Algorithms for optimization under uncertainty

Alberto Marchetti-Spaccamela
Leen Stougie

ECI- 2011

Algorithms under uncertainty

- NP-hardness is not the only hurdle we face in day-to-day algorithm design
- Lack of information is another…

- Knowing the future…
- Many real life situations we have to make decisions **"on-line"** without having full knowledge of the future

On-line Algorithms

Work **without** full **knowledge** of the **future**

- Deal with a **sequence** of events, **one event at a time**
- Future events are unknown to the algorithm
- The next event happens only after the algorithm is done dealing with the previous event

Problem 1: secretary problem

Secretary problem (also online dating game)

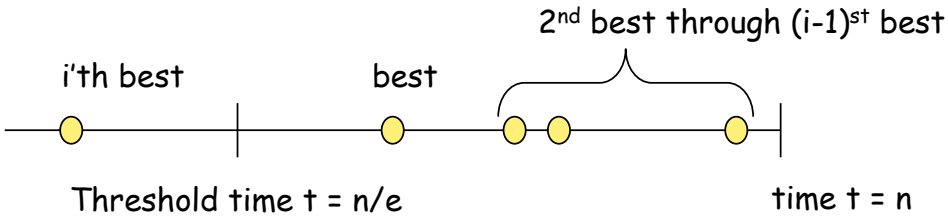
- You must hire a secretary; you interview one person a day
- There are n candidates that arrive in random order
- On the day of interview you must immediately decide whether hiring or not
- You cannot go back in time (i.e. you cannot hire a person you said "no" the day before)
- What strategy would you use to increase your chance of choosing the best candidate?

  assume we can associate to each candid. a positive value; we want to hire the candid. with max v

Secretary problem: algorithm

Algorithm: Observe first **n/e** candidates. Let **v**=max.
Pick the next candid. whose value is **> v** or take last

Theorem: Pr (picking max candidate of **S**) **> 1/e** (n large).
Proof: Select best candid. if i'th best candid. is best in first 1/e candid. and best one is first among best (i-1) candid.



Happens with prob. = $(t/n) \Sigma_{t \le i < n} (1/i) \approx (t/n) \ln(n/t) \approx 1/e$

Problem 2: ski rental

The skier's dilemma: to buy or to rent

- If you own the equipment, you take it with you, otherwise you rent
- r $ to rent, M $ to buy
- Number of trips is unknown: depends on enjoyment, injuries, weather etc.

- Goal: minimize number of $ spent

Problem 2: Off-line algorithm

If you know you will sky k times (you know the future) then the algorithm is obvious:

- If $M < kr$ then buy the first day;
- Otherwise rent

$$OPT = \min(rk, M)$$

- But k is unknown!!

Problem 2: On-line algorithm

Rule of the thumb:

Rent for floor(M/r) days (until total spent would exceed cost of buying), then buy

Analysis

- $kr < M$: $Cost_{ALG}(\sigma) = Cost_{OPT}(\sigma)$
- $kr >= M$: $Cost_{ALG}(\sigma) \leq 2M$

  $Cost_{OPT}(\sigma) = M$
- You never spend more than the double of the optimal solution

Competitive ratio (min problems)

Competitive Analysis:

- Compare the cost of an on-line algorithm with an optimal prescient algorithm on any sequence of requests. Namely off-line algorithm knows
  - the exact properties of all the events in the sequence)
  - the on-line algorithm
- The **competitive ratio** is the **ratio** between what the **on-line** algorithms **"pays"** to what the optimal **off-line** algorithm **"pays"**

Competitive ratio (min problems)

- **Formally:** let $ALG(\sigma)$ be the cost of the on-line algorithm on sequence σ. Let $OPT(\sigma)$ be the optimal off-line cost on σ then the competitive ratio is:

$$\sup_{\sigma} \frac{ALG(\sigma)}{OPT(\sigma)}$$

- Calculus: **supremum** is similar to maximum but may be achieved in the limit

Problem 3: Scheduling Jobs

Input: A set of n jobs, each job j has processing time p_j. A set of m identical machines
Goal: find a schedule that minimizes maximum completion time (makespan)
NP-complete (even for 2 machines)

The jobs are not given up-front
they arrive one by one (in adversarial order)
you have to schedule each job before seeing the next one.

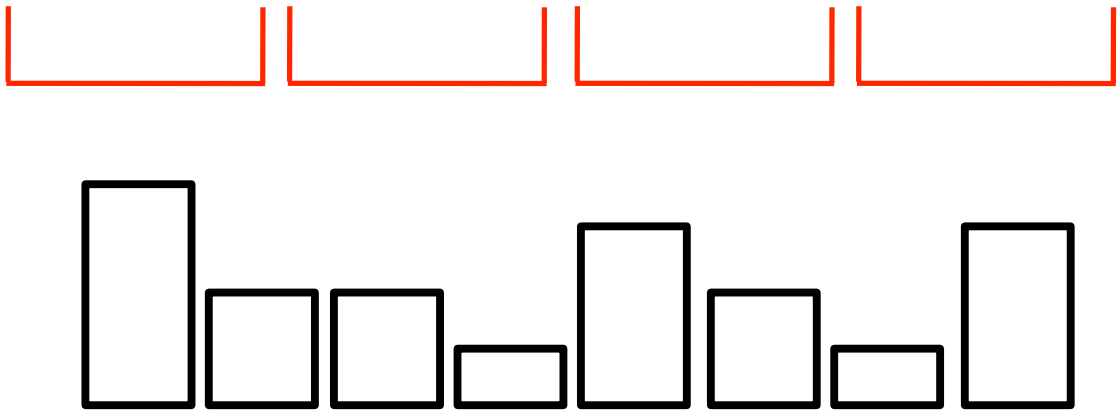
A good algorithm: Graham's Greedy Algorithm!

Graham's Greedy Algorithm

- Order the jobs $j_1, j_2, \ldots, j_n$ in some order
- Initially all the machines are empty
- For t = 1 to n
  Assign j_t to the least loaded machine so far

Note, this "online" algorithm performs within a factor of (2-1/m) of the best you could do "offline"

Moreover, you did not even need to know the processing times of the jobs when they arrived.



Problem 4: Caching

- K-competitive caching.
- Two level memory model
- If a page is not in the cache, a page fault occurs.
- A Paging algorithm specifies which page to evict on a fault.
- Paging algorithms are online algorithms for cache replacement.

Online Paging Algorithms

- Assumption: cache can hold k-pages.
- CPU accesses memory thru cache.
- Each request specifies a page in the memory system.
  - We want to minimize the page faults.

A Lower bound

- Theorem: Let ***A*** be a deterministic online paging algorithm. If ***A*** is α-competitive, then $\alpha \geq \mathbf{k}$.

- Proof: Let S ={p_1,p_2, … , p_k+1} be a set of k+1 arbitrary memory pages. Assume w.l.o.g. that A and OPT initially have p_1, … , p_k in their cache.

  In the worst case A has a page fault on any request σ_t.

LRU: Competitive Analysis

LRU : Least recently used
Evicts page whose most recent access was earliest

Theorem. LRU is k-competitive.

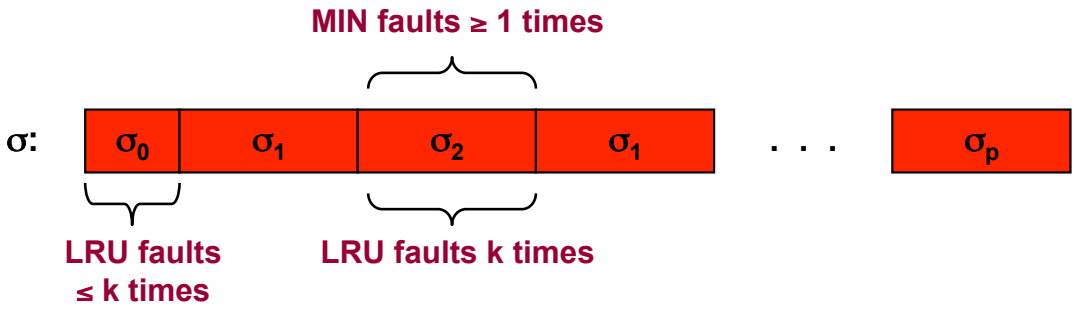
Proof: Let τ be a subsequence of σ on which LRU faults exactly k times. Let p denote page requested just before τ.

- Case 1: LRU faults in sequence τ on p.
  - τ requests at least k+1 different pages $\Rightarrow$ OPT faults at least once
- Case 2: LRU faults on a page, say q, at least twice in τ.
  - τ requests at least k+1 different pages $\Rightarrow$ OPT faults at least once

Theorem. LRU is k-competitive.

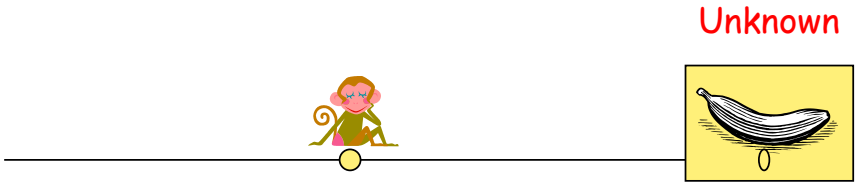
Proof: Let τ be a subsequence of σ on which LRU faults exactly k times. Let p denote page requested just before τ.

- Case 3: LRU does not fault on p, nor on any page more than once.
  - k different pages are accessed and faulted on, none of which is p
  - p is in OPT's cache at start of τ $\Rightarrow$ MIN faults at least once



Problem 5:
Monkey Looking for food

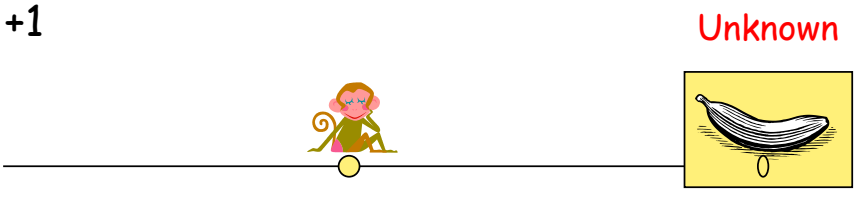
Also known as the problem of "where is the toilet in the corridor?"



What is the best competitive algorithm you can come up with?
What is its competitive ratio?

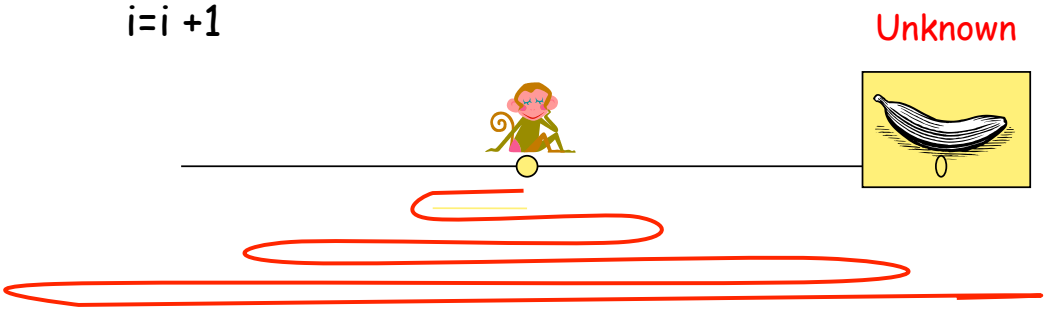
Problem 5:

Algorithm: choose one direction (say left); i=1
While not found do
  Round 2i-1: walk left for distance 2^i; return home
  Round 2i: walk right direction for 2^i; return home
  i=i +1



Problem 5:

Algorithm: choose one direction (say left); i=1
While not found do
Round 2i-1: walk left for distance 2^i; return home
Round 2i: walk right direction for 2^i; return home
i=i +1



Problem 5:

Algorithm: choose one direction (say left); i=1
While not found do
Round 2i-1: walk left for distance 2^i; return home
Round 2i: walk right direction for 2^i; return home
i=i +1

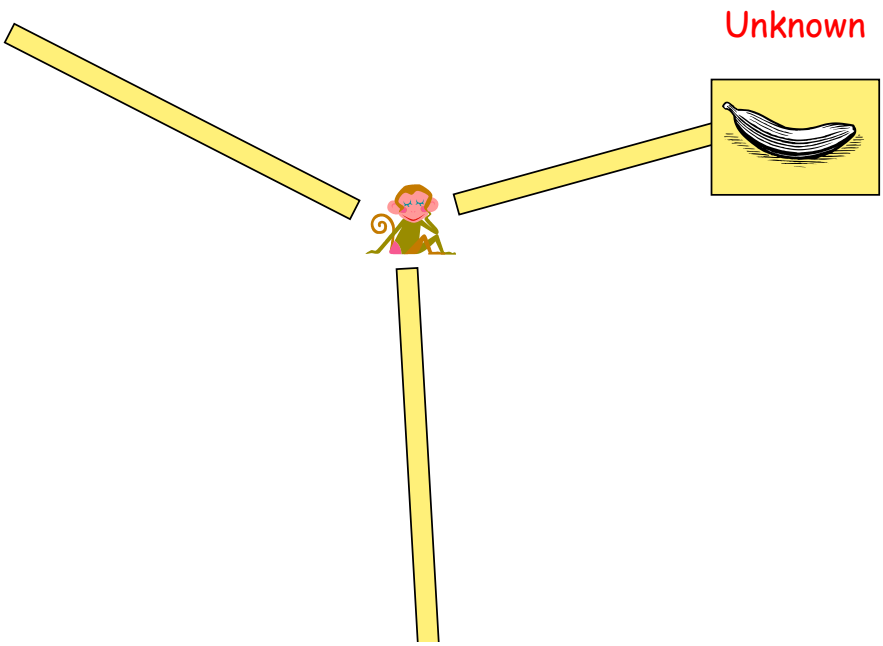
Algorithm is 13 competitive

Better algorithms

- Randomised algorithm: choose the initial direction randomly
- Change the function gives the distance walked at each round

Problem 5: (3Dim.)

- Monkey looking for food.



Questions?