CS880: Approximation and Online Algorithms	**Scribe:** Ashwin Maran
Lecture 18: Online Routing; Caching	**Date:** 10/28/2019

18.1 Online Routing

Given a graph $G = (V, E)$ with edge capacities c_e, requests (s_i, t_i, b_i) arrive over time. For any set of $s_i - t_i$ paths represented by P_i, the load on an edge e is defined as

$$l_e = \sum_{i:P_i \ni e} b_{i,e}$$

The goal of the problem is to pick $s_i - t_i$ paths P_i, such that the following quantity is minimized.

$$\max_{e \in E} \left(\frac{l_e}{c_e} \right)$$

Lemma 18.1.1 *Without loss of generality, we may assume that* $c_e = 1 \ \forall \ e \in E$.

Proof: Given an online routing problem with a graph $G = (V, E)$ having arbitrary edge capacities $c_e > 0$ with the requests (s_i, t_i, b_i) arriving over time, we can reduce it to a problem on the the graph $G = (V, E)$ having edge capacities $c'_e = 1 \ \forall \ e \in E$ with the requests (s_i, t_i, b'_i) arriving over time, where

$$b'_{i,e} = \frac{b_{i,e}}{c_e} \ \forall \ i, e$$

■

So, we might as well consider the problem as that of finding

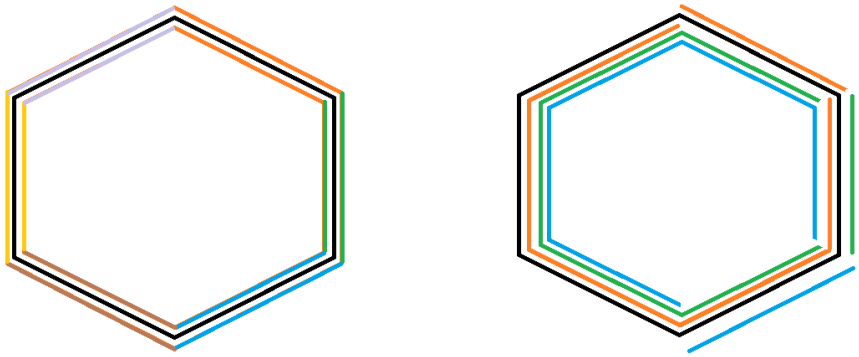
$$\min_{s_i - t_i \text{ paths } P_i} \left(\max_{e \in E} l_e \right)$$

Lemma 18.1.2 *The greedy algorithm for solving the online routing problem has a competitive ratio of* $\Omega(n)$.

Proof: Consider the cycle graph G on a set of n vertices. There are a total of $2n$ requests that arrive over time, and for each request (s_i, t_i, b_i),

$$s_i = (\lceil i/2 \rceil) \bmod n, \ t_i = (s_i + 1) \bmod n, \text{ and } b_{i,e} = 1 \ \forall \ e$$

For this problem, the hindsight optimum solution would be route each $s_i - t_i$ request along the edge (s_i, t_i) thus incurring a maximum edge load of 2.



(a) Hindsight optimum solution (b) Greedy algorithm after six requests

However, the greedy algorithm will route odd numbered requests (s_i, t_i) along the edge connecting them and the even numbered requests along the path $(s_i, (s_i - 1) \text{mod } n), ..., ((t_i + 1) \text{mod } n, t_i)$, incurring a maximum edge load of n.

$$\therefore \text{competitive ratio} = \frac{n}{2}$$

■

18.2 Online Algorithm with known hindsight optimum

Given a problem instance with $G = (V, E)$ and requests (s_i, t_i, b_i), let L^* be the hindsight optimum value. given a set of (s_i, t_i) paths P, define the potential of each edge as

$$\phi_e = \gamma^{l_e/L^*}, \text{ where } l_e = \sum_{i:P_i \in P, P_i \ni e} b_{i,e}$$

Define the potential of the paths P as

$$\phi = \sum_{e \in E} \phi_e$$

Algorithm 1 (Online Routing when hindsight optimum is known)

Given: $G = (V, E)$, requests (s_i, t_i, b_i) that arrive over time, and the known hindsight optimum value L^*.

1: Let $\gamma \leftarrow 3/2$, $P \leftarrow \emptyset$.
2: **while** request (s_i, t_i, b_i) arrives **do**
3: Compute ϕ based on the paths in P.
4: For each $s_i - t_i$ path P_i, compute ϕ_{P_i} based on the paths in $P \cup P_i$.
5: **pick** the path P_i^* such that $(\phi_{P_i} - \phi)$ is minimized.
6: $P \leftarrow P \cup P_i^*$
7: **end while**

Theorem 18.2.1 *Given an instance of the online routing problem and its hindsight optimum value, Algorithm 1 will ensure that at the end of the algorithm,*

$$\phi \leq 2m$$

where $m = |E|$.

Before proving the theorem, consider this corollary.

Corollary 18.2.2 *The competitive ratio of Algorithm 1 is* $O(\log n)$

Proof: From Theorem 18.2.1, $\phi \leq 2m$.

$$\therefore \max_e \phi_e \leq \phi \leq 2m \leq 2n^2$$

$$\therefore \max_e \left(\log(3/2) \cdot \left(\frac{l_e}{L^*} \right) \right) \leq \log(2n^2)$$

$$\therefore \max_e l_e \leq L^* \cdot \left(\frac{\log(2n^2)}{\log(3/2)} \right)$$

$$\therefore \text{competitive ratio} = \max \left(\frac{\max_e l_e}{L^*} \right) \leq O(\log n)$$

■

Let us now consider the proof of Theorem 18.2.1

Proof: We will denote the potential of an edge e after the arrival of request i with the symbol $\phi_e^{(i)}$, and the load on that edge with the symbol $l_e^{(i)}$. The total potential will be denoted by $\phi^{(i)}$. For a request (s_i, t_i, P_i), let P_i^* be the $s_i - t_i$ path in the hindsight optimum solution. Since at each step, Algorithm 1 greedily chooses the path which results in the least increase in the potential,

$$\begin{aligned}
\phi^{(i)} - \phi^{(i-1)} &\leq \text{ increase in potential if we choose path } P_i^* \\
&= \sum_{e \in P_i^*} \left((\gamma)^{\frac{l_e^{(i-1)} + b_{i,e}}{L^*}} - (\gamma)^{\frac{l_e^{(i-1)}}{L^*}} \right) \\
&= \sum_{e \in P_i^*} \gamma^{\frac{l_e^{(i-1)}}{L^*}} \left(\gamma^{\frac{b_{i,e}}{L^*}} - 1 \right)
\end{aligned}$$

Since we only consider $e \in P_i^*$, we know that $b_{i,e} \leq l_e^* \leq L^*$, where l_e^* is the load on edge e in the hindsight optimum solution. If we assume that $\gamma \geq 1$,

$$\left(\gamma^{\frac{b_{i,e}}{L^*}} - 1 \right) \leq \left(\frac{\gamma - 1}{L^*} \right) \cdot b_{i,e}$$

This follows from the fact that when $x \in [0, 1]$ and $a \geq 1$, $a^x - 1 \leq (a - 1) \cdot x$

$$\therefore \phi^{(i)} - \phi^{(i-1)} \leq \frac{\gamma - 1}{L^*} \sum_{e \in P_i^*} b_{i,e} \cdot \gamma^{\frac{l_e^{(i-1)}}{L^*}}$$

So, if t is the number of requests that arrive,

$$\phi^{(t)} - \phi^{(0)} \leq \frac{\gamma - 1}{L^*} \sum_{1 \leq i \leq t} \left(\sum_{e \in P_i^*} b_{i,e} \cdot \gamma^{\frac{l_e^{(i-1)}}{L^*}} \right)$$
$$= \frac{\gamma - 1}{L^*} \sum_{e \in E} \left(\sum_{i : P_i^* \ni e} b_{i,e} \cdot \gamma^{\frac{l_e^{(i-1)}}{L^*}} \right)$$

Since the load on each edge is cumulative, $l_e^{(i)} \leq l_e^{(t)}$ for all $i \leq t$.

$$\therefore \phi^{(t)} - \phi^{(0)} \leq \frac{\gamma - 1}{L^*} \sum_{e \in E} \left(\gamma^{\frac{l_e^{(t)}}{L^*}} \cdot \sum_{i : P_i^* \ni e} b_{i,e} \right)$$
$$= \frac{\gamma - 1}{L^*} \sum_{e \in E} \left(\gamma^{\frac{l_e^{(t)}}{L^*}} \cdot l_e^* \right)$$

Since $L^* = \max_{e \in E}(l_e^*)$, $l_e^* \leq L^* \ \forall \ e \in E$. So,

$$\phi^{(t)} - \phi^{(0)} \leq \frac{\gamma - 1}{L^*} \sum_{e \in E} \left(\gamma^{\frac{l_e^{(t)}}{L^*}} \cdot L^* \right)$$
$$= (\gamma - 1) \sum_{e \in E} \left(\gamma^{\frac{l_e^{(t)}}{L^*}} \right)$$
$$= (\gamma - 1) \cdot \phi^{(t)}$$
$$\therefore (2 - \gamma)\phi^{(t)} \leq \phi^{(0)}$$

. So, if we also assume that $\gamma \leq 2$ (i. e., $\gamma \in [1, 2]$),

$$\phi^{(t)} \leq \frac{\phi^{(0)}}{2 - \gamma}$$

Note that $\phi^{(0)} = \sum_{e \in E} \gamma^{\frac{l_e^{(0)}}{L^*}}$, where $l_e^{(0)} = 0 \ \forall \ e \in E$. So, $\phi^{(0)} = m$. Since in Algorithm 1, we set $\gamma = 3/2$, this means that at the end of the algorithm, $\phi \leq 2m$. ■

Note that in the proof of Theorem 18.2.1, we did not use the fact that L^* is equal to the hindsight optimum value. We only used the fact that L^* is at least equal to the hindsight optimum value. In fact, we can claim that

Theorem 18.2.3 *Given an instance of the online routing problem and an upper bound L^* on its hindsight optimum value, Algorithm 1 will ensure that at the end of the algorithm,*

$$\phi \leq 2m$$

where $m = |E|$.

18.3 Online Algorithm when hindsight optimum is unknown

We will now drop the assumption that the hindsight optimum value L^* is know beforehand. Consider the following algorithm.

Algorithm 2 (Online Routing)

Given: $G = (V, E)$, requests (s_i, t_i, b_i) that arrive over time.

Let $L^* \leftarrow 1$, $P \leftarrow \emptyset$.
while request arrives **do**
  For each $e \in E$, compute $l_e = \sum_{i:e \in P_i, P_i \in P} b_{i,e}$
  if $\max_{e \in E}(l_e) \leq c \cdot L^*$, where $c = \frac{\log(2m)}{\log(3/2)}$ **then**
    Compute ϕ for the paths in P
    pick the $s_i - t_i$ path P_i which minimizes the increase in ϕ
    $P \leftarrow P \cup P_i$
  else
    $L^* \leftarrow 2 \cdot L^*$
    $P \leftarrow \emptyset$
    Recompute ϕ based on the new value of L^* and P.
    pick the $s_i - t_i$ path P_i which minimizes increase in ϕ.
  end if
end while

Algorithm 2 proceeds in several phases. In each phase j, $L^* = 2^{j-1}$. Let us denote the paths picked by Algorithm 2 during stage j as $P^{(j)}$. Let us also denote the load on an edge e due to the paths picked during stage j as

$$l_e^{(j)} = \sum_{i:e \in P_i, P_i \in P^{(j)}} b_{i,e}$$

$$\therefore \max_{e \in E} l_e \leq \sum_{\text{stage } j} \max_{e \in E} l_e^{(j)}$$

since $l_e = \sum_j l_e^{(j)}$ for all $e \in E$.
Notice that during any phase j, only the paths in $P^{(j)}$ are considered in the computation of ϕ. So, we can consider the phase j of the algorithm as an instance of the online routing problem, where only the requests of that phase arrive. The hindsight optimum solution for this problem OPT'_j must be at most the hindsight optimal solution OPT for the original problem.
Let us consider the case where the algorithm enters the phase $k = \lceil (\log_2(OPT) + 1) \rceil$. During this phase k, $L^* \geq OPT$.
Since $L^* \geq OPT \geq OPT'_k$, from Theorem 18.2.3, we can conclude that

$$\max_{e \in E} l_e^{(k)} \leq c \cdot OPT'_k \leq c \cdot L^*$$

So, the algorithm never leaves this phase. Therefore, the algorithm terminates before reaching the phase $k+1$.

$$\begin{aligned}
\therefore \max_{e\in E}(l_e) &\leq \left(\sum_{j\leq\lceil\log_2(OPT)+1\rceil} \max_{e\in E} l_e^{(j)}\right) \\
&\leq \left(\sum_{j\leq\lceil\log_2(OPT)+1\rceil} c\cdot 2^{j-1}\right) \\
&= c\cdot 2^{\lceil\log_2(OPT)+1\rceil} \\
&\leq c\cdot 2^{\log_2(OPT)+2} \\
&= 4c\cdot OPT
\end{aligned}$$

$$\therefore \left(\frac{\max_{e\in E}(l_e)}{OPT}\right) \leq \log(2m)\cdot\frac{4}{\log(3/2)}$$

This proves the following theorem.

Theorem 18.3.1 *Given an instance of the online routing problem, Algorithm 2 will pick paths P_i, such that the competitive ratio is $O(\log n)$.*

18.4 Caching/Paging

In the general caching/paging problem, there are n pages of sizes s_i. The cache can store pages that have a total size of at most k. The cost of moving a page i from memory to the cache is c_i. A sequence of pages are requested. If a requested page is in the cache, it is called a *cache hit* and no cost is incurred. If a requested page is not in the cache, then it is called a *cache miss*. In the event of a cache miss, pages need to be cleared off the cache, and the requested page i must be moved to the cache. The goal of the problem is to minimize the total costs of moving pages to the cache.

18.4.1 Unweighted caching/paging

Let us consider the simpler version of the problem, where $s_i = 1$ and $c_i = 1$ for all pages $i \in [n]$. In this version of the problem, in the event of a cache miss, we have to move at most one page from the cache to make room for the new page, and our goal is simply to minimize the number of cache misses.
Caching/paging algoithms can be described by the policy used to evict pages from the cache in the event of a cache miss. Here are a few examples of *eviction policies* for the unweighted problem.

1. **LRU** (Least recently used): The page that that was requested the least recently is evicted.
2. **FIFO** (First-in first-out): The page that was requested the earliest is evicted.
3. **LIFO** (Last-in first-out): The page that was most recently requested is evicted.

Theorem 18.4.1 *No deterministic algorithm can have a competitive ratio better than k.*

Theorem 18.4.2 *LRU and FIFO have a competitive ratio of k.*

While deterministic algorithms cannot have a better competitive ratio than k, randomized algorithms perform much better. In fact,

Theorem 18.4.3 *1-bit randomized LRU has a competitive ratio of $H_k = \sum_{i \leq k} \left(\frac{1}{i}\right) \leq \log(k) + 1$.*

1-bit randomized LRU is a randomized eviction policy that works as follows. Like the LRU, for each page in the cache, it stores data about when it was last requested. However, unlike the LRU, the 1-bit randomized LRU uses only 1 bit to store this information. At the beginning, every page in the cache is assigned a value 0. When a page is requested, if it is already in the cache, its value is changed to 1. If not, then a page with value 0 is chosen uniformly at random from the cache and evicted. The requested page is moved to the cache, and it is assigned a value of 1. If there are no pages left in the cache with the value 0, then the values of all the pages in the cache are reset to 0, and the process continues.

Theorem 18.4.4 *No randomized algorithm can have a competitive ratio less than* $\log(k)$.

In fact, the competitive ratio of $\log(k)$ is tight even in the general caching/paging problem, and this can be proven using a primal-dual algorithm similar to the online set cover problem.

18.4.2 Offline unweighted caching

In order to prove Theorem 18.4.1, let us first consider the offline version of this problem. Consider the following eviction rule.

(Bellady's rule): Evict the page that is furthest in the future.

Fact 18.4.5 *Bellady's rule is the optimal eviction policy for the offline unweighted caching problem.*

We can also prove Theorem 18.4.1.

Proof: Fix some deterministic algorithm A. Let $n = k + 1$. The adversary follows the strategy of always requesting the page that is not in the cache. For this request sequence, if the adversary requests m pages, the cost incurred by algorithm A is m.

Let us consider the offline version of this problem, where $n = k + 1$. Note that because $n = k + 1$, after a page is evicted there will be no need to evict another page unless the evicted page is requested again. If we use Bellady's rule, then every time we evict a page, that means that the other $k - 1$ pages in the cache are requested before the evicted page is requested. So, there will be no need to evict another page for at least $k - 1$ further requests. Therefore, irrespective of the sequence of requests, evictions occur at most once every k requests. So, $OPT(\sigma) \leq m/k \ \forall \ \sigma$.

$$\therefore \text{ for any deterministic algorithm } A, \text{competitve ratio} = \min \frac{A(\sigma)}{OPT(\sigma)} \geq \frac{m}{m/k} = k$$

■