CS420+500: Advanced Algorithm Design and Analysis

Lectures: March 22 + March 24, 2017

Prof. Will Evans *Scribe: Sean Copeland*

Euclidean TSP is NP-hard [Papadimitriou '77]

Hamiltonian Cycle: Given an unweighted graph G

Does G contain a cycle that visits every vertex once?

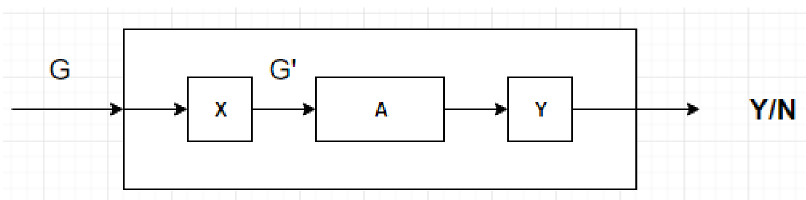
Hardness of Approximation

The general TSP is NP-hard to approximate

Claim:

If P $\neq$ NP

then there is no polynomial time c-approximation algorithm for TSP.



Transform X: Create G$'$ from G=(V,E) |V|=n where G$'$ has all edges.

$$w(u,v) = \begin{cases} 1 & if(u,v) \in E \\ c|V|+1 & if(u,v) \notin E \end{cases}$$

Transform Y: if | TSP(G$'$) | $\leq c|V|$

then output yes,

else no

Why does this work?

Edges not in the original graph are so costly that there is a gap between the cost of a tour if G contains a Ham cycle (cost=n) and cost of tour if G doesn't (cost $\leq c|V||$).

Online Algorithms

For input sequence $p_1, p_2, ..., p_n$ (very large) an <u>online</u> algorithm must produce an output given $p_1, p_2, ..., p_n$ (without seeing $p_1, p_2, ..., p_n$) for each i

<u>ex.</u>

Page replacement in cache

$p_1, p_2, ..., p_n$ is a sequence of page requests made by a program

k is cache size (number of pages)

At i^{th} request, p_i, the cache contains some k pages If p_i is not in cache (page fault) some page must be evicted from cache to make room for p_i, then p_i is added to cache.

The cost of a page replacement algorithm A on a sequence $p_1, p_2, ..., p_n$ is $f_A(p_1, p_2, ..., p_n) =$ number of faults on $p_1, p_2, ..., p_n$

Online algorithm must decide what page to evict without knowing future requests.

ex.

Page Replacement Algorithms:

Least Recently Used (LRU) - evict each page whose most recent request occurred furthest in the past

Least Frequently Used (LFU) - evict page that has been requested least often

Marking Algorithms - poor man's LRU with randomization

First In First Out (FIFO) - evict page that has been in cache longest

How do we decide best online algorithm?

1) Worst-case performance

$$max(p_1, p_2, ..., p_n) = \begin{cases} \text{f}_{\text{LRU}}(p_1, p_2, ..., p_n) = n \\ \text{f}_{\text{LFU}}(p_1, p_2, ..., p_n) = n \\ \text{f}_{\text{FIFO}}(p_1, p_2, ..., p_n) = n \end{cases}$$

2) Average case performance m = total number of pages possibly requested

Expected number of page faults on a sequence of randomly, uniformly, independently chosen

pages: E[$f_{\text{LRU}}(p_1, p_2, ..., p_n)$] = (1 - K/M) * N

E[$f_{\text{LFU}}(p_1, p_2, ..., p_n)$] = (1 - K/M) * N

E[$f_{\text{FIFO}}(p_1, p_2, ..., p_n)$] = (1 - K/M) * N

3) Competitive Analysis

How does the online algorithm's performance compare to that of the best offline algorithm?

An online algorithm A is c-competitive

if there exists b

for all $p_1, p_2, ..., p_n$

$f_{\text{A}}(p_1, p_2, ..., p_n) \leq c * f_{\text{OPT}}(p_1, p_2, ..., p_n) + b$, f_{OPT} knows future

Thm LRU and FIFO are k-competitive

Thm If A is deterministic online algorithm for paging then $c \geq k$