References:

Caching:  

Matching: last page of  

Set Cover:  
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 it as it is received.

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 $\text{OPT}$ denote the cost of an optimal offline solution.

$\text{Alg} := \text{cost of the online algorithm}$.

For minimization problems: $\forall \delta: \text{Alg}(\delta) \leq (1 + \delta) \cdot \text{OPT}(\delta) + \delta$

where $\delta$ is some constant independent of $\delta$.

For maximization: $\forall \delta: \text{Alg}(\delta) \geq \frac{1}{2} \cdot \text{OPT}(\delta)$.

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 \( o_i \) is requested.

Our cost:

\[
\begin{cases} 
0 & \text{if } o_i \text{ is in cache} \\
1 & \text{otherwise}
\end{cases}
\]

Fault occurs \( \leq 1 \) o.w. In which case we need to replace \( o_i \) with a page in cache.

Initialization: the same for every alg.

Decision: Which page to kick out?

LIFO: Last In First Out
FIFO: First In First Out
LRU: Least Recently Used

Quick detour: LRU is \( k \)-competitive.

A phase: an interval that LRU samples exactly \( k \) times.

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

Two cases:

Case 1: Same page twice.

In between \( i \) and \( j \), the \( k \) pages have been requested together with \( o_i \) and what replaces \( o_i \) becomes \( o_j \).

Case 2: \( k \) different pages. Let \( o_k \) be the last page before the phase.

It is in both OPT and alg. \( o_k \).

\( \text{LRU does not fault on } o_k \Rightarrow \text{OPT has to fault} \)

\( \text{LRU does not fault} \Rightarrow \text{Case 1} \).
LB for deterministic algorithms for paging

Let \( n = k + 1 \).

A: an arbitrary det. Alg.

Input: always request the hole in the cache of \( A \).

\[ \Rightarrow \text{Cost}_A = 1 \delta \]

Opt: kick out the page that is going to be requested furthest in the future. \( \Rightarrow \text{Cost}_O \leq 1 \delta \) since there are at least \( (k+1) \) requests between a pair \( k+1 \)

* Just so you know, randomization helps a lot!

---

**Bipartite Matching**

A bipartite graph \( G = (U \cup V, E) \).

\[ U = \{ u_1, \ldots, u_m \} \]

\[ V = \{ v_1, \ldots, v_n \} \]

Find the maximum matching.

**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 based 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 $\text{OPT}$ of that instance!

**But:** An oblivious distribution $\mathcal{P}$ over hard instances may exist! So for every Alg, $\mathbb{E}_I[\text{Alg}(I)]$ is bad!

Such $\mathcal{P}$ destroys every deterministic 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. \( \mathbb{E}_{\text{Rnd}}[\text{Rnd}(I)] \) is bad.

Yao's Lemma 2: Intuition

Let \( S(P) \) denote the support of \( P \). For every Rnd:

Worst \( I \in S(P) \) for \( \text{Rnd} \leq \text{Best} \) a det. Alg.

more formally:

\[
\forall \text{Rnd}, \min_{I \in S(P)} \{ \mathbb{E}[^{\text{Rnd}(I)}] \} \leq \max_{I} \{ \mathbb{E}[^{\text{Alg}(I)}] \}
\]
\[
\leq \max \left\{ \mathbb{E} \left[ \text{Alg}(C) \right] \right\}
\]

* These were for maximization!
Matching:
Instab:

\[ I := u_i \leftrightarrow v_j \quad \forall j \geq i \]

For every permutation \( \pi : \{n\} \rightarrow \{n\} \)

define

\[ I(\pi) := u_i \leftrightarrow v_{\pi(i)} \quad \forall j \geq i \]

let \( P \) be a uniform distribution on permutations.

Claim: For every \( \text{Rand, } \pi \), s.t. \( E[\text{Rand}(I(\pi))] \)

\[ \leq n \left(1 - \frac{1}{e}\right) + o(n) \]

By Yao's lemma, we want to prove that

for any det. algorithm \( \text{Alg} \), \( E[\text{Alg}(I(\pi))] \leq n \pi \sim P \)

We prove this in two steps:

1. Let \( \text{RAN} \text{DON} \) denote the random-neighbor
algorithm. We show that \( \forall \text{Alg}, \ E[\text{Alg}(P)] \leq E[\text{Random}(P)] \).

(2) We show that \( E[\text{Random}(S)] \leq n(1 - \frac{1}{e})^{\Omega(n)} \).
Lemma (1)

\[ \forall \text{Alg. } E[\text{Alg}(P)] \leq E[\text{Random}(I)] \]

Consider an arbitrary iteration \( i \), the set of eligible vertices is \( Q(i) = \{ v_i^j : j \geq i \} \).

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 with the same probability.

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

Item (b) for \( i = n+1 \) leads to the lemma (1).
Lemma (2)
Consider the algorithm Random.
For iteration i, define the following two (random) variables:
\[ x(i) := n - i + 1 = |Q(i)| \]
\[ y(i) := \# \text{ unmatched eligible vertices} \]
We have:
\[ \Delta x = -1 \]
\[ \Delta y = \begin{cases} -2 & \text{if } v_{mi} \text{ is unmatched AND } v_{ui} \text{ will not be matched to } v_{mi} \\ \min(0, y(i)) & \text{otherwise} \end{cases} \]
By Lemma (1) - a, we have \( Pr \left[ v_{mi} \text{ is unmatched} \right] = \frac{y(i)}{x(i)} \)
and therefore
\[ Pr \left[ y_{ui} = -2 \right] = \frac{y(i)}{x(i)} \cdot \frac{y(i) - 1}{y(i)} \]
\[ \Rightarrow \sum_{i=1}^{n} y(i) = 1 \cdot y(i) - 1 \]
\[
E(\Delta y) = -1 - \frac{y(i) - 1}{\nu(i)}
\]

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

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

\[
\frac{dy}{d\nu} = 1 + \frac{y - 1}{\nu} \implies y(i) = c + \nu(i) + \nu(i)\log\nu(i) + 1
\]

Using $y(1) = \nu(1) = 1$

\[
y(i) = 1 + \nu(i) \left( \frac{n - 1}{n} + \ln\frac{\nu(i)}{n} \right)
\]

\[
\implies y(n+1) = \frac{n}{e} - \alpha n
\]
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 \(O(n^{1-\varepsilon})\).
Online setting:

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

\(U\) is the universe

online input: \(E = \langle e_1, \ldots, e_m \rangle \subseteq U\)

At iteration \(i\), \(e_i\) arrives and we should augment the solution so that \(\{e_i, -e_i\}\) 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 $k'(\text{we know } k' \in \mathcal{O}(\log n) \cdot k)$.

We will construct an online instance $(U', E', F')$ with an optimal solution of size $k'$ s.t. an online algorithm with solution size better than $k' \cdot \log n$ requires solving $(E,F)$ with a cost better than $k'$. 
This implies a $\Omega(d^2n)$-hardness for online algorithms that run in polytime.

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

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

$|E'| = m \log N$
Construction! Let \( N \) be a power of two.

\( U' \) := comprise \( N-1 \) copies of \( E \):

\[
U_1 \quad \cdots \quad U_{N-1}
\]

We depict these elements in the following binary tree:

For every copy labeled 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 $i$.

$P_i = \langle 1 \rangle$

$P_i = \langle P_{i/2}, i \rangle$

Constructing the sets:

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

For every $f \in F$ and $i \in [N/2, N-1]$, let $U_i(f)$ denote the elements of $U_i$ that correspond to the copies of $f$.

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

$\forall f \in F, F_i := \{ j \in [1, \ldots, 2\log N] | U_{P_i(j)}(f) \}$
for every leaf \( i \), we construct an online sequence \( E_i \) as follow:

\[
E_i = \left< U_{p_i(1)}, U_{p_i(2)}, \ldots, U_{p_i(\log n)} \right>
\]

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 \( \Omega(\log n) \) sets to cover the online requests \( E_i \).

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

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

Wlog, suppose $k_2 \geq k_1$. With prob. $\frac{1}{2}$ the root

Undelete 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. $\frac{1}{2}$.

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

as desired.

Note that opt is still $k$, we can simply choose the optimal solution at the final leaf.