Online Algorithms

- Theory of Computation
- School of Electrical and Computer Engineering
- National Technical University of Athens
Άδεια Χρήσης

Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.

Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άδεια χρήσης άλλου τύπου, αυτή πρέπει να αναφέρεται ρητώς.
Types of Problems

- For certain problems, input is not available from the beginning

- Certain decisions are requested on the way
  - Output required
Online vs Offline

Online Algorithms
- Input arrives as sequence of input portions
- The system must react in response to request
- Future input is unknown
- Not optimal

Offline Algorithms
- Entire input is given in advance
- Solve the problem at hand
- Future is known
- Optimal
Competitive Analysis

- Big O complexity can’t be used:
  - For every algorithm there will be a sequence that makes it look foolish

- Competitive Ratio
  - Comparison with an optimal offline algorithm processing the same sequence of requests
  - Maximum cost over all possible input sequences divided by the cost of an optimal offline algorithm
  - Related to minimax concept of game theory
    - Online player vs Adversary
Competitive Analysis

- A little formalism:
  - $\text{cost}_A(\sigma)$: the cost of an online algorithm $A$ on the input sequence $\sigma$
  - $\text{cost}_{\text{OPT}}(\sigma)$: the cost of the optimal offline algorithm on $\sigma$

- Algorithm $A$ is $c$ competitive if there exists a constant $b$ such that on every request sequence $\sigma$:

\[
\text{cost}_A(\sigma) \leq c \cdot \text{cost}_{\text{OPT}}(\sigma) + b
\]
The Ski Rental Problem

- Cost for renting a pair of skis
- Cost for buying a pair of skis

Rent or Buy? When?
  - How do we decide?

Request = “Take a ski trip”
Actions = “rent” | “buy” | “use skis already bought”
Costs = 1, s, 0 respectively

On a sequence of t requests any sensible online algorithm is of the form:

“Rent for the first k trips, then buy, then use already bought”
The Ski Rental Problem

- **Online Cost**
  - $t$, $t \leq k$
  - $k+s$, $t > k$

- **Offline Cost**
  - $\min(s, t)$

- Find $k$ that minimizes the competitive ratio.
- For given $k$, $k+1$ maximizes the ratio
  \[
  \frac{k+s}{\min(k+1, s)}
  \]

- For given $k$, $k+1$ requests maximize the ratio. The ratio is minimized for $k = s - 1$

- The on-line player should rent until enough ski trips have occurred so that he would have done better if he had bought skis initially
Paging

- Memory management scheme
- Memory hierarchy
- Page fault minimization

- Set of $n$ pages
- RAM with capacity for $k$ pages
- The system receives requests for pages in RAM
- If the referenced page is in the RAM, the request can be served
- If not, then a page fault occurs
- The missing page is loaded from secondary storage and an online algorithm has to decide which page to evict
Paging

- **Common algorithms**
  - LRU: evict the page in memory that was requested least recently
  - FIFO: evict the page that has been longest in memory

- **Theorem**
  - FIFO and LRU are k-competitive, where k the size of main memory in pages

- There exists a more general class of algorithms that achieve a competitiveness of k
Paging

- **Marking**
  - Each page is associated with a bit called *mark*
  - Initially all pages are set as unmarked
  - Stages of page requests
  - A page is marked when it is first requested in this stage
  - On a fault, an unmarked page is evicted

- **Theorem**
  - Any marking algorithm is $k$-competitive
Paging

- Theorem
  - No deterministic online algorithm for the paging problem can achieve a ratio smaller than $k$

- Proof
  - Optimal Offline Algorithm
    - Belady’s greedy algorithm
    - “Sees” in the future
    - On a fault, evict the page whose next request occurs furthest in the future
Paging

Proof

- A and OPT start with the same set of pages in memory
- The adversary restricts its request sequence to a set of k+1 pages, the pages initially in the memory and another one
- It always requests the page that is outside of the memory
- This can be continued for an arbitrary number of requests, resulting in a sequence \( \sigma \) on which A faults on every request

- What remains is to show that \( \text{cost}_{OPT}(\sigma) \leq \left\lfloor \frac{|\sigma|}{k} \right\rfloor \)

- At each fault, the adversary evicts the page whose first request occurs furthest in the future
- The adversary is guaranteed that there will be at least k-1 pages requested between any two faults, so the adversary faults at most on every \( k^{th} \) request
Adversaries

- Online algorithms can achieve better performance if they are allowed to make random choices
- The competitive ratio of a randomized algorithm is defined with respect to an adversary

- There are three types of adversaries:
  - oblivious adversary (weak)
    - generates the whole request in advance
  - adaptive online adversary (medium)
    - it may observe the online algorithm and generate next request based to all previous requests
  - adaptive offline adversary (strong)
    - knows everything. Even randomization can’t face it
Secretary Problem

- Also known as the marriage problem, the game of googol
- There is a single secretarial position to fill
- There are $n$ applicants for the position
- The applicants can be ranked from best to worst unambiguously
- The goal is to have the highest probability of selecting the best applicant of the whole group
- They are interviewed sequentially in random order
- Immediately after the interview, the applicant is either accepted or rejected irrevocably
Secretary Problem

Strategy
- Naive: pick the \( i^{th} \) candidate: \( P(\text{Success}) = \frac{1}{N} \)
- Interview the first \( r \) applicants for \( r < n \)
- Accept the very next applicant that is better than all the first \( r \) you interviewed

- \( A = n+1 \) the best applicant, \( r \) the last that will be rejected
Secretary Problem

- Strategy
  - A won’t be chosen, unless:
    - \( n \geq r \)
    - The highest applicant in \([1,n]\) is the same as in \([1,r]\), \( P = \frac{r}{nN} \)

\[
P(\text{Success}) = P(r) = \frac{1}{N} \left[ \frac{r}{r} + \frac{r}{r+1} + \cdots + \frac{r}{N-1} \right] = \frac{r}{N} \sum_{n=r}^{N-1} \frac{1}{n}
\]

- For the optimal solution, \( P'(r)=0 \Rightarrow r = \frac{1}{e} \approx 0.37 \)

- Coincidentally, \( P(r_{max}) = \frac{1}{e} \)
Applications and Further Research

- Stock Markets
  - Algorithms for stock prediction

- Large networks
  - Network switches
  - TCP Acknowledgement

- Robot Motion Planning

- Bin Packing

- Storage Allocation and Cache Management

- Job Scheduling
Bibliography

- Online Algorithms – Susanne Albers
- Competitive Analysis of Paging: A Survey – Sandy Irani
- Algorithm Design – Kleinberg, Tardos
Questions?
Ευχαριστώ
Χρηματοδότηση

Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.

Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα» του ΕΜΠ έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του υλικού.

Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.