CS787: Advanced Algorithms

Scribe: Mayank Maheshwari and Priyananda Shenoy **Lecturer:** Shuchi Chawla
Topic: k-server Problem(continued); Online learning **Date:** 11/07/2007

22.1 Introduction and Recap

In the last lecture we were discussing the k-server problem, which is formulated as follows: Let M be a metric space with n points. There are k servers which can service requests. When a request occurs from a particular location, a server must be placed at that location. The problem is to minimize the total distance traveled by the servers. In this class we will see one algorithm for this problem, the Work Function Algoritm.

22.2 Work Function Algorithm

Let X be a configuration of the servers; i.e. the set of locations the servers are located. Then we define $OPT_i(X)$ as the optimal cost of serving requests $\sigma[1:i]$ and ending at configuration X. Then the Work Function algorithm, for servicing ith request, chooses the configuration X_i using the following rule:

$$X_i = \arg\min_X \{OPT_i(X) + d(X_{i-1}, X)\}$$

Theorem 22.2.1 *WFA is (2k-1) competitive*

Proof: We will be proving that for any sequence of requests σ

$$WFA(\sigma) \leqslant (2k-1)OPT(\sigma) + k^2 D$$

where D is the diameter of the metric space.

Let $X_0, X_1, ..., X_t$ be the configurations chosen by the WF Algorithm for servicing requests $r_0, r_1, ..., r_t$. We are interested in seeing how $OPT_i(X_i)$ evolves over time. So when the ith request r_i comes, we keep track of the difference in the optimal cost of servicing i requests and ending up at the configuration X_i chosen by our algorithm, and the optimal cost of servicing $i-1$ requests and ending up at the previous configuration X_{i-1}. This difference is given by

$$OPT_i(X_i) - OPT_{i-1}(X_{i-1})$$

Adding and subtracting $OPT_i(X_{i-1})$ we get,

$$(OPT_i(X_i) - OPT_i(X_{i-1})) + (OPT_i(X_{i-1}) - OPT_{i-1}(X_{i-1})) \tag{1}$$

The term $(OPT_i(X_{i-1}) - OPT_{i-1}(X_{i-1}))$ keeps track of how the optimal cost of ending at X_{i-1} changes due to the ith request. The term $(OPT_i(X_i) - OPT_i(X_{i-1}))$ tells us how our movement from X_{i-1} to X_i changes the potential function.

In the offline case, we used a dynamic programming approach to arrive at an optimal solution. The recurrence used by the DP was

$$OPT_i(X) = \min_{Y \ni r_i} OPT_{i-1}(Y) + d(X, Y)$$

Since $Y \ni r_i$,we have $OPT_{i-1}(Y) = OPT_i(Y)$. By taking $X = X_{i-1}$, we see that $Y = X_i$. This gives us

$$\begin{aligned} OPT_i(X_{i-1}) &= OPT_i(X_i) + d(X_i, X_{i-1}) \\ OPT_i(X_i) - OPT_i(X_{i-1}) &= -d(X_i, X_{i-1}) \end{aligned}$$

Substituting this in (1) we get,

$$(OPT_i(X_{i-1}) - OPT_{i-1}(X_{i-1})) - d(X_i, X_{i-1})$$

Summing over all the steps across the evolution,

$$\sum_i \{OPT_i(X_{i-1}) - OPT_{i-1}(X_{i-1})\} - \sum_i d(X_i, X_{i-1})$$

This sum is equal to the optimal cost of servicing all the requests and ending at the last configuration chosen by the WF Algorithm X_t. And $\sum_i d(X_i, X_{i-1})$ is nothing but the total cost incurred by our algorithm. Therefore we get

$$\begin{aligned} \sum_i \{OPT_i(X_{i-1}) - OPT_{i-1}(X_{i-1})\} - WFA(\sigma) = OPT_t(X_t) \\ \geqslant OPT(\sigma) \\ OPT(\sigma) + WFA(\sigma) \leqslant \sum_i \{OPT_i(X_{i-1}) - OPT_{i-1}(X_{i-1})\} \end{aligned}$$

To get a bound on the RHS expression, let us define the "extended cost" EXT_i as

$$EXT_i = \max_X \{OPT_i(X) - OPT_{i-1}(X)\}$$

Then

$$WFA(\sigma) + OPT(\sigma) \leqslant \sum_i EXT_i \tag{2}$$

We will now try to get a bound on EXT_i's. First we will try to get a bound on this term in the special case where $n = k + 1$, and then we will get a bound for the general case.

Claim 22.2.2 *if $n = k + 1$, then WFA the competitive ratio k*

We will prove this claim using a potential function argument. Let us define a potential function Φ as

$$\Phi_i = \sum_X OPT_i(X)$$

with $\Phi_0 = 0$. Let us consider the increase in the potential function value for a particular request i. From the way we defined EXT_i, this increase is atleast EXT_i.

$$\Phi_i - \Phi_{i-1} \geqslant EXT_i$$

Summing over all requests, we get

$$\Phi_t \geqslant \sum_i EXT_i \tag{3}$$

By definition we have,

$$\Phi_t = \sum_X OPT_t(X)$$

Since there are only $k + 1$ configurations possible, we can get an upper bound for the RHS value

$$\Phi_t \leqslant (k+1) \max_X OPT_t(X)$$

We know that $OPT(\sigma) = \min_X OPT_t(X)$. We can get an upper bound on $\max_X OPT_t(X)$ using the fact

$$\max_X OPT_t(X) \leqslant OPT(\sigma) + \max_{X,\acute{X}} d(X, \acute{X})$$

We know that $\max_{X,\acute{X}} d(X, \acute{X}) \leqslant kD$ where D is the diameter of the space. So we get

$$\Phi_t \leqslant (k+1)(OPT(\sigma) + kD)$$

Substituting this in (3)

$$\begin{aligned} \sum_i EXT_i &\leqslant \Phi_t \\ &\leqslant (k+1)OPT(\sigma) + (k+1)kD \end{aligned}$$

Using this in (2), we get

$$WFA(\sigma) \leqslant kOPT(\sigma) + (k+1)kD$$

Hence our algorithm is k-competitive for the case where $n = k+1$. We will now try to get a bound for the general case.

Claim 22.2.3 *For any values of k and n, WFA is (2k-1) competitive*

To show this, we will do the following construction. Let M be our metric space. Then we will construct a metric space $\overline{M}$ in the following manner. For every point a in M, we create a point $\overline{a}$, called the *antipode* of a , such that $d(a, \overline{a}) = D$. The distance between antipodes is equal to the distance between their corresponding original points; i.e. $d(a, b) = d(\overline{a}, \overline{b})$. The distance between a point and an antipode of another point is given by $d(a, \overline{b}) = D - d(a, b)$. It is easy to see that this construction yields a valid metric space.

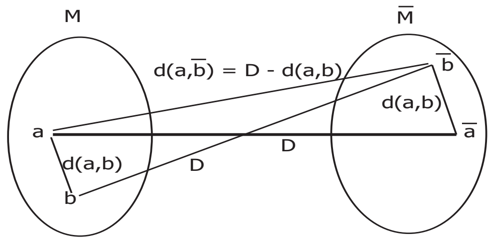

Fig 1: The metric spaces M and $\overline{M}$

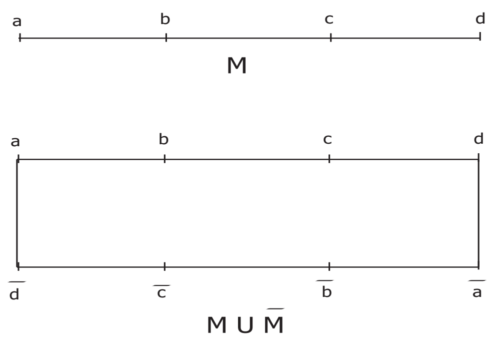

Fig 2: An example where all points lie on a line.

We transform the space M to $M \cup \overline{M}$. The problem doesn't change, since none of the newly added points issue requests.

We will consider the "worst case scenario" which maximizes the distance covered at step i. We will claim, and later give a proof sketch that this happens when a request comes from location r_i and all servers are at its antipode $\overline{r_i}$.

Claim 22.2.4 $arg \max_X \{OPT_i(X) - OPT_{i-1}(X)\} = (\overline{r_i}, \overline{r_i}, ..., \overline{r_i})$

Assuming that this claim holds good, we will proceed with obtaining an upper bound for $\sum_i EXT_i$.

Let $Y_i = (\overline{r_i}, \overline{r_i}, ..., \overline{r_i})$ be the "worst case" configuration. Then we have

$$\begin{aligned} EXT_i &= OPT_i(Y_i) - OPT_{i-1}(Y_i) \\ \sum_i EXT_i &= \sum_i OPT_i(Y_i) - \sum_i OPT_{i-1}(Y_i) \end{aligned} \tag{4}$$

Let us examine the sequence in which the optimal algorithm OPT services the requests.

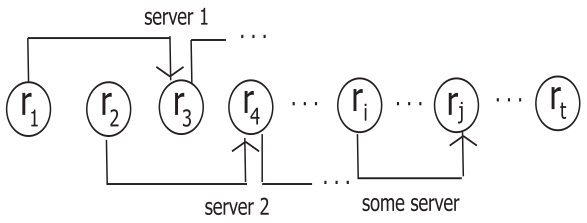

Fig 3: The sequence of how OPT services the requests

If OPT moves a server from r_i to r_j , we will match i with j. Then the following holds true

$$OPT_i(Y_i) \leqslant OPT_{j-1}(Y_j) + kd(r_i, r_j)$$

This works for most of the r_i's except the k r_i's at the end where the servers stop. For these servers

$$OPT_i(Y_i) \leqslant OPT + kD$$

Summing over all i

$$\begin{aligned}\sum_i OPT_i(Y_i) &\leqslant k(OPT + kD) + \sum_j \{OPT_{j-1}(Y_j) + kd(r_i, r_j)\} \\ &\leqslant kOPT + k^2 D + \sum_j OPT_{j-1}(Y_j) + k \sum_j d(r_i, r_j)\end{aligned}$$

but $\sum_j d(r_i, r_j) \leqslant OPT$ so

$$\begin{aligned}\sum_i OPT_i(Y_i) - \sum_j OPT_{j-1}(Y_j) &\leqslant kOPT + k^2 D + kOPT \\ \sum_i EXT_i &\leqslant 2kOPT + k^2 D\end{aligned}$$

Substituting this in (2)

$$\begin{aligned}WFA(\sigma) + OPT(\sigma) &\leqslant 2kOPT + k^2 D \\ WFA(\sigma) &\leqslant (2k-1)OPT + k^2 D\end{aligned}$$

Hence the Work Function Algorithm is $(2k-1)$-competitive.

■

We will now give a brief sketch of proving the claim we made in the general case analysis.

Proof of Claim 22.2.4(Sketch):

For simplicity, we will assume that there is only one server. Similar analysis can be done for multiple servers as well.

Suppose we get ith request r_i from location a. Let the previous request r_{i-1} be from location b. Since there is only one server, the configuration X is just a single location where the server is located, say x.

$$\begin{aligned}
&OPT_i(x) - OPT_{i-1}(x) \\
&= (OPT_{i-1}(b) + d(a,b) + d(a,x)) - (OPT_{i-1}(b) + d(b,x)) \\
&= d(a,x) - d(b,x) + d(a,b)
\end{aligned}$$

if $x = \overline{a}$ then,

$$d(a,\overline{a}) - d(b,\overline{a}) = D - (D - d(a,b)) = d(a,b)$$

So we see that the distance is maximized when the server was located at position $\overline{a}$.

■

22.3 Metric Task System Problem

The Metric Task system problem is a generalization of the k-server problem. The Work Function Algorithm is used to solve this problem and it gives a competitive ratio of $2N - 1$ where N is the number of points in the metric space.

Given: A Metric space M where $|M| = N$ and one server. M has a distance function d which satisfies the following properties:

$\forall x, y, z \in M$

- $d(x,y) \geq 0$
- $d(x,y) = d(y,x)$
- $d(x,y) \leq d(x,z) + d(z,y)$

The setting can be viewed as the server needs to execute tasks coming in a stream on machines located at points in the metric space M. A task can be well suited to a machine and so the cost of executing it at that point is low. We can also move the server to machine at another point in M which is suited to the task but we incur a cost in moving the task from one point to another. So we get a task and the server should execute it on best machine possible without incurring too much cost in moving from one machine to another.

Goal: The server needs to execute the incoming tasks on the machines located at the $|M|$ points such that the cost of executing those tasks is minimized.

To solve the *k-server problem* using metrical task system, we let each state of the system correspond to one possible configuration of the k-server problem. Therefore, the number of states is $\binom{N}{k}$. This is a more general problem and we will not delve into the intricacies of it. But the Work-Function Algorithm covered in the previous section can be used to solve the Metrical Task System problem and it gives a competitive ratio of $2N - 1$.

22.4 Online Learning

The online learning algorithms are a method to solve class of prediction problems. The problem is generally characterized by number of experts (say) n and every expert has a cost vector corresponding to the prediction associated with it. The objective is to maximize the cost associated with the cost vectors of an expert and do as well as the expert. The algorithm at every step needs to pick an expert and then the cost vector (consisting of 0's and 1's is revealed). This is similar to the Metrical Task System problem with a few differences,namely, there are two underlying assumptions for this class of problems:

1. It is free to move from one point (expert) to another i.e. the algorithm doesn't incur any cost.
2. The algorithm ALG is not compared with the optimal OPT in hindsight.

Goal: The algorithm should do nearly as well as the expert.

We will consider a few specific problems in this case. The first one is the Pick-a-Winner problem.

22.4.1 Pick-a-Winner Problem

In this problem, we get profits instead of costs and we need to maximize it.

Problem Statement: There are n different buckets and an adversary throws coins in the bucket. In this game, we need to pick a bucket and if the adversary oblivious of our choice throws the coin in that bucket, we get the coin. The game gets over when there are maximum d coins thrown in a bucket.

This is a case of oblivious adversary in which we take the help of randomization to work to our advantage. If we pick the buckets deterministically, the adversary would know our choice and would always throw the coins in the other buckets.

So if we pick the bucket uniformly at random, the expected number of coins that we get

$\mathbf{E}$[number of coins won]$= d \cdot \mathbf{Pr}$[picking a bucket u.a.r.] $= \frac{d}{n}$.

With this approach, we get a competitive ratio of $\frac{1}{n}$.

22.4.1.1 Multiplicative Updates Method(Exponential weights)

In this method of solving the Pick-a-Winner problem, we increase the probability of picking a bucket by a factor of 2 each time the adversary throws a coin in that bucket. So the algorithm can be seen as

Algorithm

- Start with, for all buckets i, $w_i = 1$ where w_i is the weight of bucket i.
- Whenever a coin falls into a bucket, double its weight.
- At every step, pick an i with probability proportional to w_i.

- Game ends when a bucket has d coins.

The adversary wants to end the game quickly and to distribute the coins over the n buckets so that the profit the algorithm achieves is not high.

Analysis

Let us say after t coin tosses, the total weight of the n buckets in the problem be $W_t = \sum_i w_i(t)$. Let the expected gain be F_t and the probability of success or winning the game is $\frac{w_i}{W_t}$ where i is the bucket we choose in round $t+1$. Now let us see how W_t evolves in the algorithm.

$W_{t+1} = W_t + w_i = W_t + W_t \cdot F_t$

So $W_{t+1} = W_t(1+F_t)$. Whenever there is a large increase in weight of the bucket, our gain is large.

At the end of the game i.e. when a bucket receives d coins, the W_{final} can be written as, where $\sum_t F_t$ is our total expected gain:

$2^d \leq W_{final} = n\prod_t(1+F_t)$

Taking natural logs on both sides,

$$\begin{aligned} d\log_e 2 &\leq \log_e n + \sum_t \log_e(1+F_t) \\ &\leq \ln n + \sum_t F_t \qquad [\ln(1+x) \leq x] \end{aligned}$$

This implies $\sum_t F_t \geq d\ln 2 - \ln n \approx 0.7d - \ln n$.

We can see that the algorithm gives a profit with a factor close to the maximum profit but we lose an additive term. To minimize the additive loss in our profit, we can modify the algorithm in a way that instead of doubling the weight of a bucket each time a coin falls in it, we increase the weight by a factor of $(1+\epsilon)$. So we can rewrite all the equations in the analysis with the factor of ϵ introduced. So now $W_{t+1} = W_t(1+\epsilon F_t)$ and $(1+\epsilon)^d \leq W_{final} = n\prod_t(1+\epsilon F_t)$

$$\begin{aligned} d\log(1+\epsilon) &\leq \log n + \sum_t \log(1+\epsilon F_t) \\ &\leq \log n + \sum_t \epsilon F_t \end{aligned}$$

$$\begin{aligned} ALG = \sum_t F_t &\geq \frac{d}{\epsilon}\log(1+\epsilon) - \frac{1}{\epsilon}\log n \qquad [\log(1+\epsilon) \geq \epsilon - \frac{\epsilon^2}{2}] \\ &\geq (1-\frac{\epsilon}{2})d - \frac{1}{\epsilon}\log n \end{aligned}$$

Now if $\epsilon = \sqrt{\frac{\log n}{d}}$, then the $ALG \geq d - \frac{3}{2}\sqrt{d\log n}$.

The additive factor which we lose here is known as the *regret bound* or the *regret of algorithm* which is the factor the algorithm looses in order to converge to the profit of the expert. We will see more on the regret bounds in the next lecture.

References

[1] Yair Bartal Lecture Notes on Online computation and network Algorithms. *http://www.cs.huji.ac.il/ algo2/on-line/on-line-course.html*.