Online Paging + Online Set Cover

Convex Optimization Code (program)

Resource Allocation Problems

Cloud / Large Cluster of processors

Heavily online, applications arrive & depart freely

→ Algo is unaware of input ahead of time
→ Maintain feasible solutions always
→ Few changes over time (can't entirely reoptimize each time)
→ "competitive", (i.e.) be reasonable for underlying objective function

Paging

k cache slots, need policy of what to evict when cache is full?

→ Well motivated in online setting

3 pages: A, B, C
2 cache slots

Offline (traditional): know exact page requests

AA B CA A BB B C A C C

\[
\text{How to maintain cache so that as few cache misses}
\]

2 cache slots

\[
\begin{align*}
\text{A} & \quad \text{B} \\
\text{A} & \quad \text{C} \\
\text{A} & \quad \text{B} \\
\end{align*}
\]

When C arrives, evict B

\[
\begin{align*}
\text{A} & \quad \text{C} \\
\text{B} & \quad \text{again comes, evict A} \\
\text{A} & \quad \text{C} \\
\end{align*}
\]

\[
\begin{align*}
\text{Goal: Minimize \# evictions.}
\end{align*}
\]
A comes, exist B

Use flow algorithms to determine optimal eviction
policy for a given sequence of request

Opt Alg = "farthest into future"

Model is flawed → don't know future requests!!

Want online algo, minimize # evictions!!

LRU: least recently used. (Intuition is programs have
locality of reference)

How good is this algo?

How to measure "goodness" of online algorithms?

\[
\text{Competitive Ratio} = \frac{\text{Worst Case ratio}}{\text{Optimal ratio}}
\]

What is competitive ratio of LRU

k cache slots

Consider the \( \sigma \) (input sequence) = 1, 2, 3, \ldots, k

Cache = empty initially

\# evictions (LRU on \( \sigma \)) = 1

\[\text{Opt}(\sigma) = \begin{array}{c}
1 \\
2 \\
3 \\
k
\end{array}
\rightarrow \begin{array}{c}
1 \\
2 \\
k
\end{array} = 2
\]

(Since it's max over all \( \sigma \))

Thus: Competitive Ratio (LRU) \( \leq \frac{k}{2} \)

Randomization of LRU

MARKING ALGORITHM...
MARKING ALGORITHM:

Suppose cache has already \( k \) pages in it initially, call each of them unmarked.

When a request arrives,

1) If it's in cache, serve it, mark that page.
2) If it's not in cache, bring it in, mark it.
3) If all pages are marked, reset markers to unmarked.

What's the competitive ratio of marking? \( O(\ln k) \)

Proof Sketch:

Phase begin

Phase end

Phase begin

Phase end

Phase begin

Phase end

Hand wavy fact: Marking is hurt most when old pages are requested after new pages.

Worst Case

Expected \# cache evictions:

\[
\leq k + \left( \frac{k}{k} \right) + \left( \frac{k}{k-1} \right) + \left( \frac{k}{k-2} \right) + \ldots + 1
\]

\( \text{Pr}(\text{first old page was thrown out by the } k \text{ new pages that entered}) \)

\( \text{Pr}(\text{2nd old page was thrown out before it arrived}) \)

\( \leq k \cdot \ln k \)
Online Set Cover

Collection of Sets
Collection of Elements (Universe of all elements)
Pick min # sets to cover universe

Alg 1: Greedy (Approx. Ratio is \( \ln n \))

\[ \sum_{\text{phs}} E\left( \sum_{\text{evict of alg in current phase}} \right) \leq 2\ln k \]

Marking outperforms LRU (exponentially better) according to competitive ratio

Ex: Any randomized algo has competitive ratio \( > 2(\ln k) \)

Let \( x_i \) be # new requests in phase \( i \)

\[ E(\text{Alg}) \leq \sum x_i \cdot \ln k \]

\[ \text{Opt} \geq \sum x_i \cdot \frac{1}{2} \]

\[ x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \]
Opt = 2, Greedy = \frac{1 + 1 + 1 + \cdots + 1}{\log_2 n} = \frac{\log n}{2}

Alg 2: LP rounding

\begin{align*}
\text{Min} & \quad \sum x_s \\
\text{s.t.} & \quad \sum_{s \in S} x_s \geq 1 \\
& \quad x_s \geq 0
\end{align*}

\[ \text{compute optimal fractional solution} \]

\[ \text{round it} \] (randomized rounding)

Greedy is inherently offline

Q: Online vs. offline: elements needing connectivity only show up over time!

We need to maintain a good set cover for the elements that have currently shown up.

Alg knows m possible sets at time 0

\[ \text{time 1: element } e_1 \text{ arrives} \]

Alg picks a set to cover \( e_1 \).

Incrementally builds a set cover solution

\[ \text{time 2: element } e_2 \text{ arrives} \]

Alg may pick a new set if existing sets don't cover \( e_2 \).

\[ \text{time 3: element } e_3 \text{ arrives} \]

Alg may pick new set if previously included sets don't cover \( e_3 \).

\[ \text{time } T \]

\[ \text{...} \]

\[ \text{...} \]
How good is an algo?

Competitive Ratio: \[ \max_\sigma \frac{E[\text{# sets bought by online algo for } \sigma]}{\text{# sets which optimal set cover would buy}} \]

Candidate Algo: (C6)

When new elt arrives, include set which can most elt.

\[ S_0 = \{ e_1, e_{100}, e_{101}, e_{102}, \ldots, e_{109} \} \]

\[ S_1 = \{ e_1, e_2, e_3, e_4, e_5 \} \]

\[ S_2 = \{ e_2, e_{200}, e_{201}, e_{202}, \ldots, e_{209} \} \]

\[ S_3 = \{ e_3, e_{300}, e_{301}, \ldots, e_{399} \} \]

\[ S_4 = \{ e_4, e_{400}, e_{401}, \ldots, e_{499} \} \]

\[ S_5 = \{ e_5, e_{500}, \ldots, e_{599} \} \]

CA picks S_1

\[ S_0 \]

\[ e_1 \text{ arrives} \]

\[ \downarrow \]

CA picks S_1

\[ S_1 \]

\[ e_2 \text{ arrives} \]

CA picks S_2

\[ S_2 \]

\[ e_3 \rightarrow \text{CA picks S}_3 \]

\[ S_3 \]

\[ \text{CA} = 5 \text{ for sequence} \]

\[ \text{Opt} = S_0 = 1 \text{ for sequence} \]

Instead, alternate Meta Algo

Maintain utility of sets (confidence that these sets are useful)

Every time new elt comes, update the confidence/utility of every set

Include the most useful set

Algo

Fractional Algorithm

\[ \text{minimize} \sum x_s \]

\[ \sum_{s \in S} x_s \geq 1 \]

\[ x_s \geq 0 \]

\[ \forall e \text{ currently arrived elements} \]

a) New elt brings new constraint
a) New elt brings new constraint on \( x \) values to be monotonically non-decreasing.

\[
\begin{align*}
x_0 &= 0.1, \quad \rightarrow x_1 = 0.2, \rightarrow x_2 = 0.7, \rightarrow x_3 = 0.9, \rightarrow x_4 = 1.
\end{align*}
\]

b) “Relaxation of constraint that a set, once bought, can't be unbought later.”

Algorithm:

- Minimize \( \sum x_e \)
  - \( \sum x_e \geq 1 \) for all \( e \in s \)

- Maximize \( \sum y_e \)
  - \( \sum y_e \leq 1 \) for all \( e \in s \)

- Initialize all \( x_e = \frac{1}{m} \)
- When new element \( e \) arrives,
  - while \( \sum x_e < 1 \)
    - update \( x_e \leftarrow 2x_e \) if \( e \) covered \( e' \)
    - update \( y_e \leftarrow y_e + 1 \) for analysis

- Cleverness:
  - If a set is repeatedly useful, its \( x \) value jumps up to 1 very quickly.

Recap algo:
- Analyze the competitive ratio of LP:
  - For any sequence, \( \text{LP cost (for third)} \leq 0 \text{Opt \# sets} \)
  - \( \text{Opt \# sets} \)

Next class:
- Online Matching

Initial all sets, \( S_1 \), \( S_2 \), \( S_3 \), \( S_4 \), \( S_5 \)