Lower Bounds for Online Algorithms

1 Introduction

For all of the problems studied so far the following were true:

1. The algorithm is given the entire input all at once.
2. The answer is a string (e.g., YES or NO or an assignment that satisfies the max number of clauses).

An ***online problem*** is one where the following are true:

1. The input is given in pieces (e.g., requests for memory access).
2. The answer is a sequence of answers given each time you see more information (e.g., assignments of whether to put a page in cache or memory).

We give a small example.

example 1. *There are bins of size 10. Numbers in* $\{1, \ldots, 10\}$ *will be coming in and you want to pack them into bins (sum in a bin is* ≤ 10*) to use as few bins as possible. But as soon as you see a number you must put it into an existing bin or create a new one. We give two algorithms.*

First Fit *Put a number in the least indexed bin that it fits, and if there is none then create a new bin.*

On input $3, 3, 3, 3, 3, 3, 4, 4, 4$ *the result is as follows.*

1. *Bin 1: 3,3,3.*
2. *Bin 2: 3,3,3.*
3. *Bin 3: 4,4.*
4. *Bin 4: 4.*

This is not optimal since that would be 3,3,4 then 3,3,4, then 3,3,4. But the algorithm cannot see ahead.

First Fit with a 1-Rule *Put a number in the least indexed bin that it fits, UNLESS when you do this you have a bin with exactly 9.*

On input $3, 3, 3, 3, 3, 3, 4, 4, 4$ *the result is optimal:*

1. *Bin 1: 3,3,4.*

2. *Bin 2: 3,3,4.*

3. *Bin 3: 3,3,4.*

There are other inputs where this algorithm is not optimal. For example, on input 3,3,3,2,8 the result is as follows.

1. *Bin 1: 3,3,2.*

2. *Bin 2: 3.*

3. *Bin 3: 8.*

This is not optimal since that would be $3, 3, 3$ *and* $2, 8$.

In analyzing online algorithms, we usually, do not really care about the running time. The source of hardness here is lack of information, not complexity assumptions like P $\neq$ NP. Since the algorithm does not know the rest of the input it may not be able to make the optimum decisions.

In online problems, we want to have a *good solution*. How to measure a *solution*? We compare the outcome of an online algorithm with the best possible offline solution. This is the notion of ***competitive ratio***, which is defined below:

Definition 1. *Suppose, someone knows the whole input and finds an optimum solution (*OPT*). We want to compare ourselves with this optimum. Let* OPT *denote the cost of an optimal offline solution. Let* ALG *denote the cost of our algorithm. α-competitive ratio is defined as follow:*

1. *(in minimization): for all σ,* $\mathrm{ALG}(\sigma) \leq \alpha \mathrm{OPT}(\sigma) + \gamma$. *Note that $\alpha \geq 1$ and we the smaller it is, the better the algorithm.*

2. *(in maximization): For all σ,* $\mathrm{ALG}(\sigma) \geq \alpha \mathrm{OPT}(\sigma)$. *Note that $\alpha \ll 1$ and the smaller it is, the better the algorithm.*

Where γ is some constant independent of σ.

For this chapter, "algorithm" means ***online algorithm***.

Exercise 1. *Prove that the First Fit algorithm for bin packing has competitive ratio* ≤ 2.

Johnson et al. [5] showed that the competitive ratio for first fit bin packing is $\frac{17}{10}$. They also look at other approaches.

2 Caching Problem

There are n pages of RAM and k pages of cache. When a page from RAM is requested it is put into the cache; however, some page from the cache has to be kicked out of cache and put into RAM. We want the pages in the cache to be ones that are going to be requested either soon or a lot.

We will assume that the first k requests have already been made and are in the cache.

Problem 2.1. CACHE

INSTANCE: : A sequence of requests for pages from RAM. The cache is size k and initially there are k pages in the cache.

QUESTION: : Every time a request is made we first check if the page is already in the cache. If so then the cost is 0 and we do not need to do anything (though we may keep track of the fact that the request was made). If the page is not in cache then we bring it into cache and put some page that was in cache into memory. The cost is 1. The goal is to minimize the cost, so minimize the number of times that a page is requested that is not in cache.

There are several algorithms that one may consider the caching problem such as:

1. **FIFO:** First In, First Out. Remove from cache the element that has been there the longest.
2. **LIFO:** Last In First Out. Remove from cache the element that was put there most recently.
3. **LRU:** Least Recently Used. Remove from cache the element that has been requested the longest ago.
4. **LFFO:** Least Frequency First Out. Remove from cache the element that has had the least requests.

Definition 2. *A **fault** is when a request is made that is not in the cache.*

Theorem 1. *LRU is k-competitive.*

Proof. Let $S = \sigma_1, \ldots, \sigma_N$ be a sequence of requests. We need to show that for every k faults that LRU makes, the optimal algorithm would make at least 1 fault.

We assume that the first k different page requests are put into the cache by both the optimal algorithm and LRU. We only consider the page requests after the first k.

We partition S (except for the first k distinct requests) as $\Sigma_1, \ldots, \Sigma_L$ where for all $i \geq 1$, in Σ_i LRU makes exactly k faults. We show that, for all $i \geq 1$, each Σ_i can be mapped to a fault of the optimal algorithm.

Let $1 \leq i \leq L$. Let σ be the last request made in Σ_{i-1}. There are three cases.

1. **Case 1:** The k page requests in Σ_i are distinct from each other and from σ. Hence σ together with the requests in Σ_i have $k+1$ distinct page requests. Hence the optimal algorithm will have a fault.

2. **Case 2:** There is some page τ that LRU faults on twice in Σ_i. Say $\sigma_i = \tau$ and $\sigma_j = \tau$. Then when the σ_i request is made τ is brought into cache; however, by the time the σ_j request is made, τ has been evicted. When it was evicted it was the Least Recently Requested page. Hence between σ_i and τ being evicted, there must have been $k+1$ distinct requests. Hence the optimal algorithm will have a fault.

3. **Case 3:** All of the faults in Σ_i are distinct from each other but one of them is σ. Hence σ must have been evicted. From here the reasoning is as in Case 2.

□

Now that we have an online k-competitive algorithm, the question arises, is there a better one? Sadly no.

Theorem 2. *There is no deterministic algorithm with competitive ratio better than k for the caching problem.*

Proof. Let ALG be a deterministic online algorithm for the cache problem. We use an adversary argument. That is, we feed the algorithm a sequence of page requests that will force it to have competitive ratio $\sim k$. We denote what we feed it $\sigma_1, \ldots, \sigma_N$. We will think of N as large, much larger than k. We will assume k divides N.

1. Initially request $k+1$ distinct pages. This will cause a fault. Let τ_1 be the evicted page.

2. Ask for τ_1. Let τ_2 be the evicted page.

3. Ask for τ_2. Let τ_3 be the evicted page (it could be that $\tau_1 = \tau_3$).

4. Ask for τ_3. Let τ_4 be the evicted page.

5. Do this N times.

If there are N requests then there will be N faults (after the first k which we do not count).

The optimal would have been to evict the page that will be requested furthest in the future. Realize that this means that after a fault there will not be another fault for at least k requests. Hence OPT causes $\leq \frac{N}{k}$ requests.

Since ALG causes $\sim N - k$ faults and OPT causes $\frac{N}{k}$ faults, the competitive ratio is bounded by:

$$\frac{N-k}{N/k} \sim k.$$

□

The above technique to prove lower bounds on deterministic algorithms is typical: for every deterministic algorithm construct a sequence of requests that cause many faults for that algorithm (which cannot see the future) but fairly few faults for the optimal algorithm (which can see the future).

But what about randomized algorithms? Fiat et al. [3] showed the following

Definition 3. *For all k, H_k is $\sum_{i=1}^{k} \frac{1}{i}$ which is $\ln k + \Theta(1)$.*

Theorem 3. *Let k be the size of the cache.*

1. *There is a randomized paging algorithm with competitive ratio $2H_k = 2\ln k + O(1)$.*
2. *If* ALG *is any randomized paging algorithm then the competitive ratio is $\geq H_k = \ln k - O(1)$.*

Proof.
1) We present the ***Marking Algorithm***

1. Initially there are k pages in cache. They are not marked.
2. Whenever a request is made, whether or not it is in cache, it is marked.
3. If a request is made for a page not in cache then a page is chosen uniformly at random from the unmarked pages to be evicted.
4. When all k pages in cache are marked, all marks except the most recent one are removed.

2) We show that any randomized paging algorithm has competitive ratio $\geq H_k$.

□

3 Yao's Lemma

Proving a lower bound on the expected runtime of an randomized algorithms seems hard! An adversary argument won't work since that just gives one input that the algorithm is bad at.

Yao's lemma [8] allows us to obtain lower bounds on the expected runtime of a randomized algorithm by looking at lower bounds on deterministic algorithms.

Assume you have some problem (e.g., CACHE). There is an associated cost that we are trying to minimize. Picture the following two scenarios:

1. Let $\mathcal{A}$ be a set of deterministic algorithms. Let q be a distribution over a set of inputs.

   Fix an $a \in \mathcal{A}$. Use q to pick the input x. We denote the expected cost $E[c(a,x)]$. We pick the a that minimizes this. Hence we have the quantity $\min_{a\in\mathcal{A}} E[c(a,x)]$.

2. Let $\mathcal{X}$ be a set of inputs. Let p be a distribution over a set of algorithms.

   Fix an $x \in \mathcal{X}$. Use p to pick the algorithm a. We denote the expected cost $E[c(a,x)]$. We pick the x that maximizes this. Hence we have the quantity $\max_{x\in X} E[c(a,x)]$.

The following is Yao's Lemma:

Lemma 1. $\max_{x\in X} E[c(a,x)] \geq \min_{a\in\mathcal{A}} E[c(a,x)]$.

To get the intuition we recap what each side of the equation is.

- $\max_{x\in X} E[c(a,x)]$. We are picking the input x to make the expected cost (picking the algorithm via distribution p) as high as possible.
- $\min_{a\in\mathcal{A}} E[c(a,x)]$. We are picking the algorithm a to make the expected cost (picking the input via distribution q) as low as possible.

4 Online Matching

Problem 4.1. ONLINE MATCHING

*INSTANCE: : A bipartite graph $((U,V),E)$ is going to be the final input. Initially we have the set V in advance, which are called **offline vertices**. The vertices in U arrive one by one, which are called **online vertices**. At the time we get u_i, we get all of its neighbors as well.*

QUESTION: : When we receive an online vertex u_i, we need to match u_i to one of the offline vertices that has not already been matched, if there is one. This decision is irrevocable. The goal is to maximize the number of matches.

The ***greedy algorithm*** for this problem will, given a new u_i, match it to the least indexed, still available, $v \in V$ such that $(u_i, v) \in E$. (This algorithm does not look particularly greedy. We discuss a truly greedy algorithm for the weighted case in the exercises.)

Exercise 2. *Let $G = ((U, V), E)$ be a bipartite graph. We will assume here and throughout this paper that $|U| = |V|$.*

1. *Show that the greedy algorithm for online matching always returns a maximal matching. Note that this is maximal, meaning that no edge can be added.*
2. *Show that for any bipartite graph a maximal matching is $\geq \frac{1}{2}|V|$.*
3. *For all n give an example of a graph where $|U| = |V| = n$ and an arrival order for U where (1) the graph has a matching of size n, but (2) the greedy algorithm produces a matching of size $\frac{n}{2}$.*
   Hint: *Use the bipartite graph where (a) for $1 \leq i \leq \frac{n}{2}$ there are edges (u_i, v_i) and $(u_i, v_{i+(n/2)})$, and (b) for $\frac{n}{2} + 1 \leq i \leq n$ there is an edge $(u_i, v_{i-(n/2)})$. See Figure 1.*
4. *Show that the greedy algorithm has competitive ratio $\frac{1}{2}$. (this follows from the Parts 2 and 3).*
5. *Show that* any *deterministic algorithm will have competitive ratio $\leq \frac{1}{2}$.*
   Hint: *Use an adversary argument.*

We now consider a promising randomized algorithm, just to show that it does not do well after all.

Theorem 4. *Consider the randomized algorithm which picks a match for u_i at random from the vertices that are available.*

1. *If you run this algorithm on the graph from Exercise 2.3 the expected competitive ratio is $\frac{3}{4}$.*
2. *Let $G = (V, U, E)$ be the following bipartite graph. $V = \{v_1, \ldots, v_n\}$. $U = \{u_1, \ldots, u_n\}$. For all i there is an edge (u_i, v_i). For all $1 \leq i \leq \frac{n}{2}$ for all $\frac{n}{2} + 1 \leq j \leq n$ there is an edge (u_i, v_j). If you run the algorithm on this graph then the expected competitive ratio is 2.*

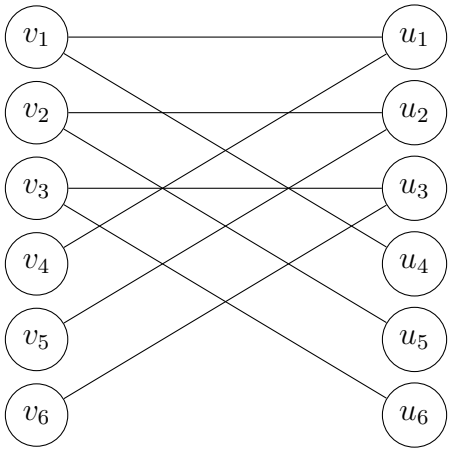

Figure 1: Greedy Online Matching has Competitive Ratio at Most $\frac{1}{2}$

Proof.
1) For $1 \leq i \leq \frac{n}{2}$ when u_i arrives the probability that it will match to $v_{i+(n/2)}$ is $\frac{1}{2}$. Hence the expected number of $v_{n/2}, \ldots, v_n$ that are available for $u_{n/2}, \ldots, u_n$ is $\frac{n}{4}$. So the expected number of matches is $\frac{n}{2} + \frac{n}{4} = \frac{3n}{4}$. The optimal is n. hence the competitive ratio is $\frac{3}{4}$.

2) The vertices $u_1, u_2, \ldots, u_n$ arrive in that order. For $1 \leq i \leq \frac{n}{2}$ if u_i gets matched to v_i thats good (in terms of maximizing matches) since it does not take take a vertex that is the only neighbor of some u_i with $i \geq \frac{n}{2} + 1$. Hence we call such a vertex ***good***.

$$\Pr[u_1 \text{ is good}) = \frac{1}{n/2+1}$$
$$Pr[u_2 \text{ is good}) = Pr[u_1 \text{ is good}) \frac{1}{n/2+1} + Pr[u_1 \text{ is bad}) \frac{1}{n/2} \leq \frac{1}{n/2}$$

We leave it as an exercise to show that

$$Pr[u_i \text{ is good}) = 1/(n/2 - i + 2)$$

Thus we have:

$$E(\text{ALG}) \leq \sum \frac{1}{n/2 - i + 2} + n/2 = O(\log n) + n/2.$$

Note that there is a matching of size n. Hence the competitive ratio is

$$\frac{O(\log n) + (n/2)}{n} \sim n.$$

□

To recap: any deterministic greedy algorithm, and the simple randomized algorithm, have competitive ratio 2. Is there a randomized algorithm with compete ratio < 2. Yes! Karp et al. [6] (see also Birnbaum & Mathieu [2], and Plotkin [7]) showed the following.

Theorem 5. *Let e be Euler's number, roughly 2.7185.*

1. *There is a randomized algorithm for online matching with competitive ratio $\frac{e-1}{e} \sim 0.632$.*
2. *All randomized algorithm for online matching have competitive ratio $\leq \frac{e-1}{e} \sim 0.632$.*

Proof sketch: We give the algorithm and prove that no randomized algorithm does better. We will not show that the competitive ratio is $\frac{e-1}{e}$.

Here is the algorithm, which we call RPERM.

1. Before the algorithm begins pick a permutation of V at random. This gives a ranking of V.
2. When $u \in U$ is seen, match it to the least element of V (according to the ranking) that is unmatched.

In preparation for applying Lemma 1 we present a distribution of inputs.

- For all i and $i \leq j$, we have the edge (u_i, v_j).
- For every permutation π, $I(\pi)$ relabels the vertices in V by π, i.e. we have edges (u_i, v_{π_i}).
- The input is $I(\pi)$, where π is chosen uniformly at random.

Claim 1: For any algorithm A we have, $E[A(P)] \leq E[\text{RPERM}(I)]$.

Proof of Claim 1

Consider an arbitrary iteration i. At (the beginning of) step i, let the set of eligible vertices be $Q(i) = \{v_{\pi_i} | j \geq i\}$.

a) Suppose that, A (or RPERM) have k unmatched eligible vertices. Any two subsets of size k from $Q(i)$ are the set of unmatched eligible vertices, with the same probability.

Let $P(i,k)$ be the probability that at iteration i, the number of unmatched eligible vertices is k. In fact, we have $Pr_{A(P)}(i,k) = Pr_{\text{RPERM}(I)}(i,k)$.

This shows that the expected number of steps that makes the number of unmatched eligible vertices 0 are the same in A and RPERM.

End of Proof of Claim 1

Claim 2: $E[\text{RPERM}(I)] \leq n(1-1/e)$

Proof of Claim 2

Consider the algorithm RPERM. For each iteration i, define the following two random variables:

- $x(i) = n - i + 1 = |Q(i)|$
- $y(i)$ =number of unmatched eligible vertices.

Consider that we have:

- $\Delta x = -1$.
- $\Delta y = -2$ if v_{π_i} is unmatched and u_i will not me matched to it.
- $\Delta y = -1$ otherwise.

By Claim 1 we have $Pr[v_{\pi_i} \text{ is unmatched}] = \frac{y(i)}{|Q(i)|}$. Therefore

$$Pr[y(i+1) - y(i) = -2] = \frac{y(i)}{x(i)} \frac{y(i)-1}{y(i)}$$

Thus, we have

$$E[\Delta y] = -1 - \frac{y(i)-1}{x(i)}$$

which gives us

$$\frac{E[\Delta y]}{E[\Delta x]} = 1 + \frac{y(i)-1}{x(i)}$$

When n goes to infinity, this can be approximated by the solution of the following differential equation:

$$\frac{dy}{dx} = 1 + \frac{y-1}{x}$$

which gives us

$$y(n+1) = \frac{n}{e} - o(n).$$

This completes the proof of the Claim.
End of Proof of Claim 2

From Lemma 1 and Claims 1,2 we have

∎

5 Online Set Cover

We first recall the usual offline Set Cover Problem.

Problem 5.1. SETCOVER

INSTANCE: : F the entire domain which we can think of as a set $\{1, \ldots, n\}$*, and E a set of subsets of F.*

QUESTION: : What is the size of the smallest collection of sets from E that contain (cover) all of the elements of F.

We will be interested in the online version of this problem.

Problem 5.2. OLS

INSTANCE: : F the entire domain which we can think of as a set $\{1, \ldots, n\}$*. It will be given ahead of time. E is a collection of subsets of F. It is also given ahead of time. Elements of F, together with a list of which sets in E cover them arrive one by one.*

QUESTION: : When we receive an elements $x \in F$ *and a set of subsets* $E_1, \ldots, E_k$ *that each cover x, pick one. Indeed, this decision is irrevocable. The goal is to minimize the number of sets in E that cover all of the elements received. (Note that we may well not receive all of the elements.)*

This online problem is very different from both Cache and Online Matching. For those two problems the offline version was in P. For SETCOVER the offline version is NP-hard.

To prove a lower bound on OLS we (surprisingly) need to assume P $\neq$ NP. Recall that Theorem **??**.4 states that if P $\neq$ NP then there is no $(1 - o(1)) \lg n$-approximation for (offline) set cover. (This means that there is no algorithm that will return a number of sets that is $\leq (1 - o(1)) \ln n \times$ OPT where OPT is the optimal (minimum) number of sets needed.) From this result it is easy to obtain the following:

Theorem 6. *If* P $\neq$ NP *then the competitive ratio for* OLS *is* $\geq (1 - o(1)) \ln n$.

We will prove a larger lower bound on the competitive ratio.

Theorem 7. *If* P $\neq$ NP *then the competitive ratio for* OLS *is* $\geq \Omega(\log^2 n)$.

Proof. Let (E, F) be an offline hard instance with an optimal solution of size k, but any algorithm that runs in polynomial time, can not compute a solution better than k' (we know $k' \in \Omega(\log(n))k$). We will construct an online instance (U', F', E') with an optimal solution of size k s.t. an online algorithm with solution of size better than $k'O(\log(n))$ requires solving (E, F) with a cost better than k'.

This implies an $\Omega(\log^2(n))$-hardness for online algorithms that run in polynomial time.

We will shortly see that for a size N, we have $|E| = m, F = n, |U'| = (N' - 1)m, |F'| = \frac{N}{2}n, |E'| = m.\log(N)$.

We depict these elements in the binary tree showed in Figure **??**. For every copy, at $i \in [1 \dots N - 1]$, we have a path from root to that copy.

Let the ordered vector of indices P_i, denote the indices of the path form root to i.

- $P_1 =< 1 >$
- $P_i =< P_{[i/2]}, i >$

Constructing the sets:

Substantively, every leaf $i \in [N/2, \dots, N - 1]$, has a copy of offline subsets F, that covers the same set of elements form $U_i, U_{i/2}, \dots, U_1$.

For every $f \in F$ and $i \in [1 \dots N - 1]$, let $U_i(f)$ denote the elements of U_i that correspond to the copies of f.

For every $i \in [N/2, \dots, N-1]$, let F_i denote a copy of F such that, for all $f \in F$, $f_i = \{j \in \{1, \dots, \log N\} \mid U_{p_i(j)}(f)\}$.

For every leaf i, we construct an online sequence E'_i as follow:

$$E'_i = \langle U_{p_i(1)}, U_{p_i(2)}, \dots, U_{p_i(\log(N))} \rangle.$$

By Yao's lemma, it is sufficient to show that if we pick a leaf i uniformly at random, then every online deterministic algorithm that runs in polynomial time uses $\Omega(\log(n)k')$ sets to cover the online requests E'_i.

Consider an arbitrary step $j \in [1, \dots, \log(N)]$ in which we receive $U_{p_i(j)}$. Consider the subtree T rooted at $U_{p_i(j)}$. Only the sets that are in the leaves of T can be useful.

We may assume that the online algorithm has to choose at least k' sets among these sets to cover $U_{p_i(j)}$. Let k_1 denote the number of selected sets in the left subtree of T, while k_2 denote the number of those at the right subtree. Consider that $k_1 + k_2 \geq k'$ which gives us $max(k_1, k_2) \geq \frac{k'}{2}$.

With out loss of generality assume that $k_2 \geq k_1$. With probability $\frac{1}{2}$ the next U-node might go to the left subtree. Hence, all the sets that contribute to k_2 will be redundant. Therefore the online solution wastes at least $\frac{k'}{2}$ sets with probability $\frac{1}{2}$. Thus, we have:

$$E[\text{size of an online solution}] \geq \frac{\log(N)}{2}\frac{k'}{2}$$

as desired.

Note that OPT is still k. we can simply choose the optimal solution at the final leaf.

□

6 Further Results

1. In Section 4 we looked at online matching for bipartite graphs where the *vertices* arrive. We found that (a) deterministic algorithms always have competitive ratio $\leq \frac{1}{2}$, (b) there is a randomized algorithms with competitive ratio $\frac{e-1}{e}$, and (c) $\frac{e-1}{e}$ is the best one can do.

   What about general graphs? What if edges arrive? Gamlash et al. [4] showed that (a) for vertex arrivals in general graphs there is a randomized algorithm with competitive ratio $\frac{1}{2} + \Omega(1)$, and (b) for edge arrivals randomization does not help.

2. Role-matchmaking is a problem where players of different skills levels arrive and must be assigned to a team as soon as they arrive. The goal is to have the teams be balanced so that no team dominates. This can get very complicated since different skills is not 1-dimensional. For example, in soccer a team may need a good Goalkeeper more than a great midfielder. This problem has immediate applications to many popular online video games where such as *League of Legends* and *Dota 2*. Alman & McKay [1] view this as a dynamic data structures problem. The show (a) assuming the 3SUM-Conjecture conjecture, any data structure for this problem requires $n^{1-o(1)}$ time per insertion or $n^{2-o(1)}$ time per query, and (2) there is an approximation algorithm that takes $O(\log n)$ per operation.

References

[1] J. Alman and D. McKay. Theoretical foundations of team matchmaking. In K. Larson, M. Winikoff, S. Das, and E. H. Durfee, editors, *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, pages 1073–1081. ACM, 2017. http://dl.acm.org/citation.cfm?id=3091277.

[2] B. E. Birnbaum and C. Mathieu. On-line bipartite matching made simple. *SIGACT News*, 39(1):80–87, 2008. https://doi.org/10.1145/1360443.1360462.

[3] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991. www.cs.cmu.edu/~sleator/papers/competitive-paging.pdf.

[4] B. Gamlath, M. Kapralov, A. Maggiori, O. Svensson, and D. Wajc. Online matching with general arrivals. In D. Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 26–37. IEEE Computer Society, 2019. https://doi.org/10.1109/FOCS.2019.00011.

[5] D. S. Johnson, A. J. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal of Computing*, 3(4):299–325, 1974. https://doi.org/10.1137/0203025.

[6] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In H. Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 352–358. ACM, 1990. https://doi.org/10.1145/100216.100262.

[7] S. Plotkin. Online algorithms, 2013. http://web.stanford.edu/class/cs369/files/cs369-notes.

[8] A. C. Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 222–227. IEEE Computer Society, 1977. https://doi.org/10.1109/SFCS.1977.24.