Advanced Algorithms

On-line Algorithms

Introduction

Online Algorithms are algorithms that need to make decisions without full knowledge of the input. They have full knowledge of the past but no (or partial) knowledge of the future.

For this type of problem we will attempt to design algorithms that are competitive with the optimum offline algorithm, the algorithm that has perfect knowledge of the future.

The Ski-Rental Problem

- Assume that you are taking ski lessons. After each lesson you decide (depending on how much you enjoy it, and if you broke your leg or not) whether to continue skiing or to stop forever.

- You have the choice of either renting skis for 1$ a time or buying skis for y$.

- Will you buy or rent?

The Ski-Rental Problem

- If you knew in advance how many times t you would ski in your life then the choice of whether to rent or buy is simple. If you will ski more than y times then buy before you start, otherwise always rent.

- The cost of this algorithm is min(t, y).

- This type of strategy, with perfect knowledge of the future, is known as an offline strategy.

The Ski-Rental Problem

- In practice, you don't know how many times you will ski. What should you do?

- An online strategy will be a number k such that after renting k-1 times you will buy skis (just before your kth visit).

- Claim: Setting k = y guarantees that you never pay more than twice the cost of the offline strategy.

- Example: Assume y=7$ Thus, after 6 rents, you buy. Your total payment: 6+7=13$.

The Ski-Rental Problem

Theorem: Setting k = y guarantees that you never pay more than twice the cost of the offline strategy.

Proof: when you buy skis in your k^{th} visit, even if you quit right after this time, $t \geq y$.

- Your total payment is k-1+y =2y-1.
- The offline cost is min(t, y) = y.
- The ratio is (2y-1)/y = 2-1/y.   ❀

We say that this strategy is (2-1/y)competitive.

The Ski-Rental Problem

Is there a better strategy?

- Let k be any strategy (buy after k-1 rents).

- Suppose you buy the skis at the kth time and then break your leg and never ski again.

- Your total ski cost is k-1+y and the optimum offline cost is min(k,y).

- For every k, the ratio (k-1+y)/min(k,y) is at least (2-1/y)

- Therefore, every strategy is at least (2-1/y)- - competitive. ❀

The Ski-Rental Problem

The general rule:

When balancing small incremental costs against a big onetime cost, you want to delay spending the big cost until you have accumulated roughly the same amount in small costs.

Competitive analysis

For a minimization problem, an online algorithm A is k-competitive if

$$C_A \leq k \cdot C_{OPT} + b \text{ for some constant b}$$
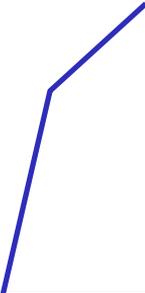
Cost of solution produced by algorithm A
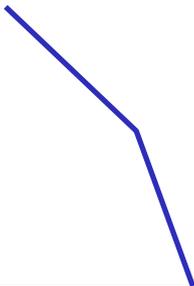
Cost of solution produced by optimal offline algorithm

The Hole in the Fence Problem

A cow wants to escape from Old McDonald's farm. It knows that there is a hole in the fence, but doesn't know on which direction it is.

Suggest the cow a path that will guarantee its freedom.
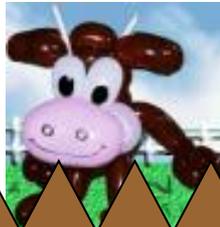
The Hole in the Fence Problem

Cow's algorithm:

1. d=1; current side = right

2. repeat:
   i. Walk distance d on current side
   ii. if find hole then exit
   iii. else return to starting point
   iv. d = 2d
   v. Flip current side
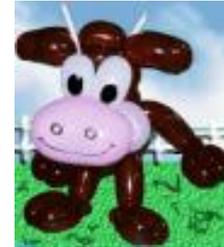
The Hole in the Fence Problem

Theorem: The cow's algorithm is 9-competitive.

In other words: The distance the cow might pass before finding the hole is at most 9 times the distance of an optimal offline algorithm (that knows where the hole is).

opt

alg
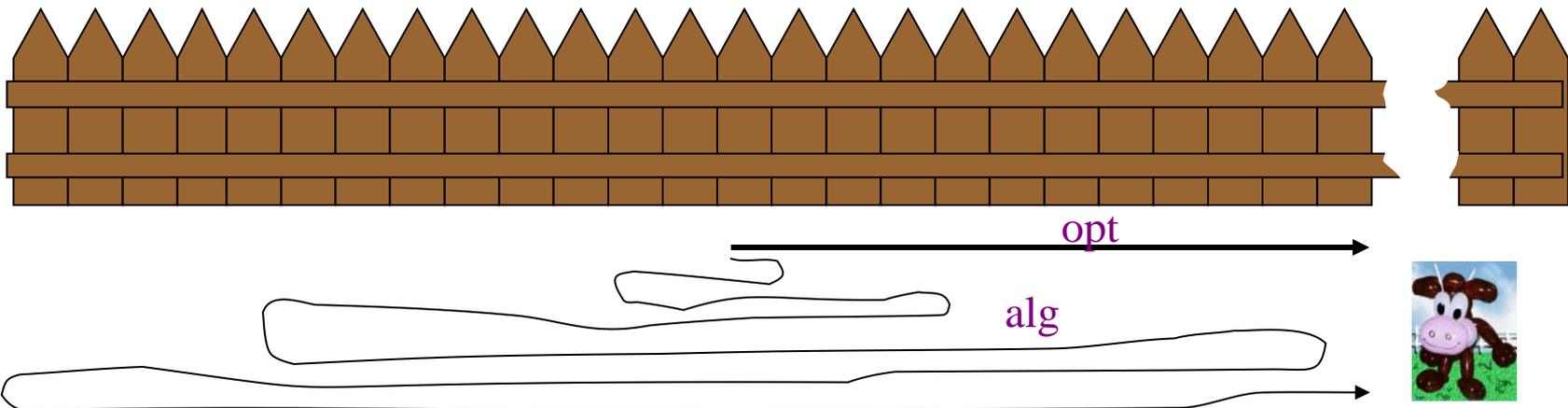
The Hole in the Fence Problem

Theorem: The cow's algorithm is 9-competitive.

Proof: The worst case is that it finds the hole a little bit beyond the distance it last searched on this side (why?).

Thus, OPT = $2^j + \varepsilon$ where j = # of iterations and ε is some small distance. Then,

Cost OPT = $2^j + \varepsilon > 2^j$

Cost COW = $2(1 + 2 + 4 + \ldots + 2^{j+1}) + 2^j + \varepsilon$

$\qquad\qquad = 2 \cdot 2^{j+2} + 2^j + \varepsilon = 9 \cdot 2^j + \varepsilon < 9 \cdot$ Cost OPT.

Edge Coloring

- An Edge-coloring of a graph $G=(V,E)$ is an assignment, c, of integers to the edges such that if e_1 and e_2 share an endpoint then $c(e_1) \neq c(e_2)$.
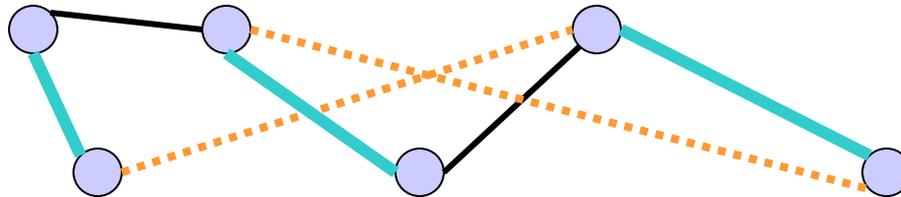
- Let Δ be the maximal degree of some vertex in G.

- In the offline case, it is possible to edge-color G using Δ or $\Delta+1$ colors (which is almost optimal).

- Online edge coloring: The graph is not known in advance. In each step a new edge is added and we need to color it before the next edge is known.

Optimal Online Algorithm for Edge Coloring

– We color the edges with numbers 1,2,3…

- Let $e=(u,v)$ be a new edge.

Color e with the smallest color which is not used by any edge adjacent to u or v.

Claim: The algorithm uses at most $2\Delta-1$ colors.

Proof sketch: assume we need the color 2Δ. It must be that all the colors $1,2,…,2\Delta-1$ are used by edges adjacent to u or v. Therefore, either u or v has Δ adjacent edges, excluding e, contradicting the definition of Δ.

u —— v
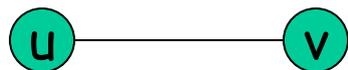
Online Edge Coloring

Claim: Any deterministic algorithm needs at least $2\Delta\text{-}1$ colors.

Proof: Assume $\exists$ an algorithm that uses only $2\Delta\text{-}2$ colors. Given Δ we add to the graph many $(\Delta\text{-}1)$-stars.

Example:
$\Delta=6$
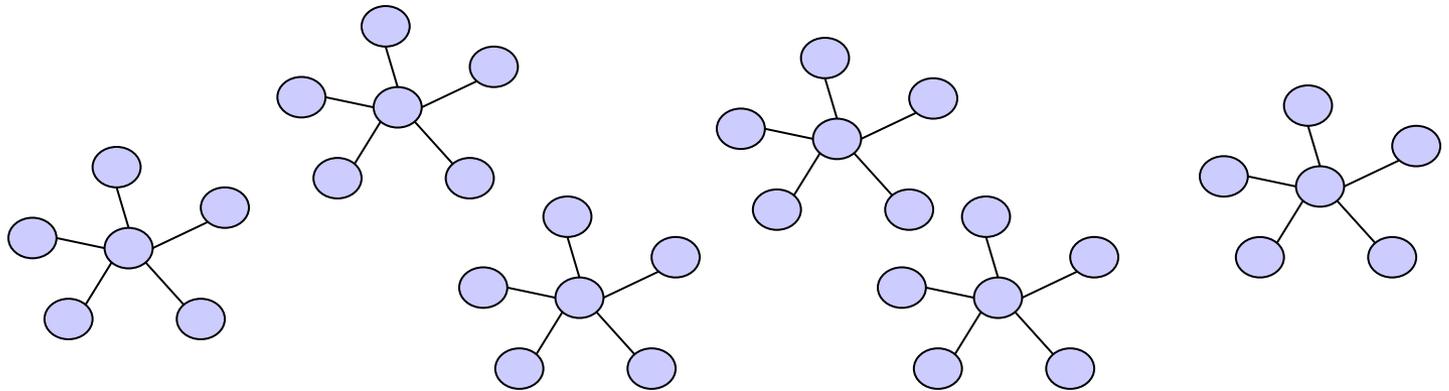
There is a finite number of ways to edge-color a $(\Delta\text{-}1)$-star with colors from $\{1,2,\ldots,2\Delta\text{-}2\}$, so at some point we must have Δ stars, all colored with the the same set of $\Delta\text{-}1$ colors.

Online Edge Coloring

Δ stars, all colored with the the same set of Δ-1 colors.
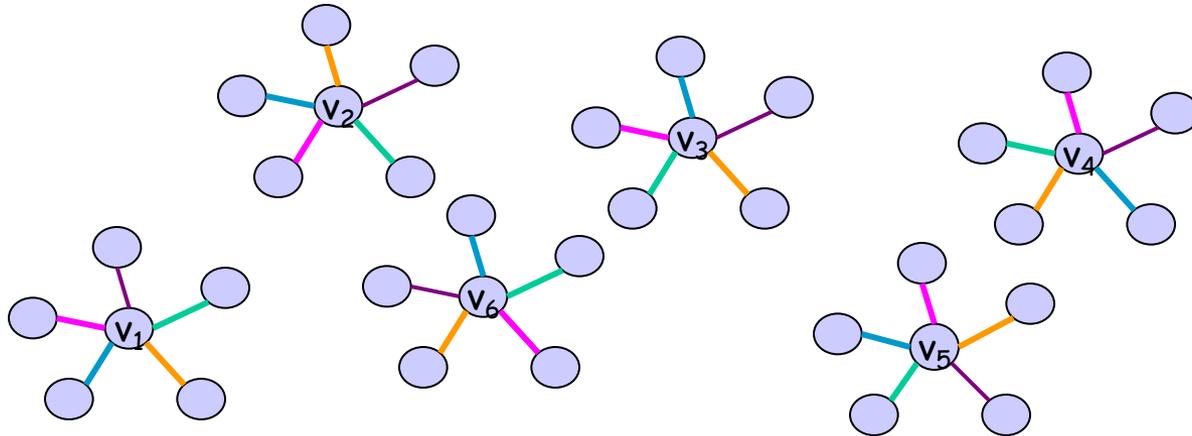
Let $v_1, v_2, …, v_\Delta$ be the centers of these stars.

We are ready to shock the algorithm!

We add a new vertex, a, and Δ edges $(a\text{-}v_1), …, (a, v_\Delta)$.

Each new edge must have a unique color (why?), that is not one of the (Δ-1) colors used to color the stars (why?) → 2Δ-1 colors must be used.

Note: the maximal degree is Δ

Online Scheduling and Load Balancing

Problem Statement:

- A set of m identical machines,
- A sequence of jobs with processing times $p_1, p_2, ...$
- Each job must be assigned to one of the machines.
- When job j is scheduled, we don't know how many additional jobs we are going to have and what are their processing times.

Goal: schedule the jobs on machines in a way that minimizes the makespan $= \max_i \sum_{j \text{ on } M_i} p_j$.

(the maximal load on one machine)

Online versus off-line

- We compare the last completion time in the schedule of the on-line algorithm to the last completion time in the schedule of an optimal off-line algorithm which knows the full sequence of jobs in advance and has unlimited calculation power.

Applications

- In operating systems:
  Assignment of jobs to computers/servers/processors
- Data storage on disks
- Giving tasks to employees
- In all these problems, on-line scenarios are often more realistic than off-line scenarios

Online Scheduling and Load Balancing

List Scheduling [Graham 1966]:

A greedy algorithm: always schedule a job on the least loaded machine.

Example: m=3 σ = 7 3 4 5 6 10

Makespan = 17
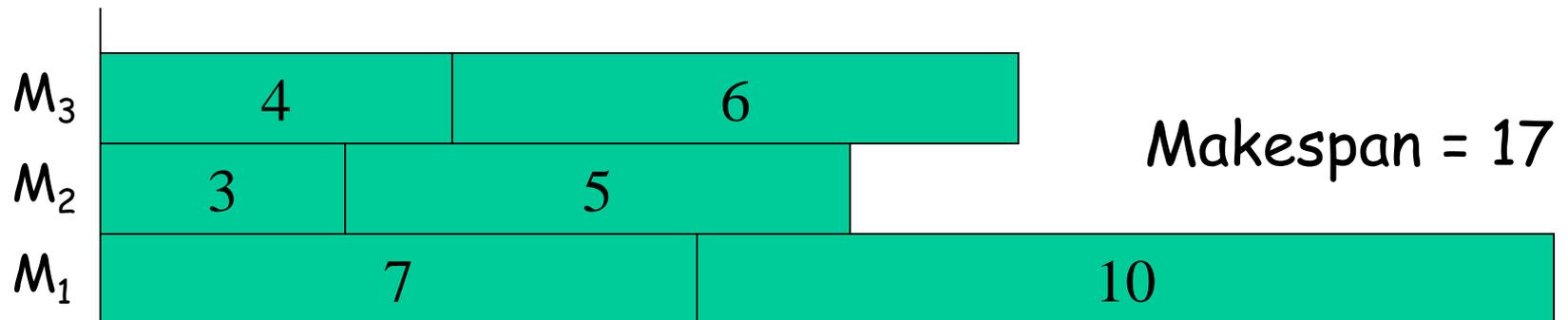
Theorem: List- Scheduling is (2-1/m)- competitive.
Proof: you did this in HW2.

The Analysis is Tight - Example

- m(m-1) unit jobs followed by a single job of size m
- OPT=m
- LS=2m-1

m

m-1

LS
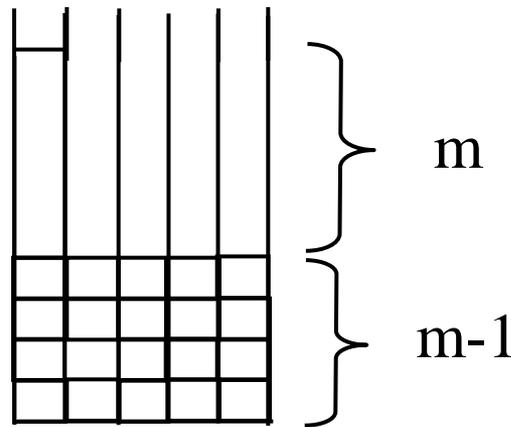
Online Scheduling

Are there any better algorithms?

Not significantly. Randomization does help.

m	deterministic			randomized	
	lower bound	upper bound	LS	lower bound	upper bound
2	1.5	1.5	1.5	1.334	1.334
3	1.666	1.667	1.667	1.42	1.55
4	1.731	1.733	1.75	1.46	1.66
∞	1.852	1.923	2	1.58	---

A lower Bound for Online Scheduling

Theorem: For m=2, no algorithm has r< 1.5

Proof: Consider the sequence σ = 1,1,2.

If the first two jobs are scheduled on different machines, the third job completes at time 3.

m_1 [1]
m_2 [2 | 3]
  alg

m_1 [3]
m_2 [1 | 2]
  opt
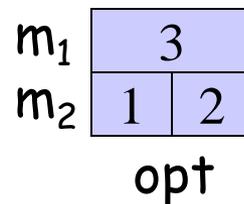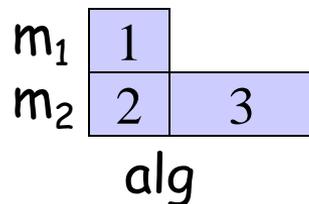
$C_A=3$, $C_{opt}=2$

r=3/2

If the first two jobs are scheduled on the same machine, the adversary stops.

m_1
m_2 [1 | 2]
  alg

m_1 [1]
m_2 [2]
  opt

$C_A=2$, $C_{opt}=1$

r=2

24
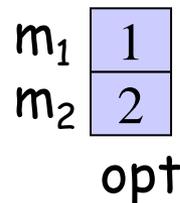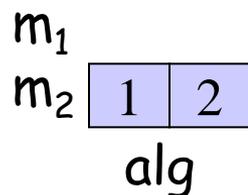
Paging: Cache Replacement Policies

Problem Statement:

- There are two levels of memory:

  - fast memory M_1 consisting of k pages (cache)

  - slow memory M_2 consisting of n pages (k < n).

- Pages in M_1 are a strict subset of the pages in M_2.

- Pages are accessible only through M_1.

- Accessing a page contained in M_1 has cost 0.

- When accessing a page not in M_1, it must first be brought in from M_2 at a cost of 1 before it can be accessed. This event is called a page fault.

Paging- Cache Replacement Policies

Problem Statement (cont.):

If M_1 is full when a page fault occurs, some page in M_1 must be evicted in order to make room in M_1.

How to choose a page to evict each time a page fault occurs in a way that minimizes the total number of page faults over time?

Paging- An Optimal **Offline** Algorithm

Algorithm LFD (Longest-Forward-Distance)

An optimal off-line page replacement strategy.

On each page fault, evict the page in M_1

that will be requested farthest in the future.

Example: $M_2=\{a,b,c,d,e\}$ n=5, k=3

σ= a, b, c , d , a , b , e , d , e , b , c , c , a , d

$$M_1 = \left\{ \begin{array}{cccccccccccc} a & a & a & a & e & e & e & e & c & c & c & c \\ b & b & b & b & b & b & b & b & b & b & a & a \\ c & d & d & d & d & d & d & d & d & d & d & d \end{array} \right.$$

                 *   *   *   *

4 cache misses in LFD

Paging- An Optimal **Offline** Algorithm

A classic result from 1966:

LFD is an optimal page replacement policy.

Proof idea: For any other algorithm A, the cost of A is not increased if in the 1st time that A differs from LFD we evict in A the page that is requested farthest in the future.

However, LFD is not practical.

It is not an *online* algorithm!

Online Paging Algorithms

FIFO: first in first out: evict the page that was entered first to the cache.

Example: $M_2 = \{a,b,c,d,e\}$ n=5, k=3

$\sigma = a, b, c, d, a, b, e, d, e, b, c, c, a, d$

$M_1 = \left\{ \begin{array}{cccccccccccc} a & b & c & d & a & b & b & b & e & e & d & d \\ b & c & d & a & b & e & e & e & d & d & c & c \\ c & d & a & b & e & d & d & d & c & c & a & a \end{array} \right.$

$\quad\quad\quad * \quad * \quad * \quad * \quad * \quad\quad\quad\quad\quad * \quad\quad\quad *$

7 cache misses in FIFO

Theorem: FIFO is k-competitive: for any sequence, #misses(FIFO) $\leq$ k #misses (LFD)

FIFO is k competitive
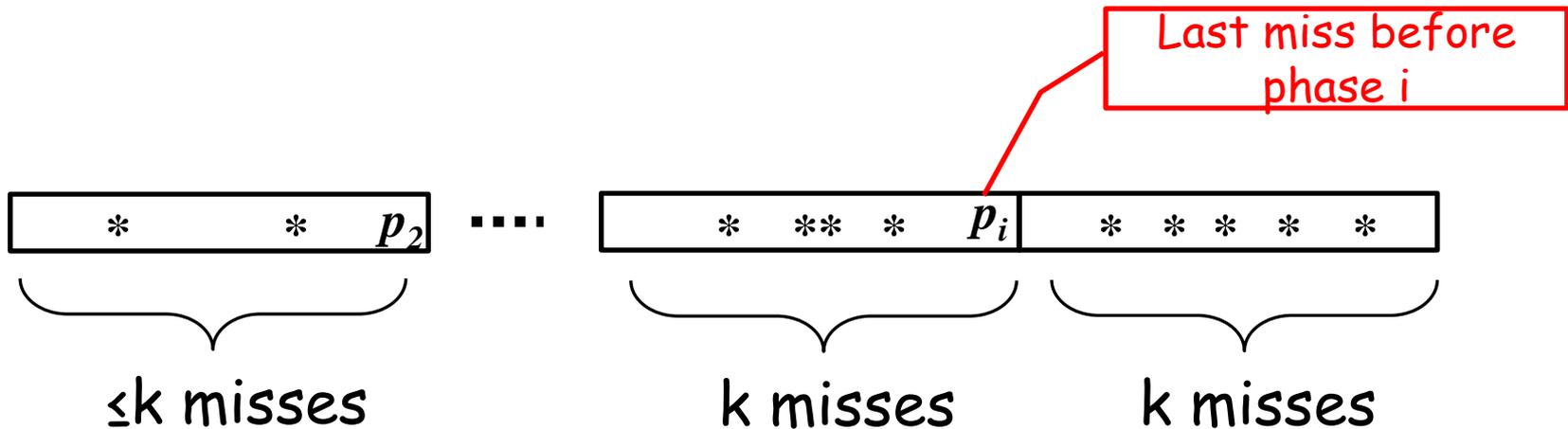
Claim: phase i contains k distinct misses different from p_i

$\Rightarrow$ OPT has at least one miss in phase i.

Proof:

- If k distinct misses different from p_i, we're done.
- If all misses different from p_i but not distinct. Some page q missed twice. q must be evicted before missed second time. $\Rightarrow$ k distinct misses.
- p_i is missed. Same argument as above with $q = p_i$.

Online Paging Algorithms

LIFO: last in first out: evict the page that was entered last to the cache.

Example: $M_2=\{a,b,c,d,e\}$ n=5, k=3

$\sigma = a, b, c, d, a, b, e, d, e, b, c, c, a, d$

$M_1 = \begin{cases} a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a \\ b\ b\ b\ b\ b\ b\ b\ b\ b\ b\ b\ b \\ c\ d\ d\ d\ e\ d\ e\ e\ c\ c\ c\ d \end{cases}$

$*\quad\quad *\ *\ *\quad *\quad\quad *$

6 cache misses in LIFO

Theorem: For all n>k, LIFO is not competitive: For any c, there exists a sequence of requests such that #misses(FIFO) $\geq$ c #misses (LFD)

Proof idea: Consider σ = 1, 2,..., k, k+1, k, k+1, k, k+1,

31

Online Paging Algorithms

LRU: least recently used: evict the page with the earliest last reference.

Example: $M_2 = \{a,b,c,d,e\}$ n=5, k=3

$\sigma = a, b, c, d, a, b, d, e, d, e, b, c$

```
        a   b   c   d   a   b   b   b   d   e
        b   c   d   a   b   d   e   d   e   b
        c   d   a   b   d   e   d   e   b   c
        *   *   *       *                   *
```

Theorem: LRU is k-competitive

Proof: Almost identical to FIFO

In practice, LRU much better than FIFO

Paging- a bound for any deterministic online algorithm

Theorem: For any k and any deterministic online algorithm A, the competitive ratio of $A \geq k$.

Proof: Assume $n = k+1$ (there are $k+1$ distinct pages). What will the adversary do?

Always request the page that is not currently in M_1

This causes a page fault in every access. The total cost of A is $|\sigma|$.

Paging- a bound for any deterministic online algorithm

What is the price of LFD in this sequence?

- At most a single page fault in any k accesses

(LFD evicts the page that will be needed in the $k+1^{th}$ request or later)

- The total cost of LFD is at most $|\sigma|/k$.

Therefore: Worst-case analysis is not so important in analyzing paging algorithm

Can randomization help? Yes!! There is a randomized $2H_k$-competitive algorithm. ($H_k = 1+1/2+1/3+...+1/k = O(\ln k)$)

The MARKING algorithm

When request r arrives:

- if r in not in cache evict a random unmarked page, and fetch r

- MARK r

- If no unmarked page, unmark all pages.

Theorem: MARKING is $2H_k$-competitive
(H_k= 1+1/2+1/3+…+1/k = O(ln k))

Proof: Consider access sequence σ. Partition σ into maximal subsequences (phases) containing k distinct pages. Each phase starts with all pages unmarked and ends with all marked.

The MARKING algorithm

Consider phase i. A request is old if it appeared in phase i-1, and new otherwise.
Let m_i be the number of new requests

Max. number of misses occurs when new requests precede old requests.

Each new requests causes a miss. $\Rightarrow m_i$ misses

After all new requests were served, there are $k-m_i$ random unmarked old pages in the cache.

Prob. that the 1st old request is one of those:

$$\frac{k - m_i}{k}$$

unmarked old pages

unmarked old pages in the cache

Prob. that the 1st request causes a miss $1 - \dfrac{k-m_i}{k} = \dfrac{m_i}{k}$

The MARKING algorithm

More generally:

After all new requests and the first j old requests were served, there are k-m_i-j random unmarked old pages in the cache.

unmarked old pages in the cache

Prob. that (j+1)'th old request is one of those? $\dfrac{k-m_i-j}{k-j}$

unmarked old pages

Prob. that (j+1)'th request is a miss $1 - \dfrac{k-m_i-j}{k-j} = \dfrac{m_i}{k-j}$

So, expected number of misses on phase i is
$$m_i + m_i\left(\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{m_i+1}\right)$$
$$= m_i + m_i\left(H_k - H_{m_i}\right) \leq m_i H_k$$

The MARKING algorithm

- Number of cache misses in phase i: $m_i H_k$

- Number of distinct pages in phases i-1 and i: $k + m_i$

- So OPT makes at least m_i misses on phases i-1,i.

- Overall, OPT makes at least $\frac{\sum m_i}{2}$ misses

- Expected number of misses of MARKING is at most $\sum m_i H_k$

- Competitive ratio is at most $2H_k$

Online Bin Packing

The input: A sequence of items (numbers), $a_1, a_2, \dots, a_n$, such that for all i, $0 < a_i < 1$

The goal: 'pack' the items in bins of size 1.   Use as few bins as possible.

Example: The input: 1/2, 1/3, 2/5, 1/6, 1/5, 2/5.

Optimal packing in two bins:

(1/2, 1/3, 1/6), (2/5, 2/5, 1/5).

Legal packing in three bins:

(1/2, 1/3), (2/5, 1/6, 1/5), (2/5)

Online BP: a_i must be packed before we know $a_{i+1}, \dots, a_n$

Online Bin Packing

Next-fit Algorithm:

1. Open an *active* bin.
2. For all i=1,2,…,n :
    – If possible, place a_i in the current active bin;
    – Otherwise, open a new active bin and place a_i in it.

Example: The input: {0.3, 0.9, 0.2}.
Next-fit packing (three bins): (0.3), (0.9), (0.2).

Theorem: Next-fit is 2-competitive.
Proof: Identical to 2-approximation in Lecture 4

Online Bin Packing

First fit algorithm: place the next item in the first open bin that can accommodate it. Open a new bin only if no open bin has enough room.

Example: items 0.6,0.2,0.8,0.1,0.3

Bin 1   0.6,0.2,0.1   OPT
Bin 2   0.8   Bin 1   0.6, 0.3, 0.1
Bin 3   0.3   Bin 2   0.8, 0.2

Theorem: $h_{ff} \leq 1.7\text{opt} +2$ (proof not here)

The HARMONIC-k Algorithm

Classify items into k intervals according to size

(1/2,1]   one item per bin
(1/3,1/2]   two items per bin

...

(1/k,1/(k-1)]   k-1 items per bin
(0,1/k]   use Next Fit

size   0 $1/k$ 1/4 1/3   1/2   1

type k 6 5 4   3   2   1
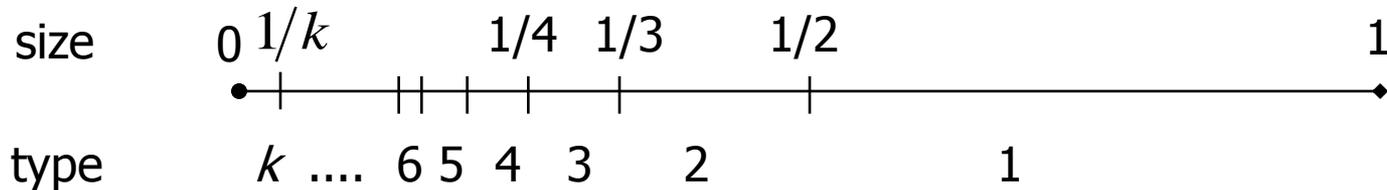
The HARMONIC Algorithm

- Each bin contains items from only one class: i items of type i per bin

- Items of **last type** are packed using **NEXT FIT**: use one bin until next item does not fit, then start a new bin

- Keeps **k-1** bins open (the bin (1/2,1] is never open)

Analysis of HARMONIC-3

- Let X be the number of bins for (1/2,1]
  - Those bins are full by more than 1/2
- Let Y be the number of bins for (1/3,1/2]
  - Those bins are full by more than 2/3
- Let T be the number of bins for (0,1/3]
  - Those bins are full by more than 2/3

Let W be the total size of all items
Then W>X/2+2Y/3+2T/3

Analysis of HARMONIC-3

- Other bounds
- OPT $\geq$ X   (items larger than 1/2)
- OPT $\geq$ W

- H3 $\leq$ X+Y+T (+2) $\leq$ 3W/2+X/4 (+2)

  $\leq$ 1.75OPT (+2)

$$W > \frac{X}{2} + \frac{2}{3}Y + \frac{2}{3}T \quad \Big| \quad \cdot \frac{3}{2} + \frac{X}{4}$$

Asymptotically, this is ignored.

Analysis of HARMONIC-4

- Let X be the number of bins for **(1/2,1]**
  - Those bins are full by more than 1/2
- Let Y be the number of bins for **(1/3,1/2]**
  - Those bins are full by more than 2/3
- Let Z be the number of bins for **(1/4,1/3]**
  - Those bins are full by more than 3/4
- Let T be the number of bins for **(0,1/4]**
  - Those bins are full by more than 3/4
- Let W be the total size of all items
  Then W>X/2+2Y/3+3Z/4+3T/4

Analysis of HARMONIC-4

- OPT $\geq$ W
- OPT $\geq$ X   (items larger than 1/2)
- OPT $\geq$ (X+2Y)/2 (items larger than 1/3)

- H4 $\leq$ X+Y+Z+T (+3) $\leq$

$$W > \frac{X}{2} + \frac{2}{3}Y + \frac{3}{4}Z + \frac{3}{4}T \quad \Big| \cdot \frac{4}{3} + \frac{X}{3} + \frac{Y}{9}$$

$\leq$ 4·W/3+X/3+Y/9 (+3) =
= 24·W/18+ (X+2Y)/18+5·X/18 (+3)
  $\leq$ 31·OPT /18 (+3) $\approx$ 1.7222·OPT (+3)

Analysis of HARMONIC

- Theorem: For any k, Harmonic-k is at most
  $$1 + \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{6 \cdot 7} + \cdots \simeq 1.691 \text{ competitive.}$$
- Proof: Not here.

Investing in the stock exchange

- You have stocks of Apple INC.
- Price changes every day.
- How much should you sell and when???

- Note: maximization problem – competitive ratio: offline OPT/ALG

- I don't know……

Investing in the stock exchange

- You have stocks of Apple INC.
- Price changes every day.
- How much should you sell and when???

- Suppose you know m ≤ price ≤ M.
- Let c = M/m

- How competitive is selling everything now?

Investing in the stock exchange

- Suppose you know m ≤ price ≤ M.
- Let c = M/m
- Selling now is c-competitive

- How about waiting to see if price gets beyond $\sqrt{Mm}$ and then selling everything?

- If sold, ratio is at least $\frac{M}{\sqrt{Mm}} = \sqrt{\frac{M}{m}} = \sqrt{c}$

- If did not sell until last day, ratio at least $\frac{\sqrt{Mm}}{m} = \sqrt{c}$

Investing in the stock exchange

- Now split sales:
- Divide stocks into log(c) equal parts
- Sell part i if price exceeds $m2^i$

Highest sold part sold for at least OPT/2.
So instead of getting OPT, got OPT/2log(c)
Competitive ratio 2log(c).
Actually, OPT/2 +OPT/4 + OPT/8 +… -> OPT,
So algorithm is in fact log(c) competitive.

What did we learn in this class?

- A little bit about many topics

- Stable marriage problem, proofs by induction

- Divide and conquer algorithms: the master thm., binary search, merge sort, counting inversions, integer/matrix mult., convex hull, closest pair

- NP-hardness

- Approximation algorithms – vertex cover, metric TSP, bin-packing, knapsack (PTAS), PCP theorem.

What did we learn in this class?

- Structured inputs – trees: MIS, facility location/covering, partition 2^i, interval graphs, Dynamic programming, tree decompositions
- Parameterized complexity: FPT, Kernelization, branching, alternative parameters, Croucelle's theorem (DP for many things).
- Linear programming – polytopes, vertices, the Simplex algorithm, Integer Programs, approximation via relaxation, LP duality, complementary slackness

What did we learn in this class?

- Randomized algorithms – Monte Carlo/Las-Vegas, Markov and Chernoff bounds, sampling, selection, skip lists, random walks, max/min cut, primality testing

- Online algorithms: ski rental, edge coloring, scheduling, paging, bin packing, sales decisions…

There is so much more

- External memory, cache oblivious algorithms
- Advanced techniques (Semi-definite programming, spectral techniques)
- Numerical algorithms
- Algorithms for "Big Data"
- Quantum algorithms
- Even more