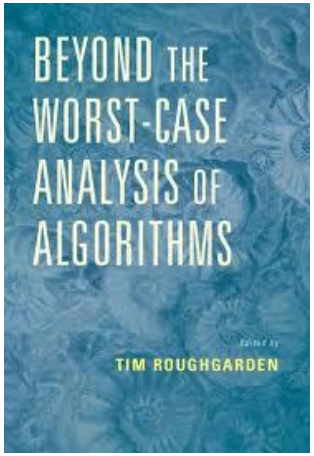
Advanced topics on Algorithms

Luciano Gualà
www.mat.uniroma2.it/~guala/

Beyond Worst-Case Analysis
Episode I
(pilot)

main reference:



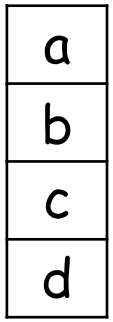
motivating examples

Cashing

- 2 types of memory:
  - a cache: small & fast
  - main memory: large & slow
- a program requesting data as a sequence of pages
- page fault (cache miss): the requested page is not in cache

algorithmic question: which page should be evicted?

cache



FIF: farthest in the future (offline optimal)

FIFO: first-in first-out

LRU: least recently used

sequence: a b c d d a e f

research challenge: develop a theory to explain LRU's empirical superiority

Linear programming

$$\text{minimize} \quad \sum_{S \in \mathcal{S}} c(S) x_S$$

$$\text{subject to} \quad \sum_{S : e \in S} x_S \geq 1 \qquad e \in U$$

$$x_S \geq 0 \qquad S \in \mathcal{S}$$

which algorithm do you know to solve a linear program?

- simplex algorithm (exponential running time in the worst case, but almost linear in practice)
- ellipsoid method (poly-time in the worst case, but not so good in practice)

research challenge:
develop theory to explain the empirical performance of the simplex method

Clustering

- give n data points, identify "meaningful" groups
- unsupervised learning
- what does it mean "meaningful"?
- usual approach: pose an objective function and optimize it
  - e.g.: k-center, k-means, k-median.

Meta-theorem: Every standard approach of modeling clustering as an optimization problem results in an NP-hard problem.

research challenge:
prove that clustering is hard only when it doesn't matter

Analysis of algorithms: why

goal 1 (Performance Prediction):
explain or predict the empirical performance of an algorithm

goal 2 (Identify Optimal Algorithms):
rank different algorithms according to their performance, and ideally to single out one algorithm as "optimal"

goal 3 (Design New Algorithms):
guide the development of new algorithms

Summarizing the performance of an algorithm: WC analysis

cost(A,z): amount of resources algorithm A consumes for input z

- e.g., running time, space, I/O operations, cost of a solution

The WC analysis summarizes the performance of as:

$$\max_{z} \text{cost}(A,z)$$

(sometime parameterized by the length n of the input z)

Pros of the WC analysis :

1. good wc upper bound are awesome: in the happy case no need to think about the nature of the application domain or the input

2. Mathematical tractability: usually easier to establish tight bounds in the worst case. (Note: the utility of a mathematical model is not necessary monotone increasing in its realism.)

3. No data model: worst-case analysis is particularly sensible for "general-purpose" algorithms (sometimes this is what you want).

Cons of the WC analysis :

1. Overly pessimistic: overestimate performance on most inputs (e.g. simplex method).

2. Can rank algorithms inaccurately: sometime fails in identifying the right algorithm to use in practice (e.g., FIFO vs LRU, Simplex method vs ellipsoid method).

3. No data model: "Murphy's Law" data model: whatever algorithm you choose, nature will conspire against you to produce the worst-possible input. Unrealistic.

Cashing
(Online paging)

the model

- 2 types of memory:
  - a cache: small & fast
  - main memory: large & slow
- page requests arrive "online": one request per time step
- online algorithm: it takes irrevocable decisions at each time step

k: size of the cache

cost(A,z): # of page faults/cache misses A incurs in the sequence z

Definition (competitive ratio) [Sleator&Tarjan, STOC 84]
A competitive ratio of an online algorithm A is:

$$\max_{z} \frac{cost(A,z)}{cost(Opt,z)}$$

optimal offline algorithm for z

FIF: farthest in the future

Theorem [Sleator&Tarjan, STOC 84]
Every deterministic paging algorithm has competitive ratio at least k.

proof

fix a deterministic online algorithm *A*

assume the number of existing pages is N=k+1

an adversarial request sequence z for *A*:
- at each time step, ask for the only missing page

A incurs a page fault at each of the n=|z| time steps

cost(*A*,z) = n

cost(Opt,z) $\leq$ k+n/k

except for the first k time steps, each page fault is follower by k-1 page hits in Opt
(k candidates for eviction, one of these will not be among the next k – 1 requests)

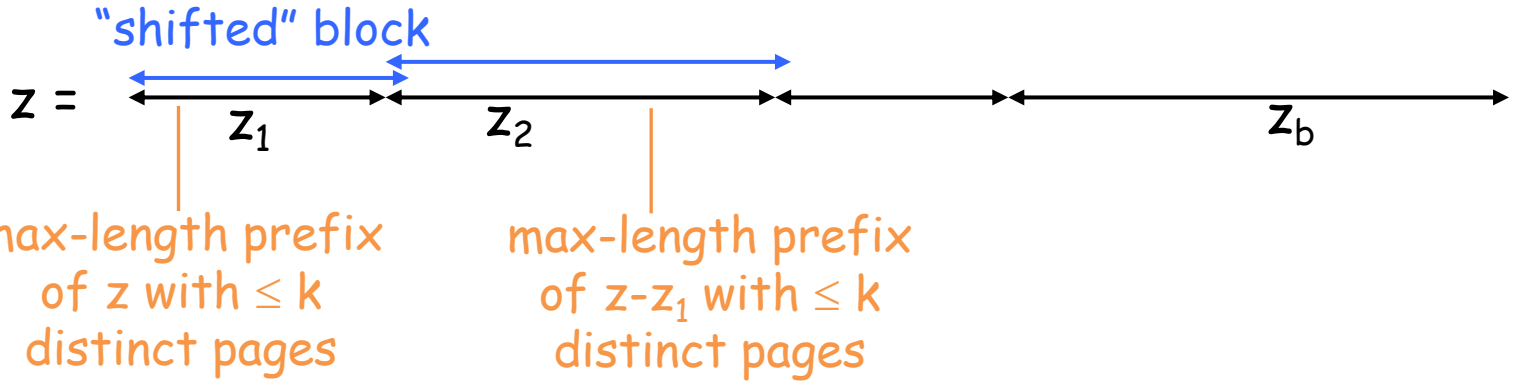
$\Rightarrow$ competitive ratio $\geq \frac{n}{k + n/k} \approx k$ when n goes to infinity

Theorem [Sleator&Tarjan, STOC 84]
The competitive ratio of LRU is at most k.

proof

fix a request sequence z

we divide z into blocks as follows:



LRU can have at most k faults in each block

Opt must have at least 1 fault in each "shifted" block

cost(Opt,z) ≥ b-1

⇒ competitive ratio ≤ $\frac{k\,b}{b-1} \approx k$ when b goes to infinity

observation 1: LRU is optimal

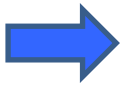
observation 2: the optimal competitive ratio is k
(it gets worse when cache becomes larger)

Flush-When-Full(FWF):
when the cache is full and a page fault is incurred, evict all the pages in the cache

FWF is better or worse than LRU? ...clearly worse!

Theorem
The competitive ratio of FWF is at most k.

(same proof works here)

 the theory cannot differentiate LRU from FWF (and FIFO...)

Resource Augmentation

idea: compare our protagonist (e.g. LRU) against a weaker competitor

weaker competitor: FIF but with a cache of size $h \leq k$

Theorem [Sleator&Tarjan, STOC 84]
The competitive ratio of LRU with cache size k is at most

$$\frac{k}{k - h + 1}$$

with respect to the optimal offline algorithm with a cache of size $h \leq k$

proof

LRU can have at most k faults in each block

consider a "shifted" block ending with a request p:

Opt with cache size h incurs at least

k (request other than p) - (h-1) (pages in cache other than p) page faults in that "shifted" block

Corollary: if LRU has $k \approx 2h$ (roughly double the cache size of Opt), then it is 2-competitive

Interpretation

A Two-Step Approach to System Design:

1. to estimate the cache size that guarantees acceptable page fault rate assuming an optimal algorithm.
2. double the cache size to guarantee acceptable performance under the LRU algorithm

It can be proved the following theorem (LRU is Loosely Competitive):

Theorem [Young 94]

For every ε, $\delta > 0$ and positive integer n, for every request sequence z, for all but a δ fraction of the cache sizes k in $\{1, 2, \ldots, n\}$, the LRU algorithm satisfies either:

(1) $\text{cost}(\text{LRU}, k, z) = O(1/\delta \log 1/\varepsilon)\ \text{cost}(\text{Opt}, k, z)$; or

(2) $\text{cost}(\text{LRU}, k, z) \leq \varepsilon |z|$