Approximation Algorithms Workshop June 13-17, 2011, Princeton

Lecture Online Algorithms and the k-server problem — June 14, 2011

Joseph (Seffi) Naor *Scribe: Mohammad Moharrami*

1 Overview

In this lecture, we first discuss the paging/caching problem. Then, we continue by defining the k-server problem and showing what has been done in the past to find competitive algorithms for this problem.

In the next section we move on to linear programming. We explain how one can use a primal-dual approach to achieve an competitive algorithm for (online) Covering/Packing LP-s. Later, we use this approach to design an algorithm for the allocation problem, the main subroutine used for solving the k-server problem.

Finally, we wrap-up this lecture with our new result which achieves a competitive factor of $\tilde{O}(\log^2 k \log^3 n)$ for the k-server problem on n nodes[1]. This is joint work with Nikhil Bansal, Niv Buchbinder, and Aleksander Madry. Our basic approach is based on the framework of [CMP08]. The key idea in our work is that instead of using the allocation problem (as suggested by [CMP08]), we use the fractional allocation problem to construct a fractional algorithm for the k-server problem. It is fairly easy to convert the fractional (deterministic) algorithm into a randomized (integral) algorithm.

2 Online Algorithms and the k-Server Problem

2.1 Paging/Caching Problem

Paging/Caching Problem: We have a universe of n pages, a cache of size $k \ll n$, and requests for pages arrive over time. If a requested page is already in the cache there is no penalty for fetching that page. Otherwise, we have to bring that page into the cache and pay the loading cost for that page. The main question is "which page should we evict from the cache when the cache is full?"

The goal of a paging algorithm is to minimize the number of cache misses. For $\alpha \geq 1$, we say that a paging algorithm is *α-competitive* if the total cost that we have to pay is less than α times the optimal algorithm[2].

The problem of minimizing the number of cache misses was studied in [ST85] by Sleator and Tarjan.

Theorem 1. *[ST85] LRU (Least Recently Used) algorithm is k-competitive.*

They also showed that their algorithm is optimal in the sense that any paging algorithm is at least k-competitive.

[1] We use $\tilde{O}$ to hide $\log \log n$ terms.

[2] We assume that the optimal algorithm knows the sequence of page requests in advance.

The algorithm and the lower bound from [ST85] were deterministic and one can ask if using randomness can help us to achieve a better bound or not. This question was answered positively in [FKL$^+$91].

Theorem 2. *[FKL$^+$91] There is a randomized (Marking) algorithm that is $O(\log k)$-competitive.*

They also showed that any randomized algorithm is $\Omega(\log k)$-competitive.

2.1.1 k-Server Problem

The k-server problem is a generalization of the paging problem that was first defined in [MMS88].

k-Server Problem: We are given a metric space (X, d) and k servers. We receive requests for servers over time. Each request is a point in the metric that needs a server, and the question is which server should we send to that point. The goal in the k-server problem is to minimize the total distance travelled by servers.

Example: We have k fire trucks on a line and we receive requests when there is a fire at some point on the line. The goal is to minimize the average response time.

The paging problem is a special case of the k-server problem where the metric is uniform, and each page corresponds to a point in the metric. The lower bounds of k and $\Omega(\log k)$ still hold for deterministic and randomized versions of the k-server problem. It was conjectured in [MMS88] that there are deterministic and randomized algorithms that meet these lower bounds.

Conjecture[MMS88]: There are k-competitive deterministic and $O(\log k)$-competitive randomized algorithms for the k-server problem on any metric space.

The current best deterministic algorithms that is known for the k-server problem is $(2k-1)$-competitive algorithm from [KP94]. On the randomized side, there is an $O(\log k)$-competitive algorithm for weighted paging from [BBN07]. However, the problem of finding an $o(k)$-competitive algorithm for general metric spaces is still open. Even for trees of depth two or line metric there are no $o(k)$-competitive known, and the best known algorithm for the line metric is $\exp(O(\sqrt{\log n}))$-competitive from [BBN10] which improved the bound $O(n^{2/3})$ from [CL06].

2.2 Online Packing and Covering Problems

Covering Problem: In a covering problem the objective is to minimize the total cost given by a linear cost function $\sum_{i=1}^{n} c_i x_i$, and the feasible solution space is defined by a set of m linear constraints of the form $\sum_{i=1}^{n} a_{i,j} x_i \geq b_j$, where the entries $a_{i,j}$ and b_j are non-negative. Moreover, in the solution variables x_i must be non-negative.

We can write a linear program for the covering problem as follows.

Primal:	
min :	$\sum_{i=1}^{n} c_i x_i$
subject to:	
$\forall j, 1 \leq j \leq m$:	$\sum_{i=1}^{n} a_{i,j} x_i \geq b_j$
$\forall_i, 1 \leq i \leq n$:	$x_i \geq 0$

The dual of this linear program describes a packing probelm.

Dual:	
max :	$\sum_{j=1}^{m} b_j y_j$
subject to:	
$\forall i, 1 \leq i \leq n$:	$\sum_{j=1}^{m} a_{i,j} y_j \leq c_i$
$\forall_j, 1 \leq j \leq m$:	$y_i \geq 0$

Online covering problem: In the online version of the covering problem, constraints arrive one by one, and at each step we are only allowed to increase the value of variables. In the dual setting variables arrive one by one and variables must be set upon arrival.

Now, we show how one can obtain an approximation algorithm for the online cover problem. Our algorithm is based on [BN09]. The key idea for finding an approximate solution is using exponential update. At step t we receive a new constraint

$$\sum_{i=1}^{n} a_i x_i \leq b_t.$$

After receiving the new constraint, we have to add a new variable y_t to the dual and add the term $b_t y_t$ to the objective function in dual. We also need to modify the constraints in the dual by adding $a_i y_t$ term to the ith constraint in the dual. We update the variables in primal and dual such that

$$\frac{dx_i}{dy_t} = \frac{a_i x_i}{c_i}.$$

The value of $\frac{dx_i}{dy_t}$ is proportional to x_i, therefore x_i varies as an exponential function of y_t. We increase the value of y_t until the new constraint in primal is satisfied. We have

$$\begin{aligned}
\frac{d(\sum_{i=1}^{n} c_i x_i)}{dy_t} &= \sum_{i=1}^{n} c_i \left(\frac{a_i x_i}{c_i}\right) \\
&= \sum_{i=1}^{n} a_i x_i \\
&\leq b_t \\
&\leq \frac{d(\sum_{j=1}^{t} b_i y_i)}{dy_t}
\end{aligned}$$

This inequality guarantees that the objective function in the dual grows faster than the objective function in the primal. Let P be the value of the solution for our primal, D be the value for the dual, and OPT be the value of the optimal solution.

It is possible to show that $\left\{\frac{y_i}{\log n}\right\}_{1\leq i\leq t}$ is a feasible solution for the dual (see [BN09] for details). We have

$$\frac{P}{\log n} \leq \frac{D}{\log n} \leq OPT \leq P \leq D,$$

therefore our algorithm is $O(\log n)$-competitive for the covering problem.

This generic idea and algorithm is applicable to many online problems including: Ski Rental Problem, Dynamic TCP-acknowledgement, Online Matching.

In general linear programming helps us find the difficulties of the online problem and gives us a general recipe for both design and analysis of online algorithms via duality.

2.3 Cote, Meyerson and Poplawski [CMP08] Approach to k-Server Problem

Hierarchically Separated Trees: We call a balanced rooted tree α-HST if length of all edges at height h is α^{-h}.

It is known that any metric space can be embedded into distribution of α-HSTs while incurring distortion $O(\log n)$[FRT03] (vertices are mapped to the leaves of the tree). Given a c-competitive algorithm for α-HSTs, we can use this result to construct an $O(c\alpha \log n)$-competitive algorithm for general metric spaces.

Cote, Meyerson and Poplawski [CMP08] used this idea and tried to solve the k-server problem on HSTs instead of general metric spaces. Their main tool in constructing the algorithm for HSTs was the allocation problem on a uniform metric. The solution to the allocation problem was used to decide how to distribute servers among leaves of tree.

2.3.1 Allocation Problem

Suppose that we have n nodes and k servers. At step t in the allocation problem a request vector

$$(h_t(1), \dots h_t(k))$$

arrives at some node i, and it is guaranteed that $h_t(1) \geq h_t(1) \dots \geq h_t(k)$. Upon receiving the request, we can move servers between nodes. The total cost that we have to pay is sum of the moving cost and the hit cost. The hit cost is given by $h_t(k_i)$, where k_i is the number of servers at location i. Our goal in the allocation problem is to minimize the total cost

$$\text{total cost} = \text{moving cost} + \text{hit cost}.$$

Note that, the paging problem is an instance of the allocation problem where request vectors are of the form

$$(\infty, 0, \dots, 0).$$

Theorem 3. *[CMP08] If there is an algorithm for the allocation problem such that*

1. *hit cost* $\leq (1+\epsilon)OPT$,

2. *move cost* $\leq \beta(\epsilon)OPT$,

then there exists an $O(\Delta\beta(\frac{1}{\Delta}))$*-competitive algorithm for k-server problem on HSTs, where* Δ *is depth of the tree.*

Using this theorem we only need to find a $\beta = O(\mathsf{poly}(\frac{1}{\epsilon})\mathsf{polylog}(k,n))$ to construct an algorithm that is $\mathsf{polylog}(k,n)$-competitive.

The main idea in proving Theorem 3 is that we apply the allocation problem recursively on the nodes of the tree. At time t for each node p we compute the cost restricted to the subtree under node p if we allow j servers in that subtree. In this construction it is very important to have a good bound on the hit cost since it multiplies over the levels in the recursion and our algorithm will end up paying $(1+\epsilon)^{\Delta}$.

The main problem with this approach is that we do not know how to find a good algorithm for the allocation problem. In their work Cote, Meyerson, and Poplawski [CMP08] gave an algorithm for a the case where we have only two nodes. Their algorithm for the allocation problem resulted in an algorithm for k-server problem on binary HSTs.

2.4 Our Results

The next result follows from joint work with Nikhil Bansal, Niv Buchbinder, and Aleksander Madry.

Theorem 4. *There is an* $\tilde{O}(\log^3 n \log^2 k)$*-competitive algorithm for the k-server problem on any metric with n points.*

We use a fractional version of the frame work in [CMP08] to prove Theorem 4.

2.5 Fractional Paging and Allocation Problems

Fractional Paging: Similar to original paging problem we have n pages and a cache of size k. We receive requests for pages over time, and if a page is not in the cache we have to bring it to the cache. The main difference between the fractional version and original paging problem is that in the fractional version we are allowed to store a fraction of a page in the cache, and the cost that we have to pay to bring a page to the cache proportional to the fraction that we have to move from outside of the cache to to the cache.

Let $p_1, \ldots p_n$ be the fraction of the pages that reside in the cache. If an update changes the state of the cache from $p_1, \ldots p_n$ to $q_1, \ldots q_n$, then the cost of that update is given by

$$\text{cost} = \frac{1}{2}\sum_{i=1}^{n} |p_i - q_i|.$$

We can interpret the fractional solution as the probability distribution of the states, such that the fraction p_i is the probability that page i is in the cache. Our cost function for the fractional solution

is the earth mover's distance between the two distributions. We can use this observation to update our distribution at each step of the algorithm and construct a randomized algorithm for the paging problem that only loses a factor of two in performance compared to the fractional solution.

Now, we can ask if a similar bound also holds for the allocation problem or not.

Fractional Allocation Problem: We define the fractional allocation problem for k servers on n nodes as follows. For $j \in \{0, \dots k\}$ and $i \in \{1, \dots, n\}$, let x_{ij} be the probability of having j servers at location i. At any time, for all $i \in \{1, \dots, n\}$, we have to satisfy the inequality

$$\sum_{j=0}^{k} x_{ij} = 1.$$

Moreover, since we have only k servers

$$\sum_{i=1}^{n} \sum_{j=0}^{k} j x_{ij} \leq k.$$

The hit cost for request vector $(h(0), \dots, h(k))$, is given by

$$\text{hit cost} = \sum_{j=0}^{k} x_{ij} h(j),$$

and the cost for moving mass ε from (i, j) to (i, j') is $\varepsilon |j - j'|$ (it corresponds to changing the number of servers at location i from j to j' in the original allocation problem).

Unlike the fractional paging problem, a fractional allocation is not a good approximation to the allocation problem.

A Gap Example: Consider the allocation problem with two nodes and k servers. Requests alternate between node one with cost vector $(1, 0, \dots, 0, 0)$, and node two with cost vector $(1, 1, \dots, 1, 0)$. Any integral solution must pay $\Omega(t)$ after t steps while the following fractional solution does not pay any move cost and only pays $O(\frac{t}{k-1})$ on the hit cost.

$x_{1,1} = 1$		hit cost $= 0$
$x_{2,0} = \frac{1}{k-1}$,	$x_{2,k} = 1 - \frac{1}{k-1}$	hit cost $= \frac{1}{k-1}$

Nonetheless, the fractional solution for the allocation problem is all we need to construct a fractional algorithm for k-server problem, and we prove an analog version of Theorem 3.

Theorem 5. *If there is an algorithm for the fractional allocation problem such that*

1. *hit cost* $\leq (1 + \epsilon) OPT$,
2. *move cost* $\leq \beta(\epsilon) OPT$,

then there exists an $O(\Delta\beta(\frac{1}{\Delta}))$-competitive fractional algorithm for k-server problem on HSTs, where Δ is depth of the tree.

We show that we can construct a randomized algorithm using the fractional algorithm while losing only $O(1)$ factor in performance, and finally we design an algorithm for fractional paging problem with $\beta(\epsilon) = \log(\frac{k}{\epsilon})$. In our work, we extended the algorithm and its analysis to the fractional allocation problem, however we do not present our extension in this lecture.

Fractional Paging Algorithm: Now, we describe our fractional paging algorithm. Our fractional paging algorithm is *implicitly* based on the primal-dual approach that was discussed earlier in Section 2.2. Suppose that we have n pages and k servers, and the current state of the cache contains a p_i fraction of page P_i. When we receive a request for a new page P_j, we need to bring $1 - p_j$ mass of page P_j into the cache. To this end we will evict mass from all other pages using multiplicative weights. The rule for page eviction (in each infinitesimal step) is that we evict page $P_{j'}$ at a rate proportional to $1 - p_{j'} + 1/k$.. The intuition for using this rule is that we want be more conservative evicting from pages having a large mass in the cache.

Overview of the analysis of the paging algorithm: Consider an optimal offline algorithm, and let $Off(t)$ be the content of the cache after t steps in this algorithm. We analyze the performance of our paging algorithm by first defining a potential function $\Phi(t)$ and then using it to offset the cost of our algorithm. Define:

$$\Phi(t) = \sum_{i \notin Off(t)} \log\left(\frac{1 + 1/k}{1 - p_i + 1/k}\right).$$

The idea behind the definition of $\Phi(t)$ is the following. For a page P_i, if our online algorithm and offline algorithm coincide, then the contribution of P_i to Φ is zero. If our online algorithm has page P_i in the cache, but the offline algorithm does not have it, then Φ is used to offset the cost, and when p_i is close to one, the contribution of P_i to Φ is large.

We bound the total cost of our algorithm by showing that

$$\Phi(t) - \Phi(t-1) + On(t) - On(t-1) \leq O(\log k)(Opt(t) - Opt(t-1)),$$

where $On(t)$ is the cost of our algorithm after t steps and $Opt(t)$ is the cost of the offline algorithm after t steps.

2.6 Concluding Remarks

The dependence on Δ in our paper was removed by extending the allocation problem to weighted stars. The main question that is still open is whether we can remove the dependence on n or not? We know that we have to lose a factor of $O(\log n)$ when we embed our metric into a distribution over HSTs, but can we even get a bound for HSTs which is independent of n? Right now our algorithm depends on the depth of the HST which is $O(\log n)$. What about special metrics such as a line metric?

References

[BBN07] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. In *FOCS*, pages 507–517, 2007.

[BBN10] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Metrical task systems and the -server problem on hsts. In *ICALP (1)*, pages 287–298, 2010.

[BN09] Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, 34(2):270–286, 2009.

[CL06] Béla Csaba and Sachin Lodha. A randomized on-line algorithm for the -server problem on a line. *Random Struct. Algorithms*, 29(1):82–104, 2006.

[CMP08] Aaron Cote, Adam Meyerson, and Laura J. Poplawski. Randomized k-server on hierarchical binary trees. In *STOC*, pages 227–234, 2008.

[FKL+91] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.

[FRT03] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC*, pages 448–455, 2003.

[KP94] Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. In *FOCS*, pages 394–400, 1994.

[MMS88] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for on-line problems. In *STOC*, pages 322–333, 1988.

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.