CPSC420 Mar22+24

Saturday, 25 March 2017 10:33

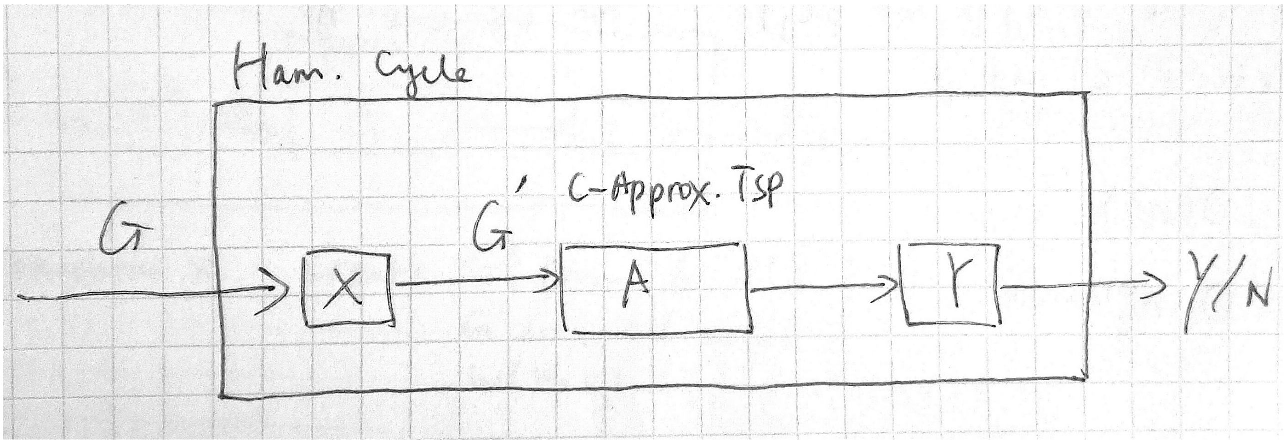
March 22 Wednesday

Euclidean TSP is NP-Hard [Papadimitriou '77]

Hamiltonian Cycle: Given unweighted graph G, does G contain a cycle that visits every vertex once?
Hamiltonian Cycle is NP-hard.

Hardness of Approximation
The general TSP is NP-Hard to approximate.

Claim: If P≠ NP then there is no polytime c-approximation algorithm for TSP.
Proof: Suppose A is a polytime c-approximation algorithm for TSP.



Transform X:

Create G' from G = (V,E), |V|=n
G' has all edges

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

Transform Y: If $|TSP\,A(G')| \leq c|V|$ then output yes, otherwise no.

Why does this work?

Edges not in the original graph are so costly that there's a gap between cost of tour if G contains a Hamiltonian cycle (cost=n) and cost of tour if G doesn't

(cost > c|V|)

March 24 Friday

Online Algorithm

For input sequence P1P2...Pn in which n is very large, an online algorithm must produce an output given a partial input P1P2...Pi (without seeing Pi+1...Pn) for each i.

Example: Page replacement in cache

P1P2....Pn is a sequence of page requests made by a program.
K is the size of cache.
At i^{th} page request, Pi, the cache contains some k pages.
If Pi is not in the cache (page fault) , some page must be evicted from cache to make room for Pi, then Pi is added to cache.

The cost of a page replacement algorithm A on a sequence P1P2...Pn is
$fA(P1P2 \ldots Pn) = \#faults\ on\ P1P2 \ldots . Pn$

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

Example Page replacement algorithm:

Least Recently Used (LRU) : Evict page whose most recent request occurred furthest in the past.
Least Frequently Used (LFU) : Evict page that has been requested least often.
Marking Algorithm: poor man's LRU (with randomization)
FIFO: Evict page that has been in cache longest.

How do we decide the best online algorithm?

1. Worst-case performance

$$\max_{P1P2 \ldots Pn} \begin{cases} fLRU\ (P1P2 \ldots Pn) = n \\ fLFU\ (P1P2 \ldots Pn) = n \\ fFIFO(P1P2 \ldots Pn) = n \end{cases}$$

2. Average-case performance:
   (m=total # of pages possibly requested)
   Expected # page fault on sequence of randomly, uniformly, independently chosen pages:
   $E[f_{LRU}(P1P2...PN)] = (1 - k/m)*n$
   $E[f_{LFU}(P1P2...PN)] = (1 - k/m)*n$
   $E[f_{FIFO}(P1P2...PN)] = (1 - k/m)*n$

3. Competitive Analysis
   How does online algorithm's performance compare to best offline algorithm?
   An online algorithm A is c-competitive if
   there exists b such as
   for all P1P2...Pn
   $f_A(P1P2...Pn) \leq c * f_{OPT}(P1P2...Pn) + b$

Thm: LRU & FIFO are k-competitive in which k = cache size.
Thm: If A is a deterministic online algorithm for paging, then $c \geq k$.