Advanced Algorithms 11/21, 2017

Lecture 10

Lecturer: Mohsen Ghaffari *Scribe: Frederik Benzing*

This draft has not been checked by the lecturer yet.

1 Online Algorithms & Competitive Analysis 2

1.1 The k-Server Problem

Problem description: For the general k-server problem we are given a metric space (V, d) on n points and a sequence of requests $r_1, r_2, \ldots \in V$. We have k servers, which are placed on points in V and each time a new request r_i arrives, one of the k servers has to move to r_i. The cost of such a move is given by the distance the server travels in (V, d).

Observe that this is a generalization of the paging problem considered in the previous lecture. To see this simply take the uniform 1-metric on the space of pages and let the positions of the servers correspond to the pages currently held in the cache.

In fact, several important problems can be rewritten as a k-server problem, which is why it has attracted some attention. It was introduced by Manasse, McGeoch and Sleator[MMS90]. Fiat, Rabani, and Ravid obtained a deterministic $2^{O(k \log k)}$ competitive algorithm/analysis[FRR90]. This was improved by Grove to $2^{O(k)}$[Gro91]. Then came a breakthrough by Koutsoupias and Papadimitriou[KP95], which obtained competitiveness ratio of $2k-1$. This is still the best known, result even though the algorithm is conjectured to be k-competitive. When turning to randomized algorithms, currently the best result was obtained by Bansal et al.[BBMN11] and is poly$(\log n, \log n)$ competitive (it uses metric tree embeddings seen in a previous lecture).

k-server Problem for Line Metrics: We will only consider a special case of the k server problem, where we are given a line-metric, i.e. $V \subset \mathbb{R}$ where $\mathbb{R}$ has the usual metric. It can easily seen that the natural greedy approach, where we always move the server closest to the new request to the request can be very bad. Indeed, consider $V = \{0, 2, 3\}$ and assume that all servers start at 0. If the requests now alternate between 2,3, then the greedy approach has cost linear in the number of requests, whereas the optimal solution only has constant cost.

Instead we propose the following k competitive 'Double Coverage' Algorithm introduced by []. Label the servers $s_1, \ldots, s_k$ and suppose that s_i is at position x_i. it is easily seen that any optimal algorithm can be adapted so that the servers never change their order so we may assume without loss of generality $x_1 \leq \ldots \leq x_k$.

Double Coverage Algorithm: Suppose we receive a request r. If $r \leq x_1$ then s_1 answers the request, similarly if $r \geq x_k$ then s_k answers the request. Otherwise, we can find some $1 \leq i < k$ so that $x_i \leq r \leq x_{i+1}$. We now let both servers s_i, s_{i+1} move towards r at the same speed. Once the first server reaches r both servers stop[1].

Before analyzing this algorithm, we remark that it can be easily generalized to tree metrics. If a new request appears then we move all servers s for which there is no other server on the shortest path from s to r towards r at the same speed (and we stop every server s once another server appears on the shortest path from s to r in this process). The analysis for this algorithm is essentially the same as the one we now present.

Analysis of Double Coverage: We will show that our algorithm is k competitive. By OPT and ALG we denote the current cost of the optimal solution and our double coverage algorithm, respectively. We mostly oppress time/round dependence of these (and other) values. Suppose that the servers of

[1]In principle, the server that does not answer the request might end up outside V, however we may just virtually update that server's position without actually moving it and then continue our algorithm. This only decreases the cost of our algorithm, since we might avoid some zig-zag moves.

double coverage are at positions $x_1 \leq \ldots \leq x_k$ and the servers of OPT are at positions $y_1 \leq \ldots \leq y_k$. We define a potential function

$$\Phi = \underbrace{k\sum_{i=1}^{k} |x_i - y_i|}_{=:\Phi_1} + \underbrace{\sum_{i<j} x_j - x_i}_{=:\Phi_2} .$$

Intuitively, the first term Φ_1 accounts for how far ALG is from OPT, while the second term Φ_2 accounts for how much ALG wins by its double coverage. It is easily seen that $\Phi_0 = 0$ and $\Phi \geq 0$ at any time. So if suffices to show

$$ALG + \Phi \leq k \cdot OPT \tag{1}$$

to prove k-competitiveness. In proving this inequality, we suppose that a new request r arrives and show that it holds after (only) OPT moves and then show that it also still holds after ALG moves. Say OPT moves server s_i to answer r by x (we may assume that only the server answering the request moves, since we can delay the moves of all other servers until they actually answer a request and we may also assume that the servers do not change their order). The the cost of OPT increases by x. At the same time Φ_1 increases by at most kx while Φ_2 stays the same. So (1) still holds after OPT's move. Now we consider ALG's move. First assume that $r \leq x_1$. In this case, ALG has cost $x = x_1 - r$. Moreover, Φ_2 increases by $(k-1)x$. Finally, since in OPT some server is at position r, we know $y_1 \leq r$ so that Φ_1 decreases by kx, so that (1) still holds after ALG's move. The case $r \geq x_k$ is analogous, so it only remains to consider the case where $x_i \leq r \leq x_{i+1}$ for some $1 \leq i < k$. Let us without loss of generality assume that x_i is closer to r than x_{i+1}, so that s_i moves $x := r - x_i$ to the right, while x_{i+1} moves x to the left. This move has cost $2x$ for ALG. Moreover, Φ_2 decreases by $2x$ (the $x_{i+1} - x_i$ decreases by $2x$ while for all $j \neq i, i+1$ the changes in $x_j - x_i$ and $x_j - x_{i+1}$ cancel). So to establish that (1) still holds it only remains to show that Φ_1 does not increase. To see this, note that some server of OPT is at position r. This implies that $y_i \geq r$ or $y_{i+1} \leq r$. So at least one of x_i and x_{i+1} moves closer to its 'partner' y_i, y_{i+1}, respectively. This decreases Φ_1 by kx while the move of the other server can increase Φ_1 by at most kx finishing the analysis.

1.2 Online Routing of Virtual Circuits

Problem Statement: We are given a graph $G = (V, E)$ on n vertices and m edges together with some edge capacities $u_e > 0$ for all $e \in E$. Now several requests requests of the form $(s_i, t_i, p(i))$ arrive in an online manner, where $s_i, t_i \in G$ and $p(i) \in \mathbb{R}_+$. For each such request, we have to send a flow of $p(i)$ from s_i to t_i along some path P_i in G. Our aim is to choose these paths P_i to minimise the maximum congestion load $\max_{e\in E} \ell_e$, where the congestion load $\ell_e$of an edge e is defined as

$$\ell_e = \frac{\sum_{P_i \ni e} p(i)}{u_e}.$$

Below, we descibe an online algorithm by Fiat et al. which is $O(\log n)$ competitive, it uses a technique know as 'multiplicative weight update' which can be used in many other contexts, too.

Algorithm Description: For the description and analysis we assume that we know the correct value Λ for the maximum congestion of (offline) OPT. We indicate how to get rid of this assumption after the analysis.
We first need some notation: Write $p_e(i) = p(i)/u_e$ so that the congestion of an edge after j rounds is given by

$$\ell_e(j) = \sum_{i \leq j : e \in P_i} p_e(i).$$

We use '*' to denote corresponding quantities of the (offline) OPT, e.g. $l_e^*(j)$ is the congestion of e after j rounds in OPT. Additionaly, we use tilde to denote quantities which are normalized by a factor of Λ, e.g. $\tilde{\ell}_e(j) = \ell_e(j)/\Lambda$. We define the cost of an edge e to be $c_e = a^{\tilde{\ell}_e(i)+\tilde{p}_e(i)} - a^{\tilde{\ell}_e(i)}$, where a is a constant to be determined later. In the resulting graph, we simply route request $i+1$ through the shortest s_{i+1}, t_{i+1} path. Note that the cost function makes edges which are already highly-congested very expensive so that they are less likely to get even more congested.

Analysis: As mentioned before, we want to show that our algorithm is $O(\log n)$ competitive. To this end, we need to show $\max_{e\in E}\tilde{\ell}_e = O(\log n)$.
We define the following potential function

$$\Phi(j) = \sum_{e\in E} a^{\tilde{\ell}_e(j)}(\gamma - \tilde{\ell}^*_e(j)),$$

where $\gamma > 1$ is another constant to be determined. Note that by our normalisation, we have $\tilde{\ell}^*_e(j) \leq 1$ implying $\gamma - \tilde{\ell}^*_e(j) = \Omega(1)$.
We will show that Φ is non-increasing for the right choice of a and γ. Togehter with the simple bound $\Phi(0) \leq \gamma m$, this shows that we have $\Phi(j) \leq \gamma m$ at all times j. This easily implies $\tilde{\ell}_e(j) = O(\log n)$ (assume otherwise, plug the value of $\tilde{\ell}_e(j)$ into the definition of $\Phi(j)$ to see $\Phi(j) > \gamma m$ as a contradiction).
We now show that Φ is non-increasing, i.e. $\Phi(j+1) \leq \Phi(j)$ for all times j. We explain how to obtain the inequalities below right after they are displayed.

$$\begin{aligned}
\Phi(j+1) - \Phi(j) = & \sum_{e\in P_{j+1}} \left(a^{\tilde{\ell}_e(j+1)} - a^{\tilde{\ell}_e(j)}\right)(\gamma - \tilde{\ell}^*_e(j)) - \sum_{e\in P^*_{j+1}} a^{\tilde{\ell}_e(j+1)}\tilde{p}_e(j+1) \qquad (2)\\
\leq & \sum_{e\in P_{j+1}} \gamma\left(a^{\tilde{\ell}_e(j)+\tilde{p}_e(j+1)} - a^{\tilde{\ell}_e(j)}\right) - \sum_{e\in P^*_{j+1}} a^{\tilde{\ell}_e(j)}\tilde{p}_e(j+1)\\
\leq & \sum_{e\in P^*_{j+1}} \left(\gamma\left(a^{\tilde{\ell}_e(j)+\tilde{p}_e(j+1)} - a^{\tilde{\ell}_e(j)}\right) - a^{\tilde{\ell}_e(j)}\tilde{p}_e(j+1)\right)\\
= & \sum_{e\in P^*_(j+1)} a^{\tilde{\ell}_e(j)}\left(\gamma\left(a^{\tilde{p}_e(j+1)} - 1\right) - \tilde{p}_e(j+1)\right)
\end{aligned}$$

The second line follows from $\tilde{\ell}^*_e(j) \geq$ and $\tilde{\ell}_e(j+1) \geq \tilde{\ell}_e(j)$. The third line uses precisely that the cost w.r.t. c_e (as defined in the description of the algorithm) of the path P_{j+1} is at most the cost of P^*_{j+1} (since we chose P_{j+1} as a minimiser of this cost).
To finish, we recall $0 \leq \tilde{p}_e(j+1) \leq 1$ and note that for example choosing $\gamma = 2$ and $a = 1 + 1/\gamma$ gives the inequality $\gamma(a^x - 1) \geq x$ for $0 \leq x \leq 1$. Using this inequality in the last line of (2) for $x = \tilde{p}_e(j+1)$ shows that $\Phi(j+1) - \Phi(j) \leq 0$ which finishes the analysis.

What about Λ? It remains to discuss how to drop the assumption of knowing the congestion load of OPT. Suppose that the algorithm described above is $\beta = \beta(n)$ competitive (we can clearly use the above analysis to find some explicit function $\beta = O(\log n)$, so that ALG is β competitive). We will run the algorithm in several phases. We start running our algorithm for the following guess $\Lambda = \min_e p_e(1)$, which is a lower bound on the congestion of OPT. If in any round j some $\tilde{\ell}_e(j)$ becomes larger than $\beta\Lambda$, we start a new phase. This means that we multiply our current guess of Λ by 2 and that we reset the current load of every edge to 0 (so we forget about any previous paths.) and that we retry routing request j.

One can check now that the resulting algorithm is 4β competitive. Indeed, suppose that there were T phases in total and consider our final guess Λ (after the last phase has ended) and note that $\Lambda \leq 2OPT$. This inequality is based on the observation that the algorithm above works whenever the congestion load of OPT is at most $OPT \leq \Lambda$, thus a phase only ends when $OPT > \Lambda$ (before updating Λ). Note also that the inequality also holds if there is only one phase by our initial guess of Λ). Finally, for any $i \leq T$, the congestion at the end of the i-th phase of ALG was at most $\beta\Lambda/2^{T-i}$. So all together, the phases have a congestion of at most $\sum_{i=1}^{T}\beta\Lambda/2^{T-i} \leq 2\beta\Lambda$ finishing the proof.

1.3 Online Bipartite Matching

A very similar problem as described below is what arises when for example search engines try to match certain ads to certain requests.

Problem Statement: We are given a bipartite graph on (disjoint) vertex sets V, U. For each vertex $u \in U$ its neighborhood in V is revealed in an online-fashion. Whenever a new vertex u is revealed we have to decide whether we keep it and match it to one of its unmatched neighbors in V (assuming such a neighbor exists) or discard it. Our decisions are irreversible and our goal is to find a matching as large as possible.

It is clear that OPT can just calculate a maximum matching. Using the fact that each maximal matching has size at least half of a maximum matching, we can see that a greedy approach for the Online Bipartite Matching Problem gives a $1/2$ competitive strategy. This is optimal for deterministic algorithms.

Karp, Vazirani, and Vazirani [] used a randomized algorithm to give a $(e-1)/e \simeq 0.63$ competitive algorithm, they also showed that this is asymptotically best possible. We describe and analyze their algorithm below, the analysis follows Birnbaum, Mathieu [].

Algorithm Description: Order V uniformly at random. When a new vertex $u \in U$ is revealed, we match it to the smallest unmatched vertex in V.

Analysis: We first observe that we may assume without loss of generality that OPT is a perfect matching. Indeed, fix some OPT (i.e. a maximum matching). We only need to show that removing a vertex x does not make the matching obtained by ALG larger. To see this, denote the matching obtained by ALG before removing x by M and the one obtained after removing x by M'. If M and M' are not identical, then x must have been matched in M and one can easily see that M and M' only differ by an alternating path starting at x (using edges from M and M' alternatingly). This already gives $|M| \geq |M'|$.
Let us now fix some maximum (perfect) matching M^*, for any vertex x we will denote by $M^*(x)$ the vertex it is matched to by M^*. Moreover, let σ be the random permutation (ranking) chosen by ALG, and let $M = M(\sigma)$ be the matching constructed by our algorithm. Let x_t be the probability that the vertex at rank t is matched by M. Then clearly the expected size of M is $\sum_{i=1}^{n} x_t$ where n is the size of both vertex classes U, V.

Observation: Let $v \in V$ and $u = M^*(v)$. It is easily seen that if v is unmatched in M, then u must be matched to some vertex v' of lower rank than v, i.e. $\sigma(v') < \sigma(v)$.

We now need a slightly technical Lemma in order to deal with some dependencies.

Lemma 1. *Let u and v be as before, and let σ' be a permutation of V, let σ_i be the permutation obtained from σ' by moving v to rank i. If v is not matched by $M' = M(\sigma')$, then u is matched by $M_i = M(\sigma_i)$ to some vertex v_i with rank $\sigma_i(v_i) \leq \sigma'(v)$.*

Proof. We write $v' = M'(u)$ and note that for any $w \in V$, we have by noting $\sigma_i(w) - 1 \leq \sigma'(w)$.
If M' and M_i match u to the same vertex $v_i = v'$, then the claim follows from the observation just before the lemma since $\sigma_i(v') - 1 \leq \sigma'(v') < \sigma'(v)$.
Otherwise, similarly to before, the two matchings M', M_i differ by an alternating path starting at v and going through v_i and v'. It is not hard to check that this path is increasing in σ_i, so writing $v_i = M_i(u)$, we get $\sigma_i(v_i) \leq \sigma_i(v')$. The claim follows from $\sigma_i(v_i) \leq \sigma_i(v') \leq \sigma'(v') + 1 \leq \sigma'(v)$ using the observation just before the Lemma. $\square$

We will, in a second, use Lemma 1 to establish $1 - x_t \leq (1/n)\sum_{i=1}^{t} x_i$, but let us first show how this inequality finished the proof. Writing $S_t = \sum_{i=1}^{t} x_i$ and subtracting S_{t-1} from both sides gives $1 - S_t \leq (1/n)S_t - S_{t-1}$ and rearranging yields $S_t \geq (1 - 1/(n+1))(1 + S_{t-1})$. These expressions for S_t are minimized when equality holds throughout. Noting $S_1 = x_1 = 1$, we obtain $S_t = \sum_{i=0}^{t-1}(1-1/(n+1))^i$ leading to $S_n = (n+1)\big(1 - (1 - 1/(n+1))^n\big)$ proving that the algorithm is $1 - 1/e$ competitive as n tends to infinity.
It remains to show $1 - x_t \leq (1/n)\sum_{i=1}^{t} x_i$. First, we denote by R_t the set of vertices in U which are matched to some vertex of rank at most t. Linearity of expectation gives $\text{Exp}[|R_t|] = \sum_{i=1}^{t} x_t$.
Given our permutation σ, let σ' be obtained from σ by taking an element $v \in V$ uniformly at random and moving it to rank t. Consider $M' = M(\sigma')$ and let $u = M^*(v)$. By choosing i appropriately, we can achieve $\sigma_i = \sigma$ (using notation from Lemma 1). So if v is not matched in M', which happens with probability $1 - x_t$, then we have by Lemma 1 that u is matched by $M = M(\sigma)$ to some vertex with rank at most t, meaning $u \in R_t$. We also have $\Pr[u \in R_t] = |R_t|/n$ (note that, crucially, R_t depends on t and σ and that our random choice u is independent of those). So we get $1 - x_t \leq |R_t|/n$ and taking expectations gives the desired result.

References

[BBMN11] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 267–276. IEEE, 2011.

[FRR90] Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 454–463. IEEE, 1990.

[Gro91] Edward F Grove. The harmonic online k-server algorithm is competitive. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 260–266. ACM, 1991.

[KP95] Elias Koutsoupias and Christos H Papadimitriou. On the k-server conjecture. *Journal of the ACM (JACM)*, 42(5):971–983, 1995.

[MMS90] Mark S Manasse, Lyle A McGeoch, and Daniel D Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.