Online Algorithms: A Primer

anonymous shark

March 20, 2026

Abstract

The k-server problem asks how k mobile servers in a metric space should be dispatched to serve an online sequence of requests so as to minimize total distance traveled. Introduced by Manasse, McGeoch, and Sleator [MMS88], it has driven the development of competitive analysis for over three decades.

This expository work[1] traces that development from foundations to the present day. We begin with the paging and ski rental problems to motivate competitive ratios and adversarial models, then analyze deterministic and randomized paging algorithms, including the marking algorithm of Fiat et al. [FKL$^+$91] and the optimal partitioning algorithm of McGeoch and Sleator [MS91]. Turning to the k-server problem proper, we present the deterministic lower bound of k, the Double coverage [CKPV91] and Balance [MMS90] algorithms for special cases, and the Work Function Algorithm of Koutsoupias and Papadimitriou [KP95], which achieves a competitive ratio of $2k-1$ on all metrics. On the randomized side, we cover the $O(\log^2 k)$-competitive algorithm of Bubeck, Cohen, Lee, Lee, and Madry [BCL$^+$18] for hierarchically separated trees and the recent breakthrough of Bubeck, Coester, and Rabani [BCR23], who disproved the longstanding randomized k-server conjecture by constructing metric spaces requiring $\Omega(\log^2 k)$ competitive ratio. We conclude with the k-taxi problem [GKP24], the weighted k-server problem [GKP23, BMC26], and the learning-augmented framework [LV21], collecting open problems throughout.

Contents

[1] PSST! If you don't want to commit to reading all this, I'd recommend reading up to Section 4 and skipping the proofs in Section 4, as some of the bound analysis gets pretty dry. I'd recommend checking out 6.1, since it's kind of fun, and 7.1 as well.

1 Introduction

Uber is currently valued at around 150 billion USD. Abstracting the business away, their core algorithmic problem comes down to having some number of drivers in a city with a streaming set of pickup and drop-off locations, and wanting to minimize overhead. If we knew all requests in advance we could solve this offline, but in practice, requests arrive one at a time and decisions must be made immediately.

This is the regime of *online algorithms*, and the k-server problem is, in my opinion, its most celebrated representative. The story of the k-server problem is one of a thirty-year conjecture that turned out to be both half-right and half-wrong. We follow the arc from Sleator and Tarjan's foundational work [ST85] through the Work Function Algorithm [KP95] to the recent disproof of the randomized conjecture [BCR23], emphasizing the techniques that were derived along the way, such as potential functions, residues, work functions, HST embeddings, and entropic regularization.

Concretely, the paper is organized as follows.

1. **Get Online!**. We define online algorithms, adversarial models, and the competitive ratio, and introduce the paging and ski rental problems.
2. **Competitive Analysis**. We cover Sleator and Tarjan's analysis of paging [ST85], the marking algorithm [FKL$^+$91], and the optimal partitioning algorithm [MS91].
3. **The k-Server Problem**. We present the k-server conjecture [MMS88], the Double Coverage [CKPV91] and RES/BAL [MMS90] algorithms, and the Work Function Algorithm with its $(2k-1)$-competitiveness proof [KP95].
4. **Randomized k-Server**. We discuss the $\Omega(\sqrt{\log k / \log\log k})$ lower bound [BKRS92], HST embeddings [Bar96, FRT04], the polylogarithmic upper bound [BBMN15], and entropic regularization [BCL$^+$18, CL22].
5. **The Randomized Conjecture is False**. We describe the $\Omega(\log^2 k)$ lower bound of Bubeck, Coester, and Rabani [BCR23] and its recursive metric-space construction.
6. **Extensions and Open Problems**. We cover the k-taxi problem [GKP24], the weighted k-server problem [GKP23, BMC26], and learning-augmented algorithms [LV21, LMS25], and collect the major open questions.

This paper might feel rushed at the end, but for what it's worth? I doubt we'll see any fractal-like metric spaces in our lifetimes.

2 Get Online!

An online algorithm $\mathcal{A}$ is given the input a piece at a time and is forced to make a decision based on the input that we have fed it so far.

Definition 2.1. We say $\mathcal{A}$ is an online algorithm if it receives a sequence of requests $\sigma = (r_1, \ldots, r_n)$ and must choose some action after each request r_t is revealed, without knowing $r_{t+1}, r_{t+2}, \ldots$.

In this regime, we usually care less about the speed of the algorithm, and rather about how close we can approximate the offline, pure solution. To this end, we need to define some kind of way to score these algorithms. Note that the problem I described above (called the k-taxi problem) can be given a score by

summing up the total amount of distance driven. Similarly, we associate a cost function $C_{\mathcal{A}}$ for the "cost" of the solution outputted by the online algorithm.

To compare it against, we define C_{OPT} as the cost of the solution produced by the Optimal Offline Algorithm (OPT has better mouthfeel than OOA). Thus we can define the notion of a competitive ratio.

Definition 2.2 (Competitive Ratio)**.** We say that an online algorithm $\mathcal{A}$ is α-competitive if there exists a constant β such that for every input sequence σ,

$$C_{\mathcal{A}}(\sigma) \leq \alpha C_{\text{OPT}}(\sigma) + \beta.$$

Definition 2.3 (Randomized Competitive Ratio)**.** A randomized online algorithm $\mathcal{A}$ is α-competitive if there exists a constant β such that for every input sequence σ,

$$\mathbb{E}[C_{\mathcal{A}}(\sigma)] \leq \alpha C_{\text{OPT}}(\sigma) + \beta,$$

where the expectation is taken over the internal randomness of $\mathcal{A}$.

We say an algorithm is not competitive if no such α exists. Note that these consider "all" possible σ sequences. However, given an online algorithm, we can construct specific adversaries to analyze the algorithm's behavior.

The oblivious adversary is the simplest possible adversary. Here, we can fix the entire request sequence σ in advance. It doesn't adapt to the algorithm's choices. Slightly more involved is the adaptive adversary, which chooses future requests based on the algorithm's past behavior.

In some sense, these are all the tools you really need to think about these problems.

2.1 Simple Problems

Before we hit the big leagues with the k-taxi question, we're going to focus on slightly easier questions. The *paging problem* is inspired by handling memory. We have a cache size k, each of which can hold one page. We have m page requests. If a page in the cache is requested, we call it a hit, and if a page not in the cache is requested, we call it a fault and incur a cost of 1. We also have to move this page into the cache and boot out another page. You can intuit this design as splitting memory between a fast and a slow part, and the cost of accessing memory from the slow memory is 1 and 0 from the fast. So, now the question is how to best design this algorithm to minimize the number of faults.

Another conventional online algorithm is called the *ski rental problem*. You've likely interacted with this problem before. Broadly, a ski-rental type problem asks if given an expensive up front cost or a less expensive repeating cost, with no real knowledge of how the future will play out (will it snow in the Bay Area?) at what point should you pay the up front cost? More specifically, assume that renting a ski costs 1 unit a day and buying skis instead costs b units. Every day you are given the option to continue renting skis or to buy a pair of skis. The online adversary, on some unknown day D, is going to break your legs, so you'd like to minimize the cost of skiing so you can pay for your medical treatment.

Both of these problems are clearly extremely relevant to us as humans (caching and knee health) and therefore we must study them.

3 Competitive Analysis

This section will cover [ST85], one of the foundational papers in what would become competitive analysis. While the term "competitive analysis" was later coined by Karlin, Manasse, Rudolph, and Sleator [KMRS88] and the framework was further developed by Manasse, McGeoch, and Sleator [MMS88], Sleator and Tarjan's paper established the key idea of comparing an online algorithm's cost against the optimal offline algorithm. The paper describes the list update problem more generally and then goes into the paging problem.

For this section, k will refer to the cache size and m will refer to the number of requests made. First, we describe the optimal offline algorithm, which is simple and greedy. In the case of a fault, eject a page that we know will never be used again, or if you can't, eject the page whose future use is the furthest away. This is Bélády's algorithm, also known as Longest Forward Distance.

Theorem 3.1 (Bélády's Algorithm)**.** *Longest Forward Distance is optimal for the paging problem.*

Proof. For contradiction, assume that longest forward distance (LFD) is not optimal. Then there exists a finite input sequence σ where LFD does not return an optimal solution. Let OPT be an optimal algorithm, and suppose LFD and OPT first differ at step i.

We will modify OPT so that its cost does not increase and it agrees with LFD on steps $1, \ldots, i$.

First suppose the ith request does not cause a page fault under LFD. Then LFD makes no eviction, while OPT must evict some page to differ. Since both caches are identical before step i, OPT does not need to evict any page. Thus we can modify OPT so that it also performs no eviction at step i. If OPT later needs the page it unnecessarily evicted, it can evict it at that later time instead, without increasing the total cost. Hence OPT can be modified to match LFD at step i without increasing cost.

Now suppose the ith request is a page fault. Then LFD and OPT evict different pages; let p be the page evicted by LFD and p' the page evicted by OPT. By definition of LFD, the next request to p occurs strictly later than the next request to p' (or p is never requested again).

Let l be the first future time at which either p or p' is requested again. Since p is requested later than p', the request at time l must be for p'.

We construct a modified algorithm OPT$'$ as follows:

- At step i, OPT$'$ evicts p instead of p'.
- From steps $i+1$ through $l-1$, OPT$'$ behaves exactly like OPT.
- At step l, OPT$'$ incurs a page fault for p' and loads it into the cache, evicting p if necessary.

Up to time $l-1$, OPT$'$ has all pages that OPT needs, so it incurs no additional faults. At time l, OPT$'$ incurs one fault for p'. However, OPT must eventually incur a fault for whichever page it does not have among $\{p, p'\}$ when that page is requested. Thus OPT$'$ incurs no more faults than OPT overall.

Therefore OPT$'$ has cost no greater than OPT and agrees with LFD for the first i steps, contradicting the choice of i as the first point of disagreement. Hence LFD is optimal. ■

Now, another name for this algorithm is called the clairvoyant algorithm. So unless we have access to a psychic, this doesn't work in the online case. However, as one might've guessed already, an adaptive adversary might screw us over.

Theorem 3.2. *Any deterministic online algorithm cannot be less than k-competitive.*

Proof. Let $\mathcal{A}$ be any deterministic online algorithm for the paging problem with cache size k. We consider a universe of $k+1$ distinct pages. Before each request, $\mathcal{A}$ has exactly k pages in its cache, so there is always one page not currently in the cache. An adversary can request precisely that page. Thus $\mathcal{A}$ incurs a page fault on every request. For a sequence of length m, the total cost of $\mathcal{A}$ is therefore m.

Now consider the optimal offline algorithm OPT. Since there are only $k+1$ pages total, at any time OPT also has k pages in its cache and is missing exactly one page. By the adversary's construction, $\mathcal{A}$ always faults on every request. OPT, however, holds k of the $k+1$ pages at any time, and after serving a request it can keep that page in cache until a request arrives for the unique page it does not currently store.

We may assume OPT uses Bélády's algorithm. When OPT faults and evicts some page w, it chooses w to be the page whose next request is farthest in the future (or which is never requested again). After the eviction, OPT's cache contains k pages, each of which has its next request strictly before the next request to w. Therefore, all k cached pages must be requested before w is requested again, giving at least k requests between consecutive OPT faults.

It follows that over a sequence of length m,

$$C_{\mathrm{OPT}} \leq \left\lceil \frac{m}{k} \right\rceil.$$

Since $C_{\mathcal{A}} = m$, we obtain

$$\frac{C_{\mathcal{A}}}{C_{\mathrm{OPT}}} \geq \frac{m}{\lceil m/k \rceil} \geq k - o(1).$$

Thus no deterministic online algorithm can have competitive ratio strictly smaller than k. ■

This is a lower bound style result for online algorithms, at least for the worst-case competitive analysis. Inside of Sleator and Tarjan [ST85], they list out a few deterministic demand paging (only replace when necessary) algorithms from previous work:

1. **Least Recently Used (LRU).** When a replacement is necessary, evict the page whose most recent access occurred farthest in the past.
2. **First-In, First-Out (FIFO).** Evict the page that has been in fast memory the longest.
3. **Last-In, First-Out (LIFO).** Evict the page that was most recently brought into fast memory.
4. **Least Frequently Used (LFU).** Evict the page that has been accessed the fewest times.

We will only really discuss the competitiveness of the LRU and FIFO.

Theorem 3.3 (LRU)**.** *LRU is k-competitive.*

Proof. Partition the request sequence into *phases* as follows. The first phase begins with the first request and ends just before the $(k+1)$-st distinct page is requested. Each subsequent phase begins with the first request not included in the previous phase and again ends just before the next request that would introduce a $(k+1)$-st distinct page within that phase. Thus, each phase contains requests to at most k distinct pages.

Consider LRU during a phase. At the beginning of a phase, the cache may contain any pages. However, during the phase there are at most k distinct pages requested, so after the first request to each of those pages, LRU will contain all of them in its cache. Therefore, LRU incurs at most k faults per phase.

Now consider the optimal offline algorithm (OPT). Between two consecutive phases, at least $k+1$ distinct pages are requested (otherwise the phases would merge). Since the cache holds only k pages, OPT must incur at least one fault in each phase after the first.

If there are m phases, LRU incurs at most km faults, while OPT incurs at least $m-1$ faults. Hence

$$\text{LRU} \leq k \cdot \text{OPT} + k,$$

which implies LRU is k-competitive. ■

This concept of a phase is the main method of chunking and analyzing these paging style algorithms.

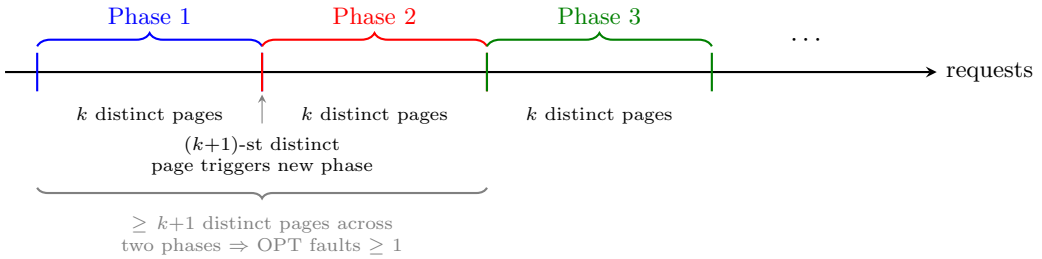

Figure 1. The phase structure used in the LRU and FIFO proofs. Each phase contains at most k distinct pages. Between consecutive phases, a $(k+1)$-st distinct page must appear, forcing OPT to fault at least once.

Theorem 3.4 (FIFO)**.** *FIFO is k-competitive.*

Proof. We use the same phase partition as in the LRU proof: each phase contains at most k distinct pages.

We claim FIFO incurs at most k faults per phase. Suppose FIFO faults more than k times in a phase. Each fault inserts a new page into the cache and evicts the oldest resident page. After k such faults, all pages currently in the cache must have been requested during the phase. A further fault would require requesting a page not currently in cache, which would imply more than k distinct pages were requested in the phase, contradicting the definition of a phase. Thus FIFO has at most k faults per phase.

As before, OPT must incur at least one fault per phase after the first, since each phase introduces at least one page not present among the previous k distinct pages.

If there are m phases, FIFO incurs at most km faults and OPT incurs at least $m - 1$ faults, so

$$\text{FIFO} \leq k \cdot \text{OPT} + k.$$

Therefore FIFO is k-competitive. ■

3.1 Marking Algorithm

In 1991, Fiat, Karp, Luby, McGeoch, Sleator, and Young [FKL+91] devised a randomized online algorithm for this problem called the marking algorithm. They were able to achieve a randomized competitive ratio of $2H_k \leq 2\ln(k) + 2$.

The algorithm is as follows. Our cache currently contains the pages 1 to k, and we "mark" all of these pages. After each request, we mark the page that is being requested. Once $k + 1$ pages are marked, all the marks except the most recently requested page are erased. If the requested page is already inside of the cache, we do nothing (other than marking it). Otherwise, a page is chosen uniformly at random from among the unmarked pages currently in the cache and is evicted; the requested page is then brought into the cache and marked.

Algorithm 1 Marking Algorithm for Paging

```
Initialize cache with pages 1 to k
Mark all pages currently in cache
for each request p in the request sequence do
    Mark page p
    if more than k pages are marked then
        Erase all marks except the mark on p
    end if
    if p is already in cache then
        continue
    else
        Choose a page uniformly at random from the unmarked pages in cache
        Evict the chosen page and bring p into cache
    end if
end for
```

Intuitively, marks ensure that during any period in which at most k distinct pages are requested, each page is evicted at most once in expectation. The paper describes this as a randomized-variant of LRU.

Theorem 3.5 (Marking Algorithm)**.** *The marking algorithm is $2H_k$-competitive.*

Proof. We use the same notion of phases, though now we partition the request sequence into phases defined by the algorithm itself: a new phase begins each time the marks are cleared. Within each complete phase, exactly k distinct pages are requested before the marks are cleared (the final phase may have fewer).

Fix a phase $p \geq 2$. At the start of the phase, the cache holds k pages, all unmarked. Let Q denote the set of (at most k) distinct pages requested in phase p. Partition the pages into three categories:

- **Old pages:** pages in $Q \cap S$, where S is the cache at the start of the phase. These are pages already in the cache that will be requested during this phase.
- **New pages:** pages in $Q \setminus S$. These are not in cache and will each cause exactly one fault when first requested.
- **Stale pages:** pages in $S \setminus Q$. These are in cache but will not be requested during this phase.

Let $d = |Q \setminus S|$ denote the number of new pages. Since $|S| = k$ and $|Q| \leq k$, there are also exactly d stale pages and $k - d$ old pages.

Every new page causes exactly one fault. Additionally, an old page causes a fault if and only if it is evicted (by a random eviction) before it is first requested. Once an old page is requested, it is marked and protected from eviction for the remainder of the phase. So the total number of faults in this phase is

$$\text{faults in phase } p = d + (\text{number of old pages prematurely evicted}).$$

Order the old pages as $o_1, o_2, \ldots, o_{k-d}$ by the time of their first request in this phase (o_1 is requested first, etc.). We claim that for each i,

$$\Pr[o_i \text{ is evicted before its first request}] \leq \frac{d}{d + (k-d) - (i-1)} = \frac{d}{k - i + 1}.$$

To see this, consider the moment just before o_i is first requested. At this point, $o_1, \ldots, o_{i-1}$ have already been requested and marked, so they are safe. The unmarked pages in the cache are o_i itself, the old pages $o_{i+1}, \ldots, o_{k-d}$ (not yet requested), and whatever stale pages have not yet been evicted. All evictions prior to o_i's request chose uniformly among unmarked pages. By a symmetry argument (each unmarked page is equally likely to be chosen at each eviction step), the probability that o_i was selected before any of the remaining d stale pages plus the other $(k - d - i)$ unrequested old pages is at most $\frac{d}{k-i+1}$.

By linearity of expectation:

$$\begin{aligned}
\mathbb{E}[\text{faults in phase } p] &= d + \sum_{i=1}^{k-d} \Pr[o_i \text{ evicted prematurely}] \\
&\leq d + \sum_{i=1}^{k-d} \frac{d}{k-i+1} \\
&= d + d \sum_{j=d+1}^{k} \frac{1}{j} \quad (\text{substituting } j = k - i + 1) \\
&= d \left(1 + \sum_{j=d+1}^{k} \frac{1}{j} \right) = d(1 + H_k - H_d) \leq d \cdot H_k,
\end{aligned}$$

Consider two consecutive phases p and $p+1$. Phase p requests a set Q_p of k distinct pages, and the $(k+1)$-st distinct page triggers the start of phase $p+1$ with its set Q_{p+1}. Thus $|Q_p \cup Q_{p+1}| \geq k + 1$. Since OPT has a cache of size k, it must incur at least $|Q_p \cup Q_{p+1}| - k \geq 1$ faults across phases p and $p+1$.

More precisely, let $d_{p+1} = |Q_{p+1} \setminus Q_p|$ be the number of new pages in phase $p+1$ relative to the previous phase's requests. Then OPT must fault at least d_{p+1} times across phases p and $p+1$, since the d_{p+1} pages in $Q_{p+1} \setminus Q_p$ were not in Q_p, and OPT's cache can hold at most k pages from Q_p.

From our bound, the marking algorithm's expected faults in phase $p+1$ is at most $d_{p+1} \cdot H_k$. Since OPT incurs at least d_{p+1} faults across the pair of phases p and $p+1$, we incur at most H_k times OPT's cost per pair. Summing over all phases and accounting for the double-counting (each phase participates in two consecutive pairs), we obtain:

$$\mathbb{E}[C_{\text{Marking}}] \leq 2H_k \cdot C_{\text{OPT}}.$$

■

Perhaps surprisingly, the marking algorithm's factor of $2H_k$ is not tight, rather, the lower bound for randomized paging is H_k, proved in the same paper [FKL+91]. McGeoch and Sleator [MS91] closed this gap with the *partitioning algorithm*, which achieves the optimal competitive ratio of exactly H_k. The key idea is to maintain a dynamically evolving labeled partition of the page universe that tracks the behavior of OPT, and then use this partition to guide randomized eviction decisions more carefully than the marking algorithm does.

The algorithm maintains an ordered sequence of disjoint sets $S_\alpha, S_{\alpha+1}, \ldots, S_\beta$ whose union is the full set of n pages. Each set S_i (except S_β) carries an integer label k_i satisfying $k_\alpha = 0$ and $k_i = k_{i-1} + |S_i| - 1$ for $\alpha < i \leq \beta - 1$, so that $k_{\beta-1} = k - |S_\beta|$. The set S_α consists of pages not in cache (analogous to "new" pages in the marking algorithm), while S_β consists of the most recently touched pages. These labels track how many pages OPT can afford to keep from each group.

The partition is augmented with a system of *marks*: for each i with $\alpha \leq i < \beta$, there are k_i distinct i-marks placed on *i-eligible* pages (those in S_i or having an $(i-1)$-mark). The algorithm keeps in cache every page in S_β and every page with a $(\beta - 1)$-mark.

Algorithm 2 Partitioning Algorithm for Paging (McGeoch–Sleator [MS91])

```
Initialize: α ← 1, β ← 2, S_1 ← pages not in cache, S_2 ← cached pages, no marks
for each request to page v do
    Let i be such that v ∈ S_i
    if v ∈ S_β then                                              ▷ Cache hit
        Do nothing
    else if α < i < β then                                       ▷ In an intermediate set
        for j = i to β − 1 do                                    ▷ Load v with marks
            if v does not have a j-mark then
                Pick a page w with a j-mark uniformly at random
                Transfer all ℓ-marks (ℓ ≥ j) from w to v
                If a (β − 1)-mark moves, load v into cache and evict w
            end if
        end for
        Move v to S_β; erase all marks on v; recompute labels k_i to restore the invariant
    else if v ∈ S_α then                                         ▷ Cache miss on uncached page
        Increment β; move v into the new S_β
        Create k − 1 new (β − 1)-marks; distribute uniformly among the k currently cached pages
        The page not receiving a mark is evicted from cache; load v
    end if
end for
```

The crucial difference from the marking algorithm is the refined partition structure. Where the marking algorithm uses a single binary distinction (marked/unmarked) and double-counts OPT's cost across consecutive phases, the partitioning algorithm maintains a multi-level partition that tracks OPT's state precisely via Lemma 1 of [MS91]. This allows the potential function $\Phi = \sum_{\alpha \leq i < \beta} (H_{k_i+1} - 1)$ to absorb the algorithm's cost exactly, eliminating the factor of 2.

Theorem 3.6 (Partitioning Algorithm [MS91])**.** *The partitioning algorithm is H_k-competitive for the paging problem.*

This first problem hopefully makes it clear that there is a division between the approximations between the optimal offline algorithm, an online algorithm with access to randomization, and a deterministic online algorithm.

4 Server Questions

The paging problem is actually a special case of a much more general framework. Manasse, McGeoch, and Sleator [MMS88] in 1988 formalized the k-server problem, which captures a broad class of online problems on metric spaces, and described certain conjectures that would direct the field over the next 30-so years. We still aren't advanced enough yet to hit the uber/taxi problem, but we're close.

First, we need to define a metric space.

Definition 4.1 (Metric Space)**.** A metric space is a pair (M, d) where M is a set and $d : M \times M \to \mathbb{R}$ that satisfies distance axioms. Specifically, $d(x, x) = 0$, $d(x, y) = d(y, x) > 0$ for $x \neq y$, and $d(x, z) \leq d(x, y) + d(y, z)$.

Now, the k-server problem is as follows. We are given some metric space M, d and a set of k "servers," where each server is represented as a labeled point in the metric space. Each request that comes in is in the form of a point in the space. As each request arrives, the algorithm must move a server to the requested point, and the total cost is the sum of all the distance covered in M, which is given to us by d. Intuitively, this captures problems like sending server repairers to servers that need it in some city, and again you want to minimize travel time and costs.

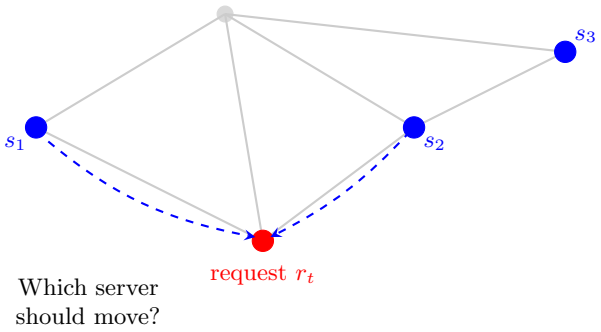

Figure 2. The k-server problem on a metric space with $k = 3$. When request r_t arrives, the algorithm must choose which server to move. The cost is the distance traveled in the metric.

Now, the first algorithm to pop into any unsuspecting student's mind (it did for me) is the greedy algorithm. For our sake, let's use the standard metric on $\mathbb{R}$, and let's use 2 servers, which we place at $0, 1$ for now. The greedy algorithm would simply be to move the closest server to the request location.

Intuitively, one would hope that this algorithm minimizes globally. However, it does not. Consider the stream of inputs alternating $(0.75, 1.25)$. Note that the server originally at 1 simply moves between them and we gain infinite cost, even though moving the 0 server to 0.75 and the 1 server to 1.25 would give us a finite cost. Therefore, the greedy algorithm doesn't even have a finite competitive ratio!

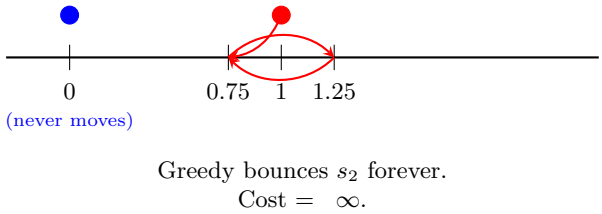

Figure 3. The greedy algorithm's failure on two servers at $0, 1$ with alternating requests at 0.75 and 1.25. Since s_2 is always the nearest server, it bounces back and forth indefinitely while s_1 never moves. The optimal solution moves each server once, for a finite total cost.

Perhaps surprisingly, a very soft modification of the greedy algorithm admits a provably competitive solution. Chrobak, Karloff, Payne, and Vishwanathan [CKPV91] introduced the Double Coverage (DC) algorithm, which is k-competitive for k servers on a line. We describe it here for the 2-server case. Let x_i, y_i denote the positions of the two servers before the ith request r_i. The Double Coverage (DC) algorithm works as follows:

- If $r_i \leq x_i$, move x_i directly to r_i.
- If $r_i \geq y_i$, move y_i directly to r_i.
- If r_i lies strictly between x_i and y_i, move both servers toward r_i at the same speed until one of them reaches the request.

Intuitively, DC spreads its servers evenly over the line and only moves both when the request is "between" them, ensuring that no server travels too far unnecessarily.

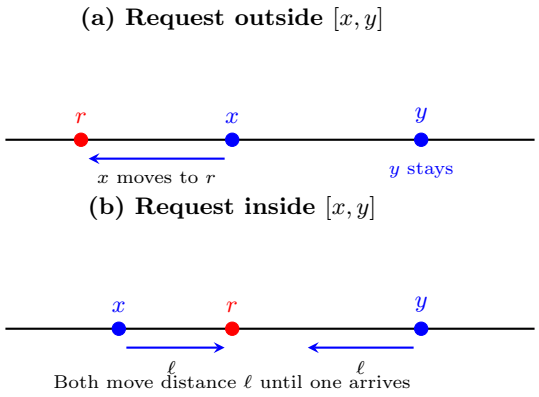

Figure 4. The two cases of Double Coverage. (a) When the request r lies outside the interval $[x, y]$, only the nearer server moves. (b) When r lies between the servers, both move toward r at equal speed until one reaches it.

Theorem 4.2 (DC)**.** *Double Coverage is 2-competitive for two servers on a line.*

Proof. We use a potential function argument. Let x and y denote the positions of the two DC servers with $x \le y$, and let p_1, p_2 denote the positions of the two OPT servers. After the t-th request has been served by both DC and OPT, define the following quantities:

- The *minimum-weight matching* between DC's servers and OPT's servers:
$$M = \min\big(|x - p_1| + |y - p_2|, |x - p_2| + |y - p_1|\big).$$
- The *spread* of DC's servers: $S = y - x \ge 0$.
- The *potential*: $\Phi = 2M + S$.

Note that $\Phi \ge 0$ always, since both $M \ge 0$ and $S \ge 0$.

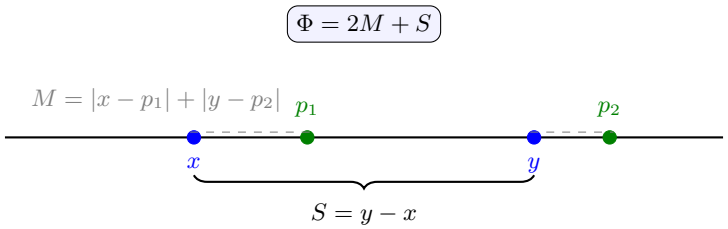

Figure 5. The potential function for the DC proof. The matching M measures how far DC's servers are from OPT's, and the spread S measures how far apart DC's servers are. The potential $\Phi = 2M + S$ balances both.

The proof is as follows. We will show that for each request r_t,

$$C_{\mathrm{DC}}(t) + \Delta\Phi(t) \le 2C_{\mathrm{OPT}}(t), \tag{4.1}$$

where $\Delta\Phi(t) = \Phi^{(\mathrm{after})} - \Phi^{(\mathrm{before})}$ is the net change in potential during the t-th request.

Summing (4.1) over all T requests and telescoping:

$$\sum_{t=1}^{T} C_{\mathrm{DC}}(t) + \Phi_T - \Phi_0 \le 2\sum_{t=1}^{T} C_{\mathrm{OPT}}(t).$$

Since $\Phi_T \geq 0$, we may drop it from the left side, yielding

$$C_{\mathrm{DC}} \leq 2C_{\mathrm{OPT}} + \Phi_0,$$

where Φ_0 is the initial potential (a constant depending on the starting configurations of DC and OPT). Thus DC is 2-competitive.

Now we must prove (4.1). Fix a request r. We decompose the potential change into OPT's response and DC's response:

$$\Delta\Phi = \Delta\Phi_{\mathrm{OPT}} + \Delta\Phi_{\mathrm{DC}}.$$

We analyze each in turn to compute delta.

Suppose OPT moves one of its servers a distance d_{OPT} to serve r. The spread $S = y - x$ does not change during OPT's move (only OPT's servers move). For the matching cost M, consider the two possible matchings. The cost of whichever matching was optimal before OPT's move changes by at most d_{OPT} (by the triangle inequality). The new minimum matching can only be cheaper, so M increases by at most d_{OPT}. Therefore,

$$\Delta\Phi_{\mathrm{OPT}} = 2\Delta M_{\mathrm{OPT}} + \underbrace{\Delta S_{\mathrm{OPT}}}_{=0} \leq 2d_{\mathrm{OPT}} = 2C_{\mathrm{OPT}}(t).$$

Now, we analyze DC's move. We consider two cases based on the position of the request r relative to the interval $[x, y]$.

In the first case, say $r \notin [x, y]$. Without loss of generality, suppose $r < x$. Then DC moves only the left server x to the request point r, traveling a distance $d_{\mathrm{DC}} = x - r > 0$. The spread increases: $\Delta S = d_{\mathrm{DC}}$.

After OPT has moved, one of OPT's servers is at r. Consider the matching that pairs DC's left server (now at r) with OPT's server at r at cost 0, and pairs DC's right server y with OPT's remaining server. Since y has not moved, the cost of the second pair is unchanged from before. Before DC's move, the matching that paired DC's left server x with OPT's server at r had cost $|x - r| = d_{\mathrm{DC}}$ for that pair; this pair now costs 0. Since the minimum matching M is at most the cost of any particular matching, M decreases by at least d_{DC}: $\Delta M \leq -d_{\mathrm{DC}}$. Therefore,

$$\Delta\Phi_{\mathrm{DC}} = 2\Delta M_{\mathrm{DC}} + \Delta S_{\mathrm{DC}} \leq 2(-d_{\mathrm{DC}}) + d_{\mathrm{DC}} = -d_{\mathrm{DC}}.$$

Now say that $r \in [x, y]$. Let $\ell = r - x$ and $\ell' = y - r$, so $\ell + \ell' = S$ and both are non-negative. Assume without loss of generality that $\ell \leq \ell'$. By the Double Coverage rule, both servers move toward r at equal speed until one arrives. The left server reaches r first (after moving ℓ), and the right server moves ℓ toward r as well. The total cost to DC is $C_{\mathrm{DC}}(t) = 2\ell$. The new spread is $S' = (y - \ell) - r = \ell' - \ell$, so $\Delta S = -2\ell$.

After OPT has moved, one of OPT's servers is at r. DC's left server is now also at r, so these can be matched at cost 0. DC's right server has moved from y to $y - \ell$. Consider the minimum-weight matching after DC's move. One candidate matching pairs DC's left server (at r) with OPT's server at r at cost 0, and pairs DC's right server $(y - \ell)$ with OPT's remaining server. Before DC's move, the matching that used the same pairing had cost $|r - x| + |y - q|$ for some OPT server q, and now the first pair costs 0 and the second pair changes by at most ℓ (since the right server moved distance ℓ). Since the minimum matching M is at most the cost of any particular matching, and we have exhibited one whose cost did not increase, we conclude $\Delta M_{\mathrm{DC}} \leq 0$. Therefore,

$$\Delta\Phi_{\mathrm{DC}} = 2\Delta M_{\mathrm{DC}} + \Delta S_{\mathrm{DC}} \leq 2(0) + (-2\ell) = -2\ell = -C_{\mathrm{DC}}(t).$$

In both cases, $\Delta\Phi_{\mathrm{DC}} \leq -C_{\mathrm{DC}}(t)$. Adding the contributions:

$$C_{\mathrm{DC}}(t) + \Delta\Phi(t) = C_{\mathrm{DC}}(t) + \Delta\Phi_{\mathrm{OPT}} + \Delta\Phi_{\mathrm{DC}} \leq C_{\mathrm{DC}}(t) + 2C_{\mathrm{OPT}}(t) - C_{\mathrm{DC}}(t) = 2C_{\mathrm{OPT}}(t).$$

This establishes (4.1), completing the proof. ■

In this proof, we introduced the notion of a *potential function*. Here, the potential measured two things: how "out of place" DC's servers were relative to OPT (via the minimum-weight matching M) and how spread out DC's servers were (via the spread S). The key mechanism is amortized analysis. Even though DC may pay a cost on a given request, this cost is always offset by a corresponding drop in potential. The factor of 2 in $\Phi = 2M + S$ is chosen precisely so that the bookkeeping works out, as when DC moves outside the interval, the spread increase is exactly canceled by twice the matching decrease, and when DC moves inside the interval, the spread decrease alone pays for everything.

4.1 Lower Bound for k-server

Note that the k-server problem is defined for a general metric space and most numbers are larger than 2. In fact, the following theorem can be shown.

Theorem 4.3 (Lower Bound on Servers)**.** *For any (symmetric) k-server problem, there is no α-competitive deterministic algorithm for $\alpha < k$.*

Proof. It suffices to exhibit a metric space on which no deterministic algorithm can achieve competitive ratio smaller than k. Consider the *uniform metric* on $k+1$ points $\{p_1, \ldots, p_{k+1}\}$, where the distance between every pair of distinct points is 1.

Just like with the paging problem, we're going to have an adaptive adversary. Let $\mathcal{A}$ be any deterministic online algorithm with k servers. At any time, $\mathcal{A}$ occupies k of the $k+1$ points, leaving exactly one point uncovered. The adversary generates a request sequence σ of length m by always requesting the unique point not currently occupied by $\mathcal{A}$. Since $\mathcal{A}$ must move a server (distance 1) to serve every request,

$$C_{\mathcal{A}}(\sigma) = m.$$

Note that consecutive requests are always to distinct points, since $\mathcal{A}$ covers the previous request before the adversary chooses the next.

We now show that the optimal offline algorithm satisfies $C_{\mathrm{OPT}}(\sigma) \leq \lceil m/k \rceil$. On the uniform metric, each server movement costs exactly 1, so C_{OPT} equals the number of requests at which OPT must move a server (i.e., the number of OPT "faults").

For now we assume without proof that OPT uses Bélády's algorithm: when OPT must move a server, it vacates the point whose next request in σ is the farthest in the future (or which is never requested again). The proof of Bélády's algorithm being optimal on the uniform metric is very similar to the one we have already done for the paging problem. Now, we claim that between any two consecutive OPT faults, at least k requests occur.

To see this, suppose OPT faults at time t, serving request r_t by moving its server off some point w (the point with the farthest next request, by Bélády's rule). After this move, OPT covers the k points $S = \{p_1, \ldots, p_{k+1}\} \setminus \{w\}$. The next OPT fault occurs when w is eventually requested. Until then, all requests are to points in S.

By the Bélády eviction rule, every point in S has its next request strictly before the next request to w. Therefore, each of the k points in S must be requested at least once before w is requested again. This gives at least k requests between the fault at time t and the next fault.

It follows that over m requests, OPT faults at most $\lceil m/k \rceil$ times (including the possible initial fault). Hence,

$$C_{\mathrm{OPT}}(\sigma) \leq \left\lceil \frac{m}{k} \right\rceil.$$

Thus, the competitive ratio of $\mathcal{A}$ on the sequence σ satisfies

$$\frac{C_{\mathcal{A}}(\sigma)}{C_{\mathrm{OPT}}(\sigma)} \geq \frac{m}{\lceil m/k \rceil} \geq \frac{m}{m/k+1} = \frac{k}{1+k/m} \to k \quad \text{as } m \to \infty.$$

More precisely, for any constant β, the inequality $C_{\mathcal{A}}(\sigma) \leq \alpha C_{\mathrm{OPT}}(\sigma) + \beta$ requires $m \leq \alpha \lceil m/k \rceil + \beta$, which fails for large m whenever $\alpha < k$. Since $\mathcal{A}$ was arbitrary, no deterministic online algorithm achieves competitive ratio strictly less than k. ∎

Generally speaking, dealing with metrics on $\mathbb{R}^n$ is somewhat annoying, and so instead we often think of dealing with metric spaces with finitely many points (oftentimes just $k+1$) and set distances between any two points such that it respects a metric. These can be represented as graphs and embedded in $\mathbb{R}^n$ so it is hopefully clear that working in the reduced metric space setting doesn't really abstract that much away.

Now, inside of [MMS90], they describe a general 2-competitive algorithm for the 2-server problem on an arbitrary n-vertex graph. Before we get there, we need a tool that will be useful for proving competitiveness.

4.1.1 Residues

Recall that in the lower bound and DC proofs, we compared the online algorithm's cost against $C_{\mathrm{OPT}}(\sigma)$, the cost of the best offline solution on the entire input σ. However, the optimal offline algorithm might end with its servers in any one of many possible configurations! So, it ends up being useful to track the optimal cost conditioned on the final state.

Definition 4.4 (State-Conditioned Optimal Cost)**.** Given a request sequence σ and an initial server configuration S_0, define $C_{\mathrm{OPT}}(\sigma, S)$ to be the minimum cost of any lazy (doesn't move if it doesn't have to) offline algorithm that processes σ starting from S_0 and ends with servers in configuration S. If no such algorithm exists, $C_{\mathrm{OPT}}(\sigma, S) = \infty$.

This function can be computed by dynamic programming, where at each step, the algorithm either stays in its current state (if the request is already covered) or transitions to a new state by moving one server. The overall optimal cost is $\min_S C_{\mathrm{OPT}}(\sigma, S)$.

Now, given an online algorithm $\mathcal{A}$ and a target competitive factor α, we define the *α-residue* as a measure of how much slack $\mathcal{A}$ has relative to α times the optimal.

Definition 4.5 (Residue)**.** Let $\mathcal{A}$ be an online algorithm with cost $C_{\mathcal{A}}(\sigma)$ after processing σ. The α-residue for state S is

$$R_\alpha(\sigma, S) = \alpha \cdot C_{\mathrm{OPT}}(\sigma, S) - C_{\mathcal{A}}(\sigma).$$

Since $\alpha \cdot C_{\mathrm{OPT}}(\sigma) = \alpha \cdot \min_S C_{\mathrm{OPT}}(\sigma, S) \le \alpha \cdot C_{\mathrm{OPT}}(\sigma, S)$ for any state S, we have

$$\alpha \cdot C_{\mathrm{OPT}}(\sigma) - C_{\mathcal{A}}(\sigma) \ge \min_S R_\alpha(\sigma, S).$$

Therefore, $\mathcal{A}$ is α-competitive if and only if there exists a constant β such that $R_\alpha(\sigma, S) \ge -\beta$ for all σ and all states S. In other words, to prove competitiveness, it suffices to show that the residues never drop below some fixed constant.

So, instead of directly bounding the ratio $C_{\mathcal{A}}/C_{\mathrm{OPT}}$, we can instead maintain an invariant on the residue vector. Since the state-conditioned optimal cost $C_{\mathrm{OPT}}(\sigma, S)$ can be updated incrementally via dynamic programming, any online algorithm can compute its own residue vector as it processes requests.

4.1.2 The RES Algorithm

With residues in hand, we can describe the algorithm RES for two servers on an n-vertex metric space (M, d). At any point, RES has two servers on distinct vertices. We adopt the convention that vertex 1 denotes the most recently requested vertex and vertex 2 denotes the other vertex occupied by RES. An offline algorithm must also cover vertex 1, but may place its other server at any of the remaining $n-1$ vertices. So at any step there are $n-1$ finite 2-residues, one for each possible location of the offline algorithm's second server.

Write $R_{1,i}$ for the 2-residue comparing RES to an offline algorithm occupying vertices 1 and i. Algorithm RES initializes

$$R_{1,i} = d(1,i) + 2d(2,i) \quad \text{for each } i \notin \{1\}.$$

When a request arrives at an unoccupied vertex i, RES moves from vertex 1 if

$$\min_k \big\{ R_{1,k} + 2d(k,i) \big\} \ge 2d(1,i) + d(1,2), \tag{4.2}$$

and moves from vertex 2 otherwise.

Algorithm 3 Algorithm RES for 2 Servers on (M, d)

Place servers at initial vertices v_1, v_2; label $1 \leftarrow v_1$, $2 \leftarrow v_2$
$R_{1,i} \leftarrow d(1,i) + 2d(2,i)$ for all $i \neq 1$
for each request at vertex i **do**
  if i is already covered **then**
    continue
  end if
  if $\min_k\{R_{1,k} + 2d(k,i)\} \geq 2d(1,i) + d(1,2)$ **then** ▷ Move from vertex 1
    Move server from 1 to i
    $R_{i,1} \leftarrow \min_k\{R_{1,k} + 2d(k,i)\} - d(1,i)$
    $R_{i,j} \leftarrow R_{1,j} - d(1,i)$ for all $j \notin \{1,i\}$
    Relabel: $1 \leftarrow i$, vertex 2 unchanged
  else ▷ Move from vertex 2
    Move server from 2 to i
    $R_{i,2'} \leftarrow \min_k\{R_{1,k} + 2d(k,i)\} - d(2,i)$
    $R_{i,j} \leftarrow R_{1,j} + 2d(1,i) - 2d(2,i)$ for all $j \notin \{2',i\}$
    Relabel: $1 \leftarrow i$, $2 \leftarrow$ old vertex 1
  end if
end for

To prove that RES is 2-competitive, Manasse, McGeoch, and Sleator [MMS90] maintain an invariant on the pairwise sums of residues. Define the function

$$Y(a,b,c,e) = \min\big\{2d(a,b) + 2d(c,e), 2d(a,c) + 2d(b,e), 2d(a,e) + 2d(b,c)\big\}.$$

This is symmetric in its arguments (permuting them does not change the value, since the minimum ranges over all three perfect matchings of four points). The triangle inequality ensures

$$Y(u,v,w,x) + 2d(u,y) \geq Y(y,v,w,x). \tag{4.3}$$

Theorem 4.6 (RES, [MMS90])**.** *RES is 2-competitive for the symmetric 2-server problem on any n-vertex metric space.*

Proof sketch. The proof shows by induction that after every request,

$$R_{1,i} + R_{1,j} \geq Y(1,2,i,j) \tag{4.4}$$

for every pair of vertices i, j (including $i = j$). Since $Y \geq 0$, this gives a constant lower bound on all residues, which by our earlier discussion implies 2-competitiveness.

With the initial values $R_{1,i} = d(1,i) + 2d(2,i)$:

$$R_{1,i} + R_{1,j} = d(1,i) + d(1,j) + 2d(2,i) + 2d(2,j).$$

To verify (4.4), consider whichever of the three matchings achieves $Y(1,2,i,j)$. If the minimum is $2d(1,j) + 2d(2,i)$, then $2d(1,i) + 2d(2,j) \geq 2d(1,j) + 2d(2,i)$, which gives $d(1,i) - d(1,j) \geq d(2,i) - d(2,j)$, so $R_{1,i} + R_{1,j} - Y = d(1,i) - d(1,j) + 2d(2,j) \geq d(2,i) - d(2,j) + 2d(2,j) = d(2,i) + d(2,j) \geq 0$. The other two cases follow by a symmetric argument.

Suppose the invariant holds and a request arrives at unoccupied vertex i. We consider the two cases of the decision rule (4.2).

Case 1: RES moves from vertex 1. Vertex i becomes the new vertex $1'$, and vertex 2 is unchanged. The updated residues are

$$R_{1',j} = \begin{cases} \min_k\{R_{1,k} + 2d(k,i)\} - d(1,i) & \text{if } j = 1, \\ R_{1,j} - d(1,i) & \text{otherwise.} \end{cases}$$

To verify (4.4) for the new labeling, one checks all pairs (j,l) using the old invariant, the triangle inequality, and property (4.3).

Case 2: RES moves from vertex 2. Vertex i becomes the new $1'$, and the old vertex 1 becomes the new $2'$. The updated residues are

$$R_{1',j} = \begin{cases} \min_k\{R_{1,k} + 2d(k,i)\} - d(2,i) & \text{if } j = 2', \\ R_{1,j} + 2d(1,i) - 2d(2,i) & \text{otherwise.} \end{cases}$$

The most delicate sub-case is verifying (4.4) for pairs j, l with $j, l \neq 2'$. Here one expands $R_{1',j}+R_{1',l}$ using the old invariant for two separate pairs, applies the decision rule (which in this case gives $\min_k\{R_{1,k}+2d(k,i)\} < 2d(1,i) + d(1,2)$), and finishes with the triangle inequality. The full case analysis involves six sub-cases and is carried out in [MMS90]. ■

The intuition behind RES is somewhat similar to the potential function argument we used for DC, as the residues encode a "budget" measuring how much slack RES has relative to twice the optimal cost, and the decision rule chooses which server to move so as to best preserve this budget. The Y function captures the minimum cost of matching two pairs of vertices, and the invariant (4.4) ensures that RES's accumulated slack always exceeds this minimum.

The same paper also describes an algorithm called BAL (short for "balance") for the case of $n-1$ servers on an n-vertex graph. BAL maintains, for each server, the total cumulative distance D_i it has traveled since the start. When a request arrives at an uncovered vertex j, BAL moves the server at vertex i that minimizes $D_i + d(i,j)$. In other words, BAL always moves whichever server would have the smallest total distance traveled after the move, spreading the work evenly across servers. Using a residue argument similar in spirit to (but simpler than) the one for RES, Manasse, McGeoch, and Sleator [MMS90] show that BAL is $(n-1)$-competitive for $n-1$ servers on n vertices.

Algorithm 4 Balance Algorithm (BAL) for $n-1$ Servers on n Vertices

```
Initialize servers at n − 1 distinct vertices of (M, d)
Set D_i ← 0 for each server at vertex i
for each request at vertex j do
    if j is already covered by a server then
        continue
    end if
    i* ← arg min_i{D_i + d(i, j)} among all vertices i with a server
    Move server from i* to j
    D_{i*} ← D_{i*} + d(i*, j)
end for
```

So at this point we have two strongly competitive algorithms, RES for $k = 2$ and BAL for $k = n-1$, each matching the k-server lower bound exactly (Theorem 4.3). I find these satisfying results, but they sit at the two extremes of the problem. For $k = 2$, RES needs to reason about pairs of servers, which is manageable. For $k = n - 1$, BAL benefits from the fact that only one vertex is ever uncovered. The difficult case is everything in between, with a moderate number of servers on a relatively large metric space, where neither trick can apply.

This is what makes the k-server conjecture, stated by Manasse, McGeoch, and Sleator in the same paper [MMS88], so compelling:

Conjecture 4.7 (k-Server Conjecture)**.** *For every metric space with more than k points, there exists a deterministic k-competitive online algorithm for the k-server problem.*

Given the lower bound we proved, k is the best one could hope for. For six years[2] after the conjecture was posed, the best known upper bound for general metrics was exponential in k, as one could always view the k-server problem as a task system on $\binom{n}{k}$ states, giving a competitive ratio exponential in k, but this is absurdly large! The situation changed dramatically in 1995, when Koutsoupias and Papadimitriou [KP95] introduced the *work function algorithm* and proved it $(2k-1)$-competitive on any metric space. This is tantalizingly close to k, and as of today, it remains the best deterministic upper bound for general metrics.

[2]Technically, it could be seven years, since it's between the conference and the journal paper. Six seven.

4.2 The Work Function Algorithm

The idea behind the work function algorithm (WFA) is so simple you can explain it while drunk[3]: it combines the greedy instinct of "move cheaply" with the global awareness of "don't stray too far from what the optimal solution would do." To make this precise, we need the state-conditioned optimal cost from the previous section.

Recall that we write $C_{\mathrm{OPT}}(\sigma, X)$ for the minimum cost of serving the request sequence σ and ending in configuration X. This function is called the *work function* after σ requests and is typically denoted w_t after the t-th request:

$$w_t(X) = C_{\mathrm{OPT}}((r_1, \ldots, r_t), X).$$

This tells us the cheapest way to serve all requests so far and end up with servers at the locations described by X. The work function satisfies the following key property. If a configuration Y already covers the request r_t, then $w_t(Y) = w_{t-1}(Y)$, since an optimal algorithm reaching Y can just stay there and serve r_t for free. For a configuration X that may or may not cover r_t, we have

$$w_t(X) = \min_{Y \ni r_t} \big\{ w_t(Y) + d(Y, X) \big\},$$

where $Y \ni r_t$ means Y is a configuration containing a server at r_t, and $d(Y, X)$ is the minimum-cost way to move servers from configuration Y to configuration X. This says that the cheapest way to serve t requests and end at X is to first reach some configuration Y that covers r_t (paying $w_t(Y)$), then rearrange to X (paying $d(Y, X)$). This property follows from the triangle inequality and will be crucial in the proof.

The WFA uses this function to decide how to move. Let X_{t-1} be the algorithm's configuration before the t-th request r_t. The WFA serves r_t by choosing the configuration X_t that minimizes the sum of the work function value and the cost of getting there:

$$X_t = \arg\min_{X \ni r_t} \big\{ w_t(X) + d(X_{t-1}, X) \big\}.$$

In practice, since we want a lazy algorithm (one that only moves a single server per request if needed, and in fact all optimal algorithms can be modified to be lazy), this amounts to choosing which server $s \in X_{t-1}$ to send to r_t. The configuration X_t is then X_{t-1} with s replaced by r_t, and the cost is $d(s, r_t)$. So the decision rule is among all servers that could move to r_t, pick the one s that minimizes $w_t(X_{t-1} \cup \{r_t\} \setminus \{s\}) + d(s, r_t)$.

Algorithm 5 Work Function Algorithm (WFA)

```
Initialize servers at configuration X_0
Set w_0(X_0) = 0 and w_0(X) = d(X_0, X) for all configurations X
for each request r_t do
    if r_t ∈ X_{t-1} then
        X_t ← X_{t-1}                                   ▷ Request already covered
    else
        Update w_t(X) = min_{Y ∋ r_t} {w_{t-1}(Y) + d(X, Y)} for all X
        X_t ← arg min_{X ∋ r_t} {w_t(X) + d(X_{t-1}, X)}
        Move the appropriate server from X_{t-1} to r_t
    end if
end for
```

Notice that WFA doesn't commit to any fixed heuristic like "move the nearest server" or "balance cumulative distances." Instead, it consults the entire history (compressed into the work function) and asks "which move leaves me closest to the optimal, accounting for both the quality of the resulting configuration and the cost of reaching it?" This is what makes it so powerful but is also why it is expensive to implement, since maintaining w_t requires tracking exponentially many configurations. But our concern here is the competitive ratio, not the running time.

[3]No comment on how the author knows. What stays in the UK stays in the UK.

Theorem 4.8 (Koutsoupias–Papadimitriou [KP95])**.** *The Work Function Algorithm is* $(2k-1)$*-competitive for the* k*-server problem on any metric space.*

Proof. Let $X_0, X_1, \ldots, X_T$ be the configurations chosen by WFA for the request sequence $\sigma = (r_1, \ldots, r_T)$. The proof tracks how $w_t(X_t)$, the work function evaluated at WFA's own configuration, evolves over time.

Consider the difference $w_t(X_t) - w_{t-1}(X_{t-1})$ at step t. We decompose this by adding and subtracting $w_t(X_{t-1})$:

$$w_t(X_t) - w_{t-1}(X_{t-1}) = \underbrace{\big(w_t(X_t) - w_t(X_{t-1})\big)}_{\text{WFA's choice}} + \underbrace{\big(w_t(X_{t-1}) - w_{t-1}(X_{t-1})\big)}_{\text{new request's impact}}. \tag{4.5}$$

The first term measures how WFA's move from X_{t-1} to X_t changes the work function value. By the WFA decision rule, X_t minimizes $w_t(X) + d(X_{t-1}, X)$ over configurations X covering r_t, while X_{t-1} is one such configuration only if $r_t \in X_{t-1}$ (in which case $X_t = X_{t-1}$ and the term is zero). When $r_t \notin X_{t-1}$, we apply the property $w_t(X_{t-1}) = \min_{Y \ni r_t}\{w_t(Y) + d(Y, X_{t-1})\}$. The WFA decision rule says X_t minimizes $w_t(X) + d(X_{t-1}, X)$ over all $X \ni r_t$. Since $X_t \ni r_t$, it is a candidate for Y in the property, so $w_t(X_{t-1}) \le w_t(X_t) + d(X_t, X_{t-1})$. For the reverse direction, let Y^* be the configuration achieving the minimum in $w_t(X_{t-1}) = \min_{Y \ni r_t}\{w_t(Y) + d(Y, X_{t-1})\}$, so that $w_t(X_{t-1}) = w_t(Y^*) + d(Y^*, X_{t-1})$. Since $Y^* \ni r_t$, the WFA decision rule gives $w_t(X_t) + d(X_{t-1}, X_t) \le w_t(Y^*) + d(X_{t-1}, Y^*) = w_t(X_{t-1})$, i.e., $w_t(X_{t-1}) \ge w_t(X_t) + d(X_t, X_{t-1})$. Thus equality holds:

$$w_t(X_{t-1}) = w_t(X_t) + d(X_t, X_{t-1}),$$

which gives $w_t(X_t) - w_t(X_{t-1}) = -d(X_t, X_{t-1})$. In other words, the first term equals the negative of WFA's movement cost at step t.

For the second term, define the *extended cost*

$$\mathrm{EXT}_t = \max_X \big\{ w_t(X) - w_{t-1}(X) \big\},$$

the maximum increase in the work function over all configurations. Since X_{t-1} is one particular configuration, $w_t(X_{t-1}) - w_{t-1}(X_{t-1}) \le \mathrm{EXT}_t$.

Substituting both bounds into (4.5) and summing over all T steps:

$$w_T(X_T) - w_0(X_0) \le \sum_{t=1}^{T} \mathrm{EXT}_t - \sum_{t=1}^{T} d(X_{t-1}, X_t).$$

Since $w_T(X_T) \ge \min_X w_T(X) = C_{\mathrm{OPT}}(\sigma)$ and $w_0(X_0) = 0$, the left side is at least $C_{\mathrm{OPT}}(\sigma)$. Recognizing $\sum_t d(X_{t-1}, X_t) = C_{\mathrm{WFA}}(\sigma)$, we rearrange:

$$C_{\mathrm{OPT}}(\sigma) + C_{\mathrm{WFA}}(\sigma) \le \sum_{t=1}^{T} \mathrm{EXT}_t. \tag{4.6}$$

The rest of the proof bounds $\sum_t \mathrm{EXT}_t$. This is where the three key ingredients of Koutsoupias and Papadimitriou come in, which they outline in their abstract. First comes quasiconvexity. Then comes duality. Finally, potential.

The work function w_t satisfies a quasiconvexity property, for any two configurations X and Y that differ in the position of a single server, the work function does not have "local maxima" along the path between them. More precisely, if X and Y differ only in that X has a server at a while Y has a server at b, then for any point c on a shortest path from a to b, the configuration Z (which agrees with X and Y except with a server at c) satisfies $w_t(Z) \le \max(w_t(X), w_t(Y))$. This is a structural property of optimal costs that follows from the triangle inequality.

Using quasiconvexity, Koutsoupias and Papadimitriou characterize the configurations that achieve the maximum increase EXT_t. They show that the maximum is achieved at a configuration Y_t that is, in a precise sense, "antipodal" to the request r_t, where the servers are as far from r_t as possible while remaining consistent with the metric structure. In the special case of $n = k + 1$ points, this antipodal configuration is simply $Y_t = M \setminus \{r_t\}$, the unique configuration that avoids the request.

For $n = k + 1$ points, define the potential $\Phi_t = \sum_X w_t(X)$, where the sum runs over all $k+1$ possible configurations. Since work functions are monotone non-decreasing ($w_t(X) \geq w_{t-1}(X)$ for all X), we have $\Phi_t - \Phi_{t-1} = \sum_X (w_t(X) - w_{t-1}(X)) \geq \max_X (w_t(X) - w_{t-1}(X)) = \mathrm{EXT}_t$, so $\sum_t \mathrm{EXT}_t \leq \Phi_T$. Since there are $k+1$ configurations and each satisfies $w_T(X) \leq C_{\mathrm{OPT}}(\sigma) + kD$ (where D is the diameter, or longest path in the graph), we get $\Phi_T \leq (k+1)(C_{\mathrm{OPT}}(\sigma) + kD)$. Plugging into (4.6):

$$C_{\mathrm{WFA}}(\sigma) \leq k \cdot C_{\mathrm{OPT}}(\sigma) + k(k+1)D.$$

So WFA is k-competitive when $n = k + 1$.

For the general case ($n > k+1$), the argument is considerably more involved. The duality lemma tells us that EXT_t is achieved at a minimizer of r_t, defined as a configuration A that minimizes $w_t(X) - \sum_{x \in X} d(r_t, x)$ over all configurations X. Intuitively, this is the configuration whose servers are "as far from the request as possible" relative to the work function.

Koutsoupias and Papadimitriou then define a potential $\Phi(w)$ as the minimum of an expression

$$\Psi(w, U, B_1, \ldots, B_k)$$

over $k+1$ configurations, specifically a "central" configuration U containing the most recent request, and k "satellite" configurations $B_1, \ldots, B_k$. The expression Ψ involves the work function values at these configurations and all pairwise distances between them. The key calculation shows that the change in potential $\Phi(w') - \Phi(w)$ bounds the extended cost EXT_t from above. Telescoping and bounding the initial and final potentials yields:

$$\sum_{t=1}^{T} \mathrm{EXT}_t \leq 2k \cdot C_{\mathrm{OPT}}(\sigma) + k^2 D.$$

Substituting into (4.6):

$$C_{\mathrm{WFA}}(\sigma) \leq (2k-1) \cdot C_{\mathrm{OPT}}(\sigma) + k^2 D,$$

which establishes $(2k-1)$-competitiveness. ∎

Remark (The antipodal viewpoint). There is an appealing geometric way to see why the factor $2k$ arises in the general case, described in the lecture notes of Chawla [Cha07].

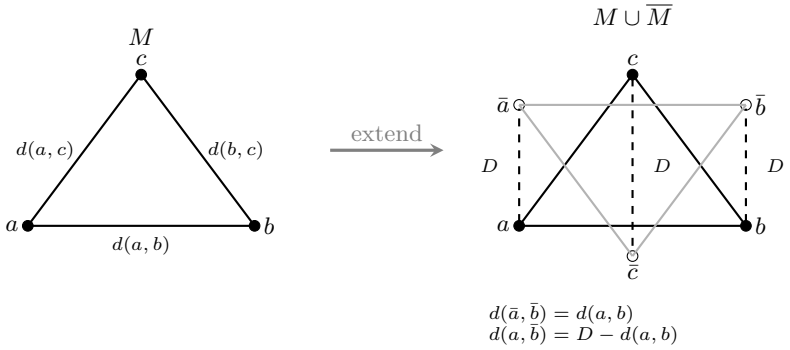

Figure 6. The antipodal construction used in the general case of the WFA proof. Each point $a \in M$ gains an antipode $\bar{a}$ at distance D (the diameter). Antipodes preserve the original distances among themselves. The "worst-case" configuration for EXT_t places all servers at the antipode of the request point.

One constructs an "extended" metric space $M \cup \overline{M}$ by adding, for each point $a \in M$, an *antipode* $\bar{a}$ with distances

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

where D is the diameter of M (see figure 6). One then considers a k-server instance on this $2n$-point space. The key observation is that the duality structure of the extended metric allows one to reduce the general-n analysis to the simpler $n = k + 1$ case; informally, the antipodal copies "double" the space so that the

BAL-style argument applies. The duality lemma's "minimizer" corresponds in this picture to placing all k servers at the antipode of the request, which is the configuration that is literally as far as possible from r_t. One can then bound $\sum_t \mathrm{EXT}_t$ by pairing each request with the OPT server that eventually serves it, giving the same $2k$ factor.

Recall that the lower bound is k, and WFA gives us an upper bound of $2k-1$. The WFA is widely conjectured to actually be k-competitive. For other metric spaces, we know k-competitive algorithms. Manasse, McGeoch, and Sleator [MMS88] settled the case $k=2$ entirely, and specific k-competitive algorithms are known for the line [CKPV91] and trees [CL91] (the DC algorithm in both cases). For WFA specifically, Bartal and Koutsoupias [BK04] proved it is k-competitive on the line, on weighted stars, and on all metrics with $k+2$ points. But for general metrics, closing the factor-of-two gap remains open.

Anyway. What happens if we start gambling?

5 Randomized k-server

The deterministic picture, as we left it, is essentially stuck at $2k-1$. Given that the marking algorithm gave us an exponential improvement from deterministic k to randomized $H_k \approx \ln k$ for paging, it is natural to ask whether randomization can help on all metrics, not just the uniform one.

5.1 The Randomized Conjecture

Recall that paging is the k-server problem on the uniform metric, and the marking algorithm achieves $O(\log k)$ competitive ratio there. Fiat et al. [FKL$^+$91] proved that no randomized paging algorithm can achieve competitive ratio better than H_k, and McGeoch and Sleator [MS91] gave a matching H_k-competitive algorithm. Since paging is a special case of k-server, any randomized k-server algorithm must have competitive ratio at least $H_k = \Omega(\log k)$ on at least one metric. The prevailing belief for many years was that the uniform metric is the hardest case for randomized algorithms, and that this logarithmic barrier is the truth everywhere.

Conjecture 5.1 (Randomized k-Server Conjecture)**.** *For every metric space with more than k points, there exists a randomized $O(\log k)$-competitive online algorithm for the k-server problem.*

This conjecture appears as folklore in the online algorithms literature, see Koutsoupias's survey [Kou09] for a discussion. We will see that this conjecture is false, but getting there required decades of work from both the upper and lower bound sides.

5.2 Lower Bounds

There are two kinds of lower bounds to track: *universal* bounds (holding on all metrics with more than k points) and *existential* bounds (showing that some metric requires a high competitive ratio). The first superconstant lower bound for randomized k-server was proved by Blum, Karloff, Rabani, and Saks [BKRS92].

Theorem 5.2 (Blum–Karloff–Rabani–Saks [BKRS92])**.** *For every metric space with at least $k+1$ points, the randomized competitive ratio for the k-server problem against an oblivious adversary is $\Omega\left(\sqrt{\log k / \log\log k}\right)$.*

Their proof uses a decomposition theorem that reduces the problem to smaller subproblems and establishes lower bounds on each piece. This universal bound was later superseded by work of Bartal, Bollobás, and Mendel [BBM06], who proved Ramsey-type theorems for metric spaces for the k-server problem, and they showed the following theorem using quantitative bounds of Bartal, Linial, Mendel, and Naor [BLMN05].

Theorem 5.3 (Metric Ramsey Theorem [BBM06, BLMN05])**.** *Every sufficiently large n-point metric space contains a subspace that is approximately a hierarchically separated tree (HST), which yields an $\Omega(\log n / \log\log n)$ lower bound for metrical task systems (MTS) on any n-point metric space.*

Since k-server on $k+1$ points is a special case of MTS, this yields a universal k-server lower bound of $\Omega(\log k / \log\log k)$, nearly matching the conjectured $\Theta(\log k)$. Equally importantly, the BBM framework established HSTs as the key structure underlying hard instances, a perspective that would drive the next two decades of progress.

5.3 Hierarchically Separated Trees

HSTs are the single most important class of metric spaces in the randomized k-server story, so we should define them properly. A *τ-hierarchically separated tree* (τ-HST) is a rooted tree T with positive vertex weights $w : V \to \mathbb{R}_+$ that are non-increasing along every root-to-leaf path, decreasing by a factor of at least τ at each step. The metric space is defined on the *leaves* of T, with the distance between two leaves being the weight of their least common ancestor.

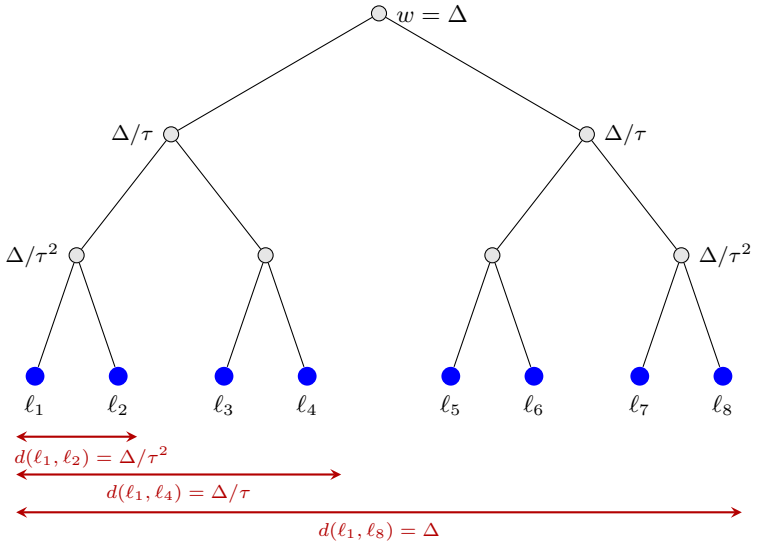

Figure 7. A τ-HST of depth 3 with branching factor 2. Internal nodes carry weights that decrease by a factor of τ at each level. The metric is defined on the leaves (blue): the distance between two leaves equals the weight of their least common ancestor. Leaves ℓ_1 and ℓ_2 diverge at depth 3 and are close (Δ/τ^2), while ℓ_1 and ℓ_8 diverge at the root and are far (Δ).

For example, the uniform metric on n points is a (degenerate) HST with a single root whose children are all the leaves, each at weight 1. A more interesting HST might have depth d with branching factor b and weights that decrease geometrically: the root has weight Δ, its children have weight Δ/τ, and so on down to Δ/τ^{d-1} at the bottom. The key feature is that the distance between two leaves depends only on their depth of divergence, as leaves that diverge high up in the tree are far apart, while leaves that share a long common path from the root are close together.

Why are HSTs central? Because of a remarkable theorem of Bartal [Bar96], who showed that every n-point metric space can be probabilistically embedded into HSTs with $O(\log^2 n)$ distortion. This was later made tight by Fakcharoenphol, Rao, and Talwar [FRT04]:

Theorem 5.4 (HST Embedding [FRT04])**.** *Every n-point metric space can be probabilistically embedded into a distribution over HSTs with distortion $O(\log n)$.*

This means that if you have an α-competitive algorithm for k-server on HSTs, you can turn it into an $O(\alpha \log n)$-competitive algorithm on any n-point metric space, at the cost of a single $O(\log n)$ factor from the embedding. So solving k-server on HSTs is "almost" the same as solving it in general.

5.4 The Polylogarithmic Upper Bound

The first major breakthrough in the randomized regime came from Bansal, Buchbinder, Madry, and Naor [BBMN15], who gave the first polylogarithmic-competitive randomized algorithm for k-server on general metrics in FOCS 2011, though it was published in 2015 proper:

Theorem 5.5 (Bansal–Buchbinder–Madry–Naor [BBMN15])**.** *There exists a randomized $O(\log^3 n \cdot \log^2 k \cdot \log\log n)$-competitive algorithm for the k-server problem on any n-point metric space.*

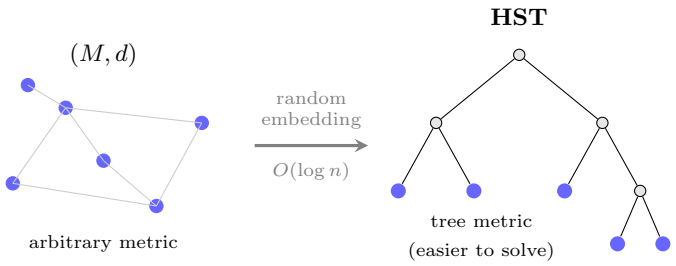

Figure 8. The HST embedding reduction. Any n-point metric can be randomly embedded into a distribution over HSTs with $O(\log n)$ expected distortion. An α-competitive algorithm on HSTs yields an $O(\alpha \log n)$-competitive algorithm on the original metric.

Their algorithm works by reducing the problem to HSTs (via Bartal's embedding framework) and then solving recursively on the tree. Since the competitive ratio involves both n and k, it only improves on the deterministic $2k - 1$ bound when n is sub-exponential in k. Still, this was a landmark result, as it was the first randomized algorithm to break through the linear-in-k barrier for a wide range of metric spaces. The competitive ratio's dependence on n, however, meant that the randomized conjecture, which demands $O(\log k)$ independent of the metric, remained open.

The next breakthrough came from Bubeck, Cohen, Y. T. Lee, J. R. Lee, and Madry [BCL$^+$18], who introduced the technique of multiscale entropic regularization.

Theorem 5.6 (Bubeck–Cohen–Lee–Lee–Madry [BCL$^+$18])**.** *There exists a randomized $O(\log^2 k)$-competitive algorithm for the k-server problem on any HST.*

This was the first time the competitive ratio on HSTs was bounded purely in terms of k. The idea is to formulate the k-server problem as a continuous optimization problem on a polytope of "fractional server allocations," and then use mirror descent with an entropy-based potential function that operates at multiple scales of the tree simultaneously. Coester and Lee [CL22] further developed these entropic regularization techniques in the related setting of metrical task systems. Combined with Bartal's static HST embedding, BCLLM gives $O(\log^2 k \cdot \log n)$ on n-point metric spaces. BCLLM also give a dynamic embedding yielding $O(\log^3 k \cdot \log \Delta)$, where Δ is the aspect ratio (the ratio of the largest to smallest nonzero distance), which removes the dependence on n at the cost of depending on Δ.

A natural next step would be to remove the dependence on the geometry entirely. Lee [Lee18] proposed an elegant approach using "fusible" HSTs, which were dynamic embeddings that adapt to the request sequence rather than being fixed in advance. The key idea is that by allowing the HST approximation to evolve over time, one can avoid the $O(\log n)$ distortion penalty of a static embedding and obtain a competitive ratio of $O(\log^6 k)$ depending only on k. However, a non-trivial gap was subsequently found in the proof, and as of this writing the claimed result has been retracted by the author.[4] The fusible HST framework remains a promising direction, but the $O(\text{poly}(\log k))$ competitive ratio on general metrics is not yet established.

On general metric spaces, the situation is even less resolved. The best standing upper bound is the $O(\log^2 k \cdot \log n)$ obtained by combining BCLLM's HST algorithm [BCL$^+$18] with the $O(\log n)$-distortion embedding of FRT [FRT04]. BCLLM also give $O(\log^3 k \cdot \log \Delta)$ via a dynamic embedding, where Δ is the aspect ratio. The randomized conjecture predicted $O(\log k)$.

6 The Randomized Conjecture is False

In 2022, Bubeck, Coester, and Rabani [BCR23] proved several bombshell lower bounds that completely reshaped the landscape.

Theorem 6.1 (Bubeck–Coester–Rabani [BCR23])**.** *The following lower bounds hold for the randomized k-server problem:*

[4] See `https://homes.cs.washington.edu/~jrl/papers/kserver-erratum.html`. Lee notes that a proposed fix exists but its integration with the rest of the argument is not yet complete.

1. *There exist $(k+1)$-point metric spaces on which the randomized competitive ratio is $\Omega(\log^2 k)$.*

2. *For all n-point metric spaces with $n > k$, the randomized competitive ratio is at least $\Omega(\log k)$, and this is asymptotically tight.*

The first bound kills the randomized k-server conjecture! On certain metrics, no randomized algorithm can do better than $\Omega(\log^2 k)$, whereas the conjecture predicted $O(\log k)$ everywhere. The second bound is also significant, as it improves the previous universal bound of $\Omega(\log k / \log\log k)$ (due to BBM [BBM06] and BLMN [BLMN05]) to a tight $\Omega(\log k)$ on all metrics, matching the paging lower bound.

6.1 How to Build a Hard Metric

The construction that achieves the $\Omega(\log^2 k)$ lower bound is recursive and deeply clever. We will describe the high-level idea without attempting to reproduce the full proof.

Bubeck, Coester, and Rabani construct a sequence of metric spaces $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \ldots$ of growing size, parameterized by a sequence of natural numbers $m_0 \le m_1 \le m_2 \le \cdots$. Each metric is defined as the shortest-path metric of an underlying graph. The base case $\mathcal{M}_0$ is simply a single edge of weight 1. The metric $\mathcal{M}_1$ is a cycle of $6m_0$ copies of $\mathcal{M}_0$, which is a cycle with $6m_0$ edges. Two antipodal vertices s and t are designated as "special," with $\operatorname{diam}(\mathcal{M}_1) = d_1(s,t)$.

More generally, $\mathcal{M}_{w+1}$ is built from $\mathcal{M}_w$ in the same way that $\mathcal{M}_1$ was built from $\mathcal{M}_0$: take a cycle of $6m_w$ edges, and replace each edge $\{u, v\}$ by a copy of $\mathcal{M}_w$ whose special vertices are identified with u and v. The result is a "cycle of cycles" with a cycle of $6m_w$ copies of $\mathcal{M}_w$, glued together at their special vertices. The new special vertices s and t in $\mathcal{M}_{w+1}$ are chosen to be antipodal, so that $\operatorname{diam}(\mathcal{M}_{w+1}) = d_{w+1}(s,t)$. Notice that $\mathcal{M}_{w+1}$ can be viewed as consisting of a left path of $3m_w$ copies of $\mathcal{M}_w$ and a right path of another $3m_w$ copies, forming the two halves of the cycle from s to t.

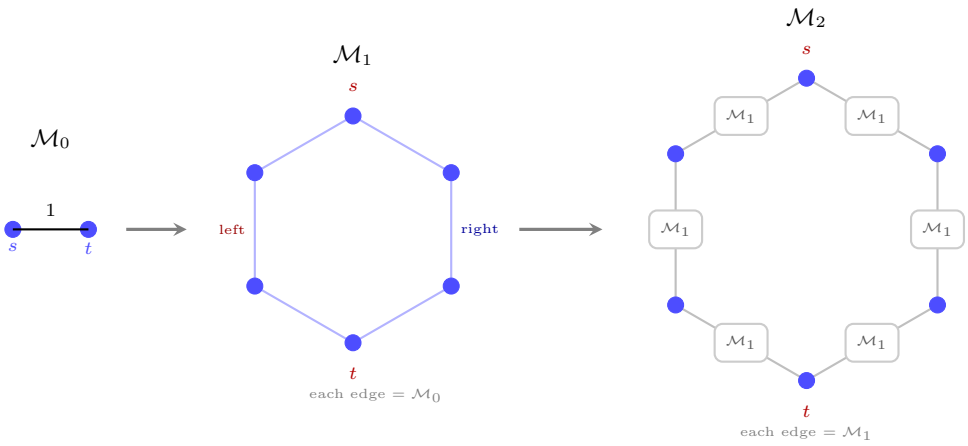

Figure 9. The recursive metric space construction of Bubeck, Coester, and Rabani [BCR23]. The base case $\mathcal{M}_0$ is a single edge. $\mathcal{M}_1$ is a cycle of $6m_0$ copies of $\mathcal{M}_0$, with antipodal special vertices s and t. $\mathcal{M}_2$ is a cycle of $6m_1$ copies of $\mathcal{M}_1$, each copy replacing an edge of the cycle. The two halves from s to t form a left path and a right path.

The key insight is that this "cycle of cycles" structure creates multiple scales at which the adversary can challenge the algorithm. On the uniform metric (paging), there is only one scale, and the best the adversary can do is force a $\log k$ factor. But on $\mathcal{M}_w$, the adversary can exploit the cycle structure at each level of the recursion independently.

At the top level, the adversary forces the algorithm to commit its servers to one half of the cycle (say the left path of copies of $\mathcal{M}_{w-1}$), then issues requests on the other half. Within each copy of $\mathcal{M}_{w-1}$, the adversary recurses, exploiting the cycle structure at the next scale down. Since the algorithm must spread its mass across both halves of the cycle, the adversary can always find a half where the algorithm is

underrepresented[5]. With $d \sim \log k$ levels of recursion, the lower bound accumulates to $\Omega(\log^2 k)$.

6.2 Where Things Stand

Let us summarize the current state of affairs for the randomized k-server problem. On HSTs, we have:

$$\Omega(\log k) \leq \text{randomized competitive ratio on HSTs} \leq O(\log^2 k).$$

The upper bound is from BCLLM [BCL+18]. The lower bound of $\Omega(\log k)$ already follows from the paging lower bound, since the uniform metric is a (degenerate) 1-HST. Closing the gap between $\log k$ and $\log^2 k$ on HSTs remains open.

On general metric spaces, the situation is even less resolved. The best standing upper bound is the $O(\log^2 k \cdot \log n)$ obtained by combining BCLLM's HST algorithm [BCL+18] with the $O(\log n)$-distortion embedding of FRT [FRT04], which still depends on n. BCLLM also give $O(\log^3 k \cdot \log \Delta)$ via a dynamic embedding, where Δ is the aspect ratio. Lee's fusible HST approach [Lee18] would give $O(\log^6 k)$ independent of both n and Δ, but the proof remains incomplete. From below, BCR [BCR23] construct specific $(k+1)$-point metric spaces (not HSTs, but cycle-of-cycles graphs) on which the competitive ratio is $\Omega(\log^2 k)$, and they prove that on all metrics with more than k points, the competitive ratio is $\Omega(\log k)$, improving the previous universal bound of $\Omega(\log k / \log\log k)$ due to BBM and BLMN, and matching the paging lower bound. Whether a competitive ratio depending only on k is achievable on all metrics, let alone what the right polynomial in $\log k$ is, remains wide open.

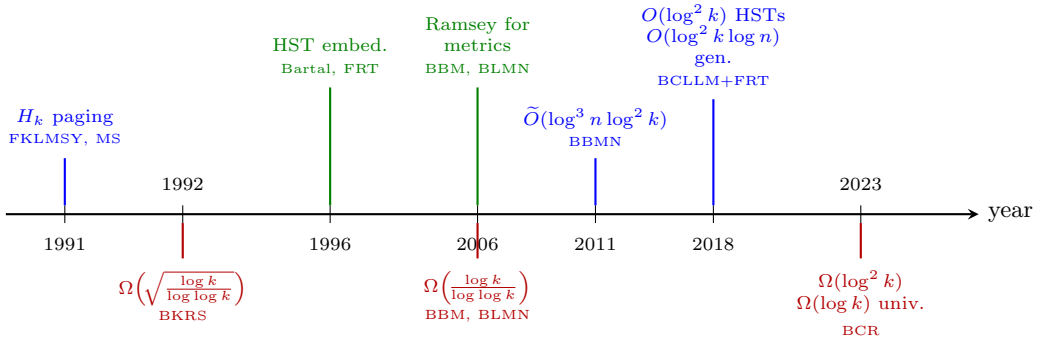

Figure 10. Timeline of key results in the randomized k-server problem. Upper bounds in blue (above), lower bounds in red (below), and structural results in green (top).

It is worth pausing to appreciate the arc of this story. For over thirty years, the community believed that randomization should reduce the k-server competitive ratio from $\Theta(k)$ down to $O(\log k)$, matching paging. The intuition was that the uniform metric, where every point looks the same, should be the hardest case for randomized algorithms, since the adversary has the least structure to exploit. Bubeck, Coester, and Rabani showed that this intuition is exactly backwards, as metrics with more structure are harder, because the adversary can exploit each scale independently. And with Lee's result still incomplete, even the basic question of whether an n-independent polylogarithmic competitive ratio is achievable on all metrics remains unresolved.

7 The k-taxi Problem, Online Algorithms, and Further Questions

We began this paper with a 150-billion-dollar rideshare company. The k-server problem captures only half of Uber's problem, since their drivers don't just teleport to a passenger and vanish, but they pick the passenger up at a source and drive them to a destination. This motivates a natural generalization.

[5]Take the road less travelled by, à la Robert Frost.

7.1 The k-taxi Problem

The k-taxi problem was introduced by Fiat, Rabani, and Ravid [FRR90], building on a formulation of Karloff. We are given an n-point metric space (M, d) and k taxis (servers). Requests now arrive as *pairs* (s_q, t_q), where s_q is a source (the pickup) and t_q is a sink (the destination). To serve a request, the algorithm must move some taxi to s_q, then transport it to t_q, before the next request arrives.

There are two versions of the problem depending on what we charge for. In the easy version, we include the cost of driving from s_q to t_q in the objective, and there is a straightforward constant-factor reduction to k-server. The more interesting hard version only charges for the *overhead*, or the distance taxis travel while not serving a request. This is what Uber actually wants to minimize, and it is what we mean by k-taxi from here on.

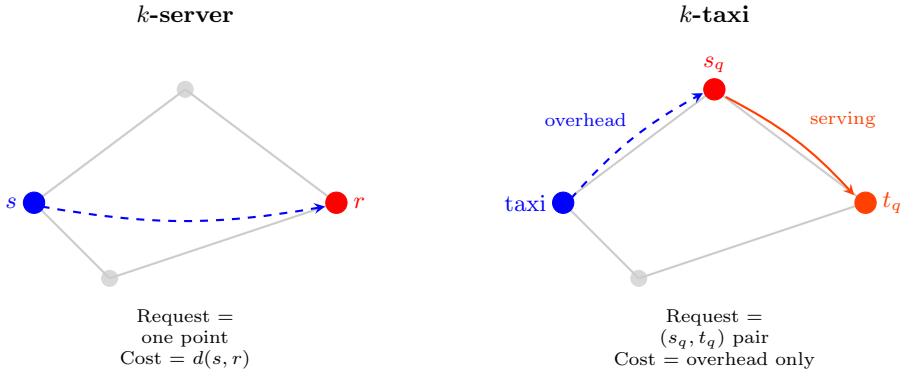

Figure 11. Comparison of k-server and k-taxi. In k-server, a request is a single point and the cost is the distance traveled to reach it. In k-taxi, a request is a source-sink pair; the taxi must reposition to the source (overhead, which we pay for) and then serve the passenger to the sink (free). The k-server problem is the special case where $s_q = t_q$ for every request.

Notice that k-server is the special case where $s_q = t_q$ for every request, so that there is no "serving" phase and all movement is overhead. One might hope that k-taxi is only mildly harder than k-server. In the deterministic setting, this hope is badly wrong. Coester and Koutsoupias [CK19] showed that the deterministic competitive ratio for k-taxi is $\Omega(2^k)$, which is exponentially worse than the $\Theta(k)$ picture for k-server. Intuitively, the serving phase reshuffles the taxis in ways that the algorithm cannot control, and a clever adversary can exploit this to create exponentially many traps.

The randomized setting is more forgiving. Prior to 2024, several randomized algorithms were known, but all had competitive ratios that were either exponential in k or polynomial in n:

- $O(2^k \log n)$, due to Coester and Koutsoupias [CK19], based on a $(2^k - 1)$-competitive algorithm on HSTs.
- $O((n \log k)^2 \log n)$, due to Bubeck, Buchbinder, Coester, and Sellke [BBCS21], via a reduction to metrical service systems under transformations of the metric space.
- $2^{O(\sqrt{\log \Delta \cdot \log k})} \cdot \log_\Delta n$, due to Buchbinder, Coester, and Naor [BCN21], using ideas from the Double Coverage algorithm, where Δ is the aspect ratio (ratio between longest and shortest distance in the metric space) of the metric.

The best lower bound is $\Omega(\log^2 k)$, inherited from the k-server lower bound of Bubeck, Coester, and Rabani [BCR23]. None of the upper bounds above are polylogarithmic in all parameters simultaneously, so a large gap persisted.

Gupta, Kumar, and Panigrahi [GKP24] resolved this at SODA 2024 by giving the first polylogarithmic competitive ratio for k-taxi.

Theorem 7.1 (Gupta–Kumar–Panigrahi [GKP24]). *There exists a randomized $O(\log^3 \Delta \cdot \log^2(nk\Delta))$-competitive algorithm for the k-taxi problem on any n-point metric space with aspect ratio Δ.*

Their approach is a departure from the combinatorial techniques used by all previous k-taxi algorithms. They formulate a new covering LP relaxation for k-taxi on HSTs, derived from the min-cost flow formulation of the offline problem, and solve it with a hierarchical primal-dual algorithm that exploits the compositionality of the LP constraints across subtrees of the HST. The rounding step adapts existing techniques from the HST k-server literature [BBMN15, BCL$^+$18].

That the dependence on Δ remains is notable. For metric spaces with polynomially bounded aspect ratio, this gives a truly polylogarithmic guarantee. But removing the $\log \Delta$ dependence entirely, which would put k-taxi on equal footing with k-server (where $O(\log^2 k)$ is known on HSTs), remains an open and interesting challenge.

7.2 The Weighted k-server Problem

Another natural generalization gives the servers different weights. In the *weighted k-server problem*, each server i has a weight $w_i > 0$, and the cost of moving server i a distance d is $w_i \cdot d$. This models settings where resources have different deployment costs, where you value some Ubers more than others.

On general metrics, the weighted k-server problem is notoriously difficult, and even the uniform metric (where all pairwise distances are 1) is hard. Gupta, Kumar, and Panigrahi [GKP23] studied this offline case at APPROX 2023 and proved a striking collection of results, which we summarize here. On the computational side, they showed that assuming the Unique Games Conjecture, no polynomial-time algorithm can achieve a sub-polynomial approximation factor for the offline weighted k-server problem on the uniform metric, even with c-resource augmentation for any $c < 2$. That's a lot of words, but I'm already almost hitting 30 pages, so I'm not covering them all. Maybe CS254B.

On the online side, Ayyadevara and Chiplunkar [AC21] showed that the randomized competitive ratio for weighted k-server is at least exponential in k, even on the uniform metric. Very recently, Bijoy, Mondal, and Chiplunkar [BMC26] gave a matching $\exp\big(O(k^2)\big)$-competitive randomized upper bound on uniform metrics, resolving the doubly-vs-singly exponential gap. This is a dramatic contrast with the unweighted case, where $O(\log^2 k)$ is achievable on HSTs.

7.3 Learning-Augmented Algorithms

All of the results we have discussed live squarely in the worst-case framework. The adversary is omniscient (or at least oblivious), the algorithm knows nothing about the future, and the competitive ratio measures performance against a pathological input sequence that the algorithm will almost certainly never encounter in practice. Hopefully we are unlikely to see fractal-like metric spaces in the real world.

This motivates a different question: what if the algorithm has access to (possibly imperfect) predictions about the future? The *learning-augmented* or *algorithms with predictions* framework, introduced by Lykouris and Vassilvitskii [LV21] and Purohit, Svitkina, and Kumar [KPS18], formalizes this idea. The algorithm receives a prediction (say, from a machine-learned model) alongside each request, and the goal is to design algorithms that are simultaneously near-optimal when the predictions are accurate, and no worse than the best classical online algorithm when the predictions are arbitrarily bad.

Ideally, the competitive ratio degrades smoothly as a function of the prediction error η, interpolating between these two extremes.

For paging, Lykouris and Vassilvitskii [LV21] showed how to modify the marking algorithm to incorporate predictions of the next arrival time for each page. Their algorithm achieves a competitive ratio that decreases with the prediction error and is always at most $O(\log k)$, matching the best unconditional bound. Rohatgi [Roh20] and Wei [Wei20] subsequently improved the error dependence. In particular, this means that a learning-augmented paging algorithm can *beat* the H_k lower bound when predictions are good, while never performing worse than the classical guarantee.

For the k-server problem on the line, Lindermayr, Megow, and Simon [LMS25] gave a learning-augmented variant of the Double Coverage algorithm. Their algorithm integrates a user-defined confidence parameter that controls how much to trust the predictions: when the confidence is high and the predictions are accurate, the algorithm achieves 1-consistency (i.e. optimal performance), while retaining k-robustness. Moreover, they showed that this consistency-robustness tradeoff is essentially optimal within a natural class of deterministic algorithms that respect the local and memoryless structure of DC.

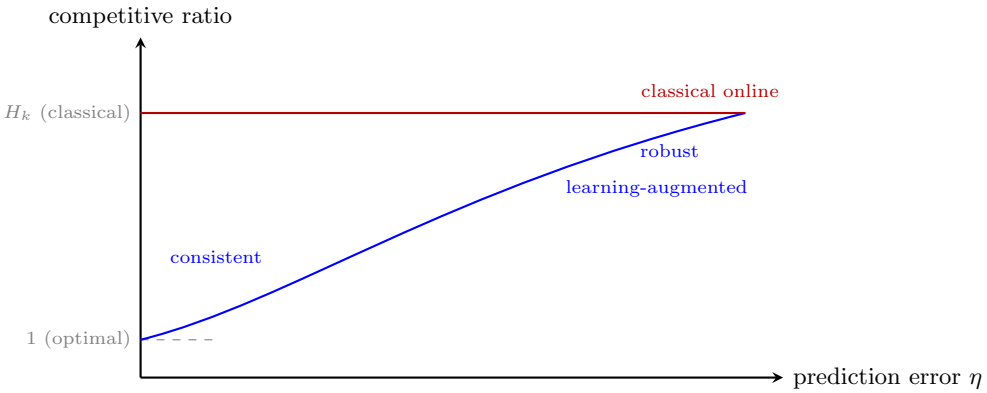

Figure 12. The learning-augmented paradigm. A classical online algorithm has a fixed competitive ratio regardless of input (red). A learning-augmented algorithm (blue) interpolates: it is *consistent* (near-optimal) when the prediction error η is small and *robust* (matching the classical bound) when η is large.

Antoniadis, Coester, Eliáš, Polak, and Simon [ACE+20] extended this paradigm to general metrical task systems, which as we have seen in the form of k-server. Their framework shows that any MTS algorithm can be augmented with predictions while preserving robustness, though the consistency guarantees depend on the structure of the specific problem. The same authors [ACE+23] later showed how to combine multiple predictors for MTS, achieving $O(\ell^2)$-competitiveness against the best dynamic combination of ℓ predictors.

This line of work is growing rapidly. Mitzenmacher and Vassilvitskii [MV20] provide a survey of the broader landscape, and a comprehensive list of results is maintained at the Algorithms with Predictions website [LM23]. Much remains to be done, particularly for the k-server and k-taxi problems on general metrics, where learning-augmented algorithms that are both practically efficient and theoretically well-understood are largely absent.

7.4 Open Problems

We close by collecting some of the major open questions that emerge from this survey.

The oldest and most famous is the *k-server conjecture* itself: is there a deterministic k-competitive algorithm for every metric space? The work function algorithm gives $2k-1$, and it is widely believed to actually be k-competitive, but proving this has resisted all efforts for thirty years. Even the special case of WFA on the Euclidean plane is open.

On the randomized side, the central question left open by the disproof of the randomized conjecture is: *what is the correct competitive ratio on HSTs?* We have $\Omega(\log k)$ from paging and $O(\log^2 k)$ from BCLLM [BCL+18]. Closing this gap is perhaps the single most important open problem in the area. A related question is whether a competitive ratio depending *only* on k (and not on the number of points n or the aspect ratio Δ) is achievable on all metric spaces. Lee's fusible HST approach [Lee18] would give $O(\log^6 k)$, but the proof remains incomplete.

For k-taxi, the gap is even wider, with $\Omega(\log^2 k)$ versus $O(\log^3 \Delta \cdot \log^2(nk\Delta))$. Removing the dependence on Δ and n, or proving that such dependence is necessary, would be a significant advance.

For weighted k-server, the situation is arguably the most mysterious. The exponential lower bound on the randomized competitive ratio suggests that this problem is fundamentally harder than its unweighted cousin, but as far as I am aware, we do not have a clear understanding of why. Understanding the boundary between tractable and intractable server problems seems like a broad and important direction.

Finally, the learning-augmented framework offers an interesting bridge between theory and practice, but the theory is still in its infancy for server problems beyond the line. Designing learning-augmented k-server algorithms for general metrics, with provable consistency-robustness tradeoffs, is wide open.

We began with Uber and ended with fractal-like metric spaces that no rideshare company will ever encounter. The gap between the theory of online algorithms and the practice of dispatching cars in San

Francisco remains vast. But the landscape has never been more interesting: the randomized conjecture, which guided the field for three decades, is false; the correct answer on HSTs is unknown; and the learning-augmented paradigm offers a way to make worst-case theory relevant to a world where predictions, however imperfect, are cheap and abundant.

References

[AC21] Nikhil Ayyadevara and Ashish Chiplunkar. The randomized competitive ratio of weighted k-server is at least exponential. In *29th Annual European Symposium on Algorithms (ESA)*, volume 204 of *LIPIcs*, pages 9:1–9:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

[ACE+20] Antonios Antoniadis, Christian Coester, Marek Eliáš, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 345–355. PMLR, 2020.

[ACE+23] Antonios Antoniadis, Christian Coester, Marek Eliáš, Adam Polak, and Bertrand Simon. Mixing predictions for online metric algorithms. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pages 969–983. PMLR, 2023.

[Bar96] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–193, 1996.

[BBCS21] Sébastien Bubeck, Niv Buchbinder, Christian Coester, and Mark Sellke. Metrical service systems with transformations. In *Proceedings of the 12th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 185 of *LIPIcs*, pages 49:1–49:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

[BBM06] Yair Bartal, Béla Bollobás, and Manor Mendel. Ramsey-type theorems for metric spaces with applications to online problems. *Journal of Computer and System Sciences*, 72(5):890–921, 2006.

[BBMN15] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph (Seffi) Naor. A polylogarithmic-competitive algorithm for the k-server problem. *Journal of the ACM*, 62(5):40:1–40:49, 2015.

[BCL+18] Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k-server via multiscale entropic regularization. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*, pages 3–16, 2018.

[BCN21] Niv Buchbinder, Christian Coester, and Joseph (Seffi) Naor. Online k-taxi via double coverage and time-reverse primal-dual. In *Integer Programming and Combinatorial Optimization (IPCO)*, volume 12707 of *LNCS*, pages 15–29. Springer, 2021.

[BCR23] Sébastien Bubeck, Christian Coester, and Yuval Rabani. The randomized k-server conjecture is false! In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 581–594. ACM, 2023. Best Paper Award.

[BK04] Yair Bartal and Elias Koutsoupias. On the competitive ratio of the work function algorithm for the k-server problem. *Theoretical Computer Science*, 324(2–3):337–345, 2004. Preliminary version in STACS 2000.

[BKRS92] Avrim Blum, Howard J. Karloff, Yuval Rabani, and Michael E. Saks. A decomposition theorem and bounds for randomized server problems. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 197–207, 1992.

[BLMN05] Yair Bartal, Nathan Linial, Manor Mendel, and Assaf Naor. On metric Ramsey-type phenomena. *Annals of Mathematics*, 162(2):643–709, 2005.

[BMC26] Adithya Bijoy, Ankit Mondal, and Ashish Chiplunkar. Weighted k-server admits an exponentially competitive algorithm. In *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4188–4208. SIAM, 2026.

[Cha07] Shuchi Chawla. Lecture 22: k-server problem (continued); online learning. CS787: Advanced Algorithms, University of Wisconsin–Madison. Scribed by Mayank Maheshwari and Priyananda Shenoy, 2007. `https://pages.cs.wisc.edu/~shuchi/courses/787-F07/scribe-notes/lecture22.pdf`.

[CK19] Christian Coester and Elias Koutsoupias. The online k-taxi problem. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 1136–1147. ACM, 2019.

[CKPV91] Marek Chrobak, Howard Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.

[CL91] Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20:144–148, 1991.

[CL22] Christian Coester and James R. Lee. Pure entropic regularization for metrical task systems. *Theory of Computing*, 18:Paper No. 23, 24 pp., 2022.

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

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

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

[GKP23] Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Efficient algorithms and hardness results for the weighted k-server problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 275 of *LIPIcs*, pages 12:1–12:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

[GKP24] Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Poly-logarithmic competitiveness for the k-taxi problem. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4220–4246. SIAM, 2024.

[KMRS88] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

[Kou09] Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.

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

[KPS18] Ravi Kumar, Manish Purohit, and Zoya Svitkina. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018.

[Lee18] James R. Lee. Fusible HSTs and the randomized k-server conjecture. In *Proceedings of the 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 438–449, 2018. Main result retracted; see erratum at `https://homes.cs.washington.edu/~jrl/papers/kserver-erratum.html`.

[LM23] Alexander Lindermayr and Nicole Megow. ALPS — algorithms with predictions. `https://algorithms-with-predictions.github.io/`, 2023. Comprehensive bibliography maintained online.

[LMS25] Alexander Lindermayr, Nicole Megow, and Bertrand Simon. Boosting double coverage for k-server via imperfect predictions. *Algorithmica*, 2025. Preliminary version at ITCS 2022.

[LV21] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *Journal of the ACM*, 68(4):24:1–24:25, 2021.

[MMS88] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for on-line problems. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 322–333, 1988.

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

[MS91] Lyle A. McGeoch and Daniel Dominic Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

[MV20] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 646–662. Cambridge University Press, 2020.

[Roh20] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1834–1845. SIAM, 2020.

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

[Wei20] Alexander Wei. Better and simpler learning-augmented online caching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 176 of *LIPIcs*, pages 60:1–60:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

Figure 13. I'm tired of being online...