But: An oblivious distribution P over hard instances may exist!

S.t. for every Alg, $\mathbb{E}_{I \sim P}[\mathrm{Alg}(I)]$ is bad!

Such P, destroys every det. algorithm.

(iii) How about Randomized algorithms?
What can we do even assuming the knowledge of a randomized algorithm Rnd?
For every Rnd, we want an instance I, s.t. $\underset{\sim Rnd}{E}[Rnd(I)]$ is bad.

Yao's Lemma: Intuition

$\boxed{\text{(ii) is harder than (iii)}}$

Let S(P) denote the support of P. For every Rnd:
Worst $I \in S(P)$ for Rnd $\leq$ Best a det. Alg can do on P

more formally:

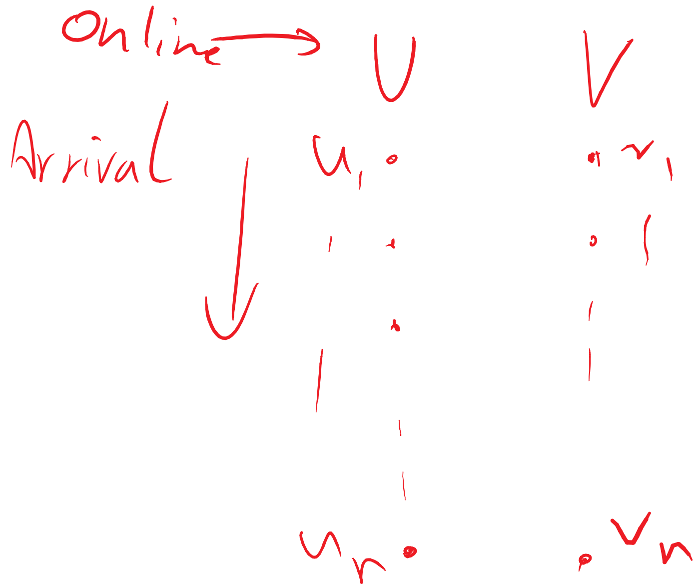
$$\forall Rnd, \min_{I \in S(P)} \{E[Rnd(I)]\}$$

$$\leq \max_{Al} \{ E[Alg(I)] \}$$

$$\leq \max_{Alg} \left\{ \mathbb{E}_{I \sim P} [Alg(I)] \right\}$$

* These were for Maximization!

Matching:



Instan:

$I := u_i \longleftrightarrow v_j \quad \forall j \geq i$

For every permutation $\pi : [n] \to [n]$

define $I(\pi) := u_i \longleftrightarrow v_{\pi_j} \quad \forall j \geq i$

Let P be a uniform dist. on permutations.

Claim: for every Rnd, $\exists \pi$, s.t. $\mathop{E}_{\sim Rnd}[Rnd(I(\pi))]$

$\leq n(1-\frac{1}{e}) + o(n)$

By Yao's lemma, we want to prove that

for any det. algorithm Alg, $\mathop{E}_{\pi \sim P}[Alg(I(\pi))] \leq$ "

We prove this in two steps:

(1) Let RANDOM denote the random-neighbor

algorithm. We show that $\forall$ Alg, $E[Alg(P)] \leq E[Random(P)]$

(2) We show that $E[Random(P)] \leq n(1-\frac{1}{e}) + o(n)$

Lemma (1)

$\forall$ Alg. $E[Alg(P)] \leq E[Random(I)]$

Consider an arbitrary iteration i, the set of 'eligible vertices' is $Q(i) := \{v_{\pi_j} \mid j \geq i\}$

We have by induction on i that:

(a) Suppose Alg or Random have k unmatched eligible vertices. Any two subsets of size k from $Q(i)$, are the unmatched eligible set withe the same probability

(b) $Pr(k, i)$, having exactly k unmatched eligible vertices at time i, is the same for $Alg(P)$ and $Random(I)$.

item(b) for $i = n+1$ leads to the lemma(1)

Thursday, November 6, 2014 2:24 PM

Lemma (2)

Consider the algorithm Random.

For iteration i, define the following two (random) variables:

$x(i) := n-i+1 = |Q(i)|$

$y(i) :=$ # unmatched eligible vertices

We have:

$\Delta x = -1$

$$\Delta y = \begin{cases} -2 & \text{if } v_{\pi_i} \text{ is unmatched AND } u_i \text{ will not be matched to it} \\ -1 & \text{o.w.} \end{cases}$$

By Lemma(1)-a, we have $\Pr[v_{\pi_i} \text{ is unmatched}] = \frac{y(i)}{|Q(i)|}$

and therefore

$$\Pr[y(i+1) - y(i) = -2] = \frac{y(i)}{x(i)} \cdot \frac{y(i)-1}{y(i)}$$

$\rightarrow E[\Delta y] = 1 \quad y(i)-1$

$x(i) \qquad y(i)$

$$\Rightarrow E[\Delta y] = -1 - \frac{y(i)-1}{x(i)}$$

$$\Rightarrow \frac{E[\Delta y]}{E[\Delta x]} = 1 + \frac{y(i)-1}{x(i)}$$

when $n \to \infty$ this can be closely approximated by the solution of differential equation

$$\frac{dy}{dx} = 1 + \frac{y-1}{x} \longrightarrow y(i) = c_1 x(i) + x(i)\log x(i) + 1$$

using $y(1) = x(1) = n$ $\qquad y(i) = 1 + x(i)\left(\frac{n-1}{n} + \ln\frac{x(i)}{n}\right)$

$$\Rightarrow y(n+1) = \frac{n}{e} - o(n)$$

Sunday, November 9, 2014 10:34 AM

Set Cover

Input: (E, F)

$E :=$ universe of elements

$F :=$ a collection of subsets of E

Goal: choose a minimum number of subsets in F such that every element is covered (at least once).

* It's not possible to approximate the solution better than $(1-\varepsilon)\log n$, unless NP can be solved in time $n^{O(\log \log n)}$

Online setting:

offline input: (U, F) where $|F| = n$

$U :=$ is the universe

online input: $E = \langle e_1, \dots, e_m \rangle \subseteq U$

At iteration i, e_i arrives and we should augment the solution so that $\{e_1 \dots e_i\}$ is covered.

Reduction from Set Cover to Online Set Cover

Let (E, F) be an offline hard instance with an optimal solution of size k, but any algorithm that runs in polynomial time, cannot compute a solution better than k' (we know $k' \in \Omega(\lg n)\, k$)

We will construct an online instance (U', F', E') with an optimal solution of size k s.t. an online algorithm with solution size better than $k' o(\lg n)$ requires solving (E, F) with a cost better than k'.

This implies a $\Omega(\lg^2 n)$-hardness for online algorithms that run in poly time.

We will shortly see that for a size N, we have

$|E| = m \quad |F| = n \quad |U'| = (N-1)m \quad |F'| = \frac{N}{2} n$

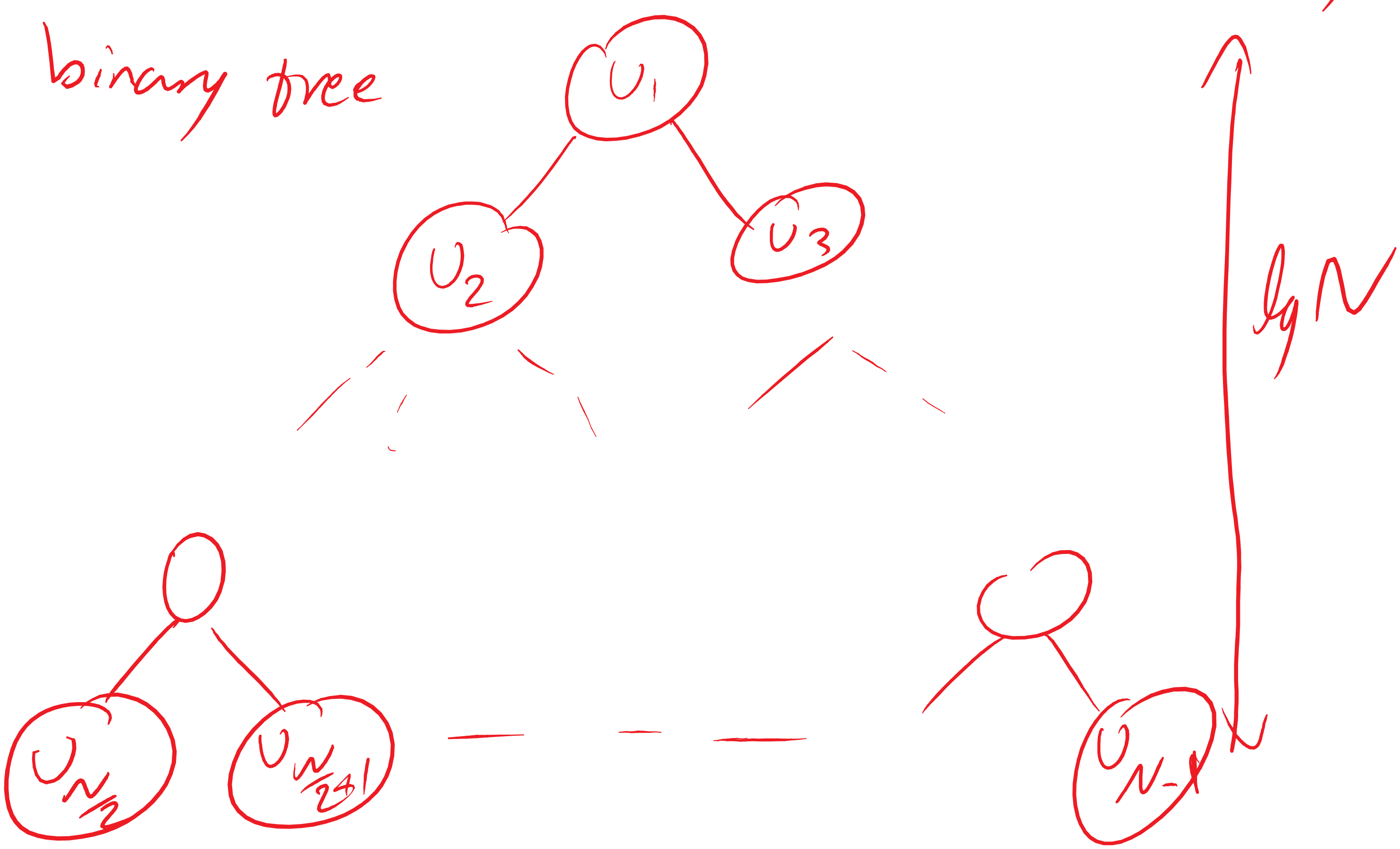
$|G'| = m \lg N$

Construction: Let N be a power of two.

$U' :=$ comprise $N-1$ copies of E:

U_1 — U_{N-1}

We depict these elements in the following binary tree



For every copy indexed at $i \in [1 .. N-1]$, we have a path from root to that copy.

Let the ordered vector of indices P_i denote the indices of the path from root to i.

$$\begin{cases} P_1 = \langle 1 \rangle \\ P_i = \langle P_{\lfloor i/2 \rfloor}, i \rangle \end{cases}$$

Constructing the sets:

Intuitively, every leaf $i \in [\frac{N}{2} .. N-1]$, has a copy of offline subsets F, that covers the same set of elements from $U_i, U_{\lfloor i/2 \rfloor}, \ldots U_1$.

For every $f \in F$ and $i \in [1 .. N-1]$, let $U_i(f)$ denote the elements of U_i that correspond to the copies of f.

For $\forall i \in [\frac{N}{2} .. N-1]$, let F_i denote a copy of F s.t.

$$\forall f \in F, \quad f_i := \left\{ j \in [1, \ldots, \lg N] \;\middle|\; U_{P_i(j)}(f) \right\}$$

For every leaf i, we construct an online sequence E_i' as follow:

$$E_i' = \langle U_{\underbrace{P_i(1)}_{=1}}, U_{P_i(2)}, \dots, U_{\underbrace{P_i(\lg N)}_{=i}} \rangle$$

By Yao's lemma, it is sufficient to show that if we pick a leaf i uniformly at random, then every online det. algorithm that runs in poly-time, uses $\Omega(\lg n k')$ sets to cover the online requests E_i'.

Consider an arbitrary step $j \in [1..\lg N]$ in which we receive $U_{P_i(j)}$. Consider the subtree T rooted at $P_i(j)$. Only the sets that are in the leaves of T can be useful. Thus (wlog.) we may assume that the online algorithm has to choose at least k' sets among these sets to cover $U_{p_i(j)}$. let k_1 denote the # of selected sets in the left subtree of T, while k_2 denote # of those at right: $k_1+k_2 \geq k'$

$$\Rightarrow \max(k_1, k_2) \geq \frac{k'}{2}$$

w.l.o.g. suppose $k_2 \geq k_1$. With prob. $\frac{1}{2}$ the next U-node might go to the left subtree. hence all the sets that contribute to k_2 will be redundant. Therefore the online solution wastes at least $\frac{k'}{2}$ sets w.p. $\frac{1}{2}$.

$$\Rightarrow E[\text{size of an online solution}] \geq \frac{\lg N}{2} \times \frac{k'}{2}$$

as desired.

Note that opt is still $\underline{k}$. we can simply choose the optimal solution at the final leaf.