CS880: Approximation and Online Algorithms	**Scribe:** Luke Swanson
Lecture 21: Metrical Task Systems and Lower Bounds	**Date:** 11/08/2019

21.1 Caching Problems

Recall, in previous lectures, we discussed the *caching* problem. In this problem, we are given a cache of size k, i.e. the cache may hold up to k pages of data at once. We assume n pages of data in memory, with $n > k$, else the problem is trivial. The goal is to determine an eviction policy for pages in cache, which minimizes the total number of cache misses over an input sequence $\sigma \in [n]^T$.

We later noted a generalization of this problem, called *weighted caching*. In this variation, we allow the cost of a cache miss to vary by page so the cost of missing page i is c_i. The objective then becomes to minimize the cost of cache misses over the input sequence, rather than the total number of cache misses.
The original caching problem is then simply the weighted caching case where all weights are 1.

21.2 k-server Problem

A further generalization is the *k-server problem*, also covered in a previous lecture. Here, we have a metric space on n points, and k servers. Our input sequence is a series of requests $\sigma \in [n]^t$, and we service a request by moving one server to the point in the metric space corresponding to the request. Then the goal is to find a sequence of server moves minimizing the total movement costs.
For a uniform metric, where the distance is the same between all pairs of points, k-servers captures the caching problem.
If we have a star metric, then the k-servers problem captures weighted caching. This is as follows: Consider a star graph with n vertices connected to the central vertex v. All servers begin at the central vertex, and the edge cost from v to i is just $\frac{c_i}{2}$. When a page is brought into cache on a cache miss, we move one server to vertex i. When the page is evicted, we move the server back to v. Thus, for each cache miss, we eventually pay the full c_i, discounting k vertices at the start and end state, which may only have half the cost page. As this cost is only a constant additive term with respect to T (the length of σ), we can ignore it.

From the k-server problem, we have an associated *k-server conjecture*:
For any metric space, there exists a deterministic algorithm for the k-server problem with competitive ratio k.
While this is not a particularly complicated conjecture, it remains open. The strongest general result on metric spaces is a $2k - 1$ competitive ratio, and a competitive ratio of k has been shown for any metric with $n = k + 1$.
Since the introduction of randomized algorithms for caching, we have the *randomized k-server*

conjecture.
This states that for any metric space, there exists a randomized algorithm for the k-server problem that achieves an $O(\log k)$ competitive ratio.
Further investigation has looked at the relationship between competitive ratio and n, the size of the metric space. As noted in a previous lecture on embeddings in trees, if we can solve a problem efficiently on trees, we can perform a low-distortion embedding from a metric space to a tree metric. In particular, if we can solve a problem in polynomial time on trees, we can solve the embedded problem in poly($logn$). This approach was used to show poly($\log(k, n)$) competitive ratio on HSTs, which implies poly($\log(k, n)$) on general metrics. A later result showed a poly($\log(k)$) competitive ratio on HSTs, and $O(\log^6(k))$ competitive ratio for general metrics.This result used the mirror descent technique, which we will discuss in a later lecture.
Lastly, some work has been done on lower bounds. Specifically, there is a lower bound of $\Omega(\frac{\log k}{\log \log k})$ on any metric over $n \geq k + 1$ points. In other words, for non-trivial cases (i.e. $n \leq k$ so no server must ever move), we can not achieve a better competitive ratio.

21.3 Metrical Task Systems

Finally, we arrive at a broader problem, called a *metrical task system*. Here, we have a metric space on N points, and a single server. The request sequence becomes $\sigma \in (\mathbb{R}^{tN})^T$. To service a request, we must move the server to a location $i \in [N]$, and incur a service cost c_i^t, i.e. the cost to service request t from location i. We must minimize the total combined cost of movements and services. That is, $\min \sum^T m_i + c_i^t$.
Despite the fact that MTS has only a single server, it captures the k-server problem. Consider the N points in the metric space of MTS as all possible k-tuples of server configurations in the k-server problem. Whenever the k-server problem would require we move a server to location i, we set the MTS service cost to ∞ if the configuration represented by a point $j \in [N]$ does not contain i, else 0. The movement cost for MTS is then just the sum of costs to move servers to the new configuration in the k-server problem.
It should be noted, of course, that the size of the MTS space is exponential in the size of the k-server metric space.

21.3.1 Bounds

The first paper on metrical task systems showed lower and upper bounds of $2N - 1$. This was achieved with the work function algorithm, which we cover in detail in a later lecture.
Randomized algorithms are conjectured to be $O(\log N)$, and have been shown to be $\Omega(\log N)$. MTS is shown to be $O(\log N \lg \log N)$ on HSTs, implying $O(\log^2 N \log \log N)$, and a later result showed MTS is $O(\log N \cdot \text{diameter})$ on HSTs, leading to $O(\log^2 N)$ on general metrics.

21.3.2 Experts Problem

Related to this is the Experts problem, which again will be covered in greater detail in a later lecture. Here, we have n experts, and an incoming cost vector at each step. The algorithm must choose one expert, and pay that expert's cost. In the online version of the problem, the expert must be selected before seeing the cost vector for the given round.
In this Experts problem, we compete against the optimal location (optimal choice of expert) and perform a regret analysis, rather than a hindsight optimal choice. Also, note that the Experts problem typically does not include a movement cost, only a cost for each expert.
As such, algorithms for the MTS problem can often be useful for the less-general Experts problem. The remainder of our discussion will concertn lower bounds on deterministic and randomized algorithms, and an introduction to the work function algorithm.

21.4 Lower Bounds for Randomized Algorithms

21.4.1 Minimax Principle

First, we introduce Yao's Minimax Principle. This is a result from the study of 2-player zero-sum games. Suppose players in the game have some choice of moves to make, and a set strategy to win the game. Players are allowed to randomize their moves, and if they so choose, the result of the game will always be the same regardless of which player made the first move.
For analysis of online algorithms, we consider our algorithm to be the player making the first move, and an adversary as the second player.
Consider $\text{Cost}(R, x)$ for an algorithm R on input x. We wish to minimize the ratio $\frac{\text{Cost}(R,x)}{\text{Opt}(x)}$, while the adversary wants a maximum ratio. Then our optimization is the following: $\min_R(\max_x \frac{\text{Cost}(R,x)}{\text{Opt}(x)})$.
That is, for a worst-case choice of x (maximizing the ratio w.r.t. x), we choose the algorithm that minimizes w.r.t. R, given x.
On the other hand, if the adversary goes first, we get $\max_{x\ D}(\min_A \frac{E_{x\ D}[\text{Cost}(A,x)]}{E_{x\ D}[\text{Opt}(x)]})$.
That is, the adversary now must choose to randomize, because it was forced to go first. Hence, we choose an algorithm that minimizes the ratio, and the adverary chooses x from distribution D to maximize the ratio.
As the result of one player maximizing is independent of which player goes first, the minimax principle says that $\min_R(\max_x \frac{\text{Cost}(R,x)}{\text{Opt}(x)}) = \max_{x\ D}(\min_A \frac{E_{x\ D}[\text{Cost}(A,x)]}{E_{x\ D}[\text{Opt}(x)]})$.
Therefore, for any fixed R, D, $\max_x \frac{E[\text{Cost}(R,x)]}{\text{Opt}(x)} \geq \min_A \frac{E_{x\ D}[\text{Cost}(A,x)]}{E_{x\ D}[\text{Opt}(x)]}$

21.4.2 Lower Bound for Unweighted Caching

We will now show that the competitive ratio for unweighted caching is $\Omega(\log k)$.
Consider $n = k + 1$, with pages σ_i selected randomly from $[n]$, $\forall i$. For any deterministic online algorithm A, the probability that a given σ_i is not in the cache is just $\frac{1}{k+1}$. Hence, we have expected $\text{Cost}(A, \sigma) = \frac{|\sigma|}{k+1}$, the expected number of cache misses.
Now, consider the offline optimum, which evicts the page furthest in the future. With $n = k + 1$,

suppose that at time t, we evict a page that next occurs at time r. Clearly, the next $r-t$ elements of σ must be cache hits, since there is exactly one more page than spaces in cache. Further, we must have each of the other k pages appear at least once from times t to r, else that page did not occur until after the newly evicted page.
Finding the expected time between misses, $r-t$, is then just the coupon collector's problem for $k+1$ pages, and we have $E[r-t] = (k+1)H_{k+1}$. Then $E[\text{Opt}(\sigma)] = \frac{|\sigma|}{(k+1)H_{k+1}}$ over the entire sequence.
Substituting the expressions above, the competitive ratio is $\frac{E[\text{Cost}(A,\sigma)]}{E[\text{Opt}(\sigma)]} = \frac{|\sigma|}{k+1} \cdot \frac{(k+1)H_{k+1}}{|\sigma|} = H_{k+1}$.
For C.R. $= H_{k+1}$, we then have $\Omega(\log k)$.

21.4.3 Lower Bound for MTS

Next, we consider a lower bound for the MTS. Specifically, we will show the competitive ratio is $\Omega(\log N)$.
Consider a uniform metric over the N points in the metric space. Let $\sigma_t =$ location $i \in [N]$, chosen uniformly at random, and assign cost 1 to i, cost 0 to all $j \in [N], j \neq i$. Finally, let $d(i,j) = 1 \forall i \neq j \in [N]$. With these conditions, at each step we may either remain in place and pay a service cost of 1, or incur a movement cost of 1 and service cost 0.
Now, we have $E[\text{Cost}(A,\sigma)] = \frac{|\sigma|}{N}$, as there is a $\frac{1}{N}$ chance of incurring a movement cost on each step of σ. By a similar approach to the lower bound on unweighted caching, we consider gaps between consecutive "misses" for Opt.
The distances between misses are then of exponential length $N \log N$, and we have competitive ratio $\Omega(\log N)$.

21.4.4 Deterministic Algorithms

Finally, we discuss a few considerations for deterministic online algorithms.
Consider a greedy algorithm for the k-server problem. In some online algorithms, like unweighted caching, any choice carries equal weight, so a greedy algorithm is not applicable.
For k-servers, though, there may be a clear best choice at each step. Consider a simple graph with nodes A, B, C, where $d(A,B) = 1$, $d(B,C) = D$, 2 servers, and an initial configuration with one server at B, the other at C. Now, suppose the input sequence is $\sigma = ABABAB\ldots$. Then on the first input, the greedy choice is to move the server at B to A. In the following step, we move A back to B, etc. Then the cost for this input sequence is $|\sigma|$, which we can make as large as we like. On the other hand, the optimal choice is to move the server at C to A in the first step, incurring a cost of $D+1$ for the entire σ.
Clearly, then, a greedy algorithm is not a good choice, and has an unbounded ratio.
We shall briefly introduce a few other algorithms for k-servers, which will be discussed in detail in the following lectures.

21.4.4.1 Balance Algorithm

Here, we move all "closest" servers toward the request, until one server reaches the request. This algorithm works well on trees, but not so well on general metrics.

21.4.4.2 Work Function Algorithm

Given input sequence σ, let $\sigma^{(t)}$ denote the request sequence up to time t. Then define a work function $W^{(t)}(\alpha)$ = cheapest cost (service + movement) to service $\sigma^{(t)}$ and end in configuration α. Then the algorithm is as follows:
If at step $t-1$, algorithm is at configuration $\alpha^{(t)}$, then at step t, choose $\alpha^{(t)} = \text{argmin}_\alpha\{w^{(t)}(\alpha) + \alpha(\alpha^{(t-1)}, \alpha)\}$.