CS369N: Beyond Worst-Case Analysis
Lecture #2: Models of Data in Online Paging*

Tim Roughgarden†

February 19, 2010

1 Online Paging: the Model and Basic Examples

We first discuss some preliminaries from Sleator and Tarjan [16], a famous paper that founded the competitive analysis of online algorithms. The are many satisfying results in [16]; perhaps unfairly, we focus only on the less satisfying ones.[1]

The setup is this:

- There is a slow memory with N pages.
- There is a fast memory (a *cache*) that can only hold $k < N$ of the pages at a time.
- Page requests arrive ”online”, one request per time step.
- If the page request p_t at time t is already in the cache, zero cost is incurred.
- If p_t is not in the cache, it needs to be brought in; if the cache is full, one of its k pages must be evicted (without knowing what the future requests will be). One unit of cost is incurred in this case.[2]

In last lecture's notation, we have thus defined cost(A, z) as the number of "page faults" (a.k.a. "cache misses") that A incurs on the page request sequence z.

We now come to a standard and important definition: the *competitive ratio* of algorithm A is its worst-case performance relative to an optimal *offline* algorithm OPT, which has full

*©2009–2010, Tim Roughgarden.

†Department of Computer Science, Stanford University, 462 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: `tim@cs.stanford.edu`.

[1] A good general reference for this section and the next is [4, Chapter 3].

[2] A more general model allows arbitrary changes to the cache at every time step, whether or not there is a hit or miss, with the cost incurred equal to the number of changes. We will focus on the stated model, which corresponds to "demand paging" algorithms.

knowledge of the page sequence z up front:

$$\max_z \frac{\text{cost}(A, z)}{\text{cost}(OPT, z)}.$$

One traditionally deems an online algorithm A "better than" another one B if and only if it has a competitive ratio that is closer to 1.[3]

We can interpret the competitive ratio as a form of instance optimality, in the sense of the last lecture: if an online algorithm A has a competitive ratio of α, then it is instance optimal with optimality ratio α — that is, it has cost at most α times that of every (online or offline) algorithm on every input. Thus it is no surprise that small competitive ratios are relatively rare: it translates to instance optimality with the additional severe restriction that the designed algorithm A is online.

Recall that the optimal offline algorithm OPT for minimizing the number of page faults on a request sequence is a natural greedy algorithm.

Theorem 1.1 (Belady's Theorem [3]) *The* Furthest-in-the-Future *algorithm, which always evicts (on a cache miss) the page that will be requested furthest in the future, always minimizes the number of page faults.*

The proof is a standard (but slightly annoying) greedy exchange argument; see e.g. [13, Section 4.3].

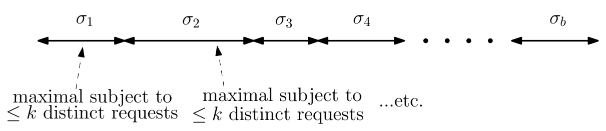
Example 1.2 (Least Recent Used (LRU)) In every systems course, when discussing paging or caching, one learns about the *Least Recently Used (LRU)* algorithm. On a cache miss, the algorithm evicts the page whose most recent request is as far back in the past as possible. The idea is to attempt an online simulation of the Furthest-in-the-Future algorithm, using the past as a predictor for the future. Many empirical studies show that LRU performs very well — not much worse than the offline optimal algorithm, and noticeably better than other obvious online algorithms like FIFO. The usual explanation one learns is: "real data exhibits locality — that is, recent requests are likely to be requested again soon — and LRU automatically adapts to and exploits this locality."

Thus in some sense we already know the "correct" algorithm — or at least a near-optimal one — in the form of LRU. But if LRU is the answer, what is the question? In this lecture we adopt the approach of a natural scientist rather than an engineer: we take excellent empirical performance of LRU as a real-world phenomenon, and we seek an accurate and transparent theoretical explanation for it.

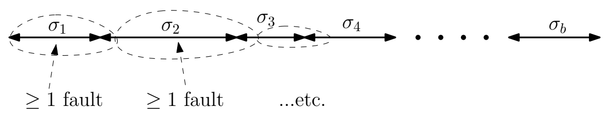
2 Some Basic Worst-Case Upper and Lower Bounds

So what does (worst-case) competitive analysis have to say about the performance of the LRU algorithm versus that of other online algorithms? We begin with an upper bound,

[3]A detail: usually (but not always) one ignores additive terms in the competitive ratio. In other words, we think about sequences of inputs z such that $\text{cost}(OPT, z) \to \infty$.


σ_1
σ_2
σ_3
σ_4
σ_b
maximal subject to $\leq k$ distinct requests
maximal subject to $\leq k$ distinct requests
...etc.

(a) Blocks of a Request Sequence



(b) Page Fault Lower Bound

Figure 1: Proof of Theorem 2.1. In (a), the blocks of a page request sequence; the LRU algorithm incurs at most k page faults in each. In (b), the optimal offline algorithm incurs at least one page fault in each "shifted block".

which is a special case of one in [16].

Theorem 2.1 (Upper Bound for LRU [16]) *The competitive ratio of the LRU algorithm for online paging is at most k, the size of the cache.*

Proof: Consider an arbitrary request sequence σ. We can break σ in *blocks* $\sigma_1, \sigma_2, \ldots, \sigma_b$, where σ_1 is the maximal prefix of σ in which only k distinct pages are requested; the block σ_2 starts immediately after and is maximal subject to only k distinct pages being requested (ignoring what was requested in σ_1); and so on.

The first important point is that LRU faults at most k times within a single block — at most once per page requested in the block. The reason is that once a page is brought into the cache, it won't be evicted until k other distinct pages get referenced, which can't happen until the following block. Thus LRU incurs at most bk page faults, where b is the number of blocks. See Figure 1(a).

Second, we claim that an optimal offline algorithm must incur at least $b-1$ page faults. To see this, consider the first block plus the first request of the second block. Since σ_1 is maximal, this represents requests for $k+1$ distinct page requests, and no algorithm can serve them all without a page fault. Similarly, suppose the first request of σ_2 is the page p. After an algorithm serves the request for p, the cache can only contain $k-1$ pages other than p. But by maximality of σ_2, the rest of σ_2 and the first request of σ_3 contain requests for k distinct pages other than p; these cannot all be served without incurring another page fault. And so on, resulting in at least $b-1$ cache misses. See Figure 1(b). The theorem follows. ■

No deterministic online paging algorithm has a better competitive ratio (again, a special case of a result in [16]).

Theorem 2.2 (Lower Bound for Deterministic Algorithms [16]) *Every deterministic paging algorithm has competitive ratio at least k.*

Proof: Take $N = k+1$ and fix a deterministic online algorithm A. Since there is always some page missing from A's cache, one can define inductively a sequence σ so that A faults on every single page request. The Furthest-in-the-Future (FIF) algorithm, whenever it incurs a page fault on the sequence σ, has k candidates for eviction, and one of these will not be among the next $k-1$ requests. Thus the FIF algorithm follows every cache miss with at least $k-1$ cache hits. The theorem follows. ■

3 Discussion of Theorems 2.1 and 2.2

Here's a question:

Question 3.1 Are Theorems 2.1 and 2.2 interesting results?

The answer depends on our opinion about a different question.

Question 3.2 What is the point of analyzing paging algorithms and LRU in particular using competitive analysis?

An ambitious answer to the second question is:

> *to explain theoretically the observed empirical behavior of LRU and other paging algorithms.*

For this strong goal, Theorems 2.1 and 2.2 are clear failures. Quantitatively, the competitive ratio of k is laughably huge (k is pretty big in most systems), far worse than the small percentage error one observes on "real data". Qualitatively, these results suggest that the relative performance[4] of every deterministic online algorithm (LRU in particular) degrades with the cache size. Not only is this refuted by empirical results, but it suggests that devoting resources (e.g., transistors on a microchip) to bigger caches is pointless and that they would be better spent anywhere else. This is obviously bad advice.

But if we recall how stringent the definition of a competitive ratio is — again, even more so than instance optimality — we shouldn't be surprised to see such high competitive ratios. *And this does not necessarily mean that the theoretical results are meaningless or useless.* A much more reasonable answer to Question 3.2 is:

> *to give an accurate ordinal ranking on online algorithms and thereby provide good advice about which one to use.*

For this weaker goal, Theorems 2.1 and 2.2 appear to be successful: LRU has the smallest-possible competitive ratio among the class of all deterministic online paging algorithms. (While a silly algorithm like LIFO — "last-in, first out" — does not, as can be seen via a sequence that alternates requests between two different pages.)

But the plot thickens if we consider the Flush-When-Full (FWF) algorithm.

[4]The proof of Theorem 2.2 doesn't say that absolute performance of an algorithm degrades with the cache size. It shows that for every cache size one might fault on every request; and by contrast, the optimal algorithm's performance improves as the cache size increases.

Example 3.3 (Flush-When-Full (FWF)) The Flush-When-Full algorithm works as follows. In addition to handling page requests, it keeps track of the blocks $\sigma_1, \sigma_2, \ldots$ of the request sequence (recall the proof of Theorem 2.1). Note that this is easy to do online. Then, every time a new block σ_i begins, the FWF algorithm evicts *its entire cache*.[5] Note that the algorithm will not need to evict anything until the block ends (at which point it again evicts everything).

The way to think about the FWF algorithm is that it assumes that the two sets of k distinct pages referenced in two consecutive blocks will have no overlap. In this case, all of the cache evictions would have been eventually necessary anyway, and no harm is done. When there is overlap between consecutive blocks, however, the FWF algorithm will incur more page faults than LRU (which won't fault on the pages in the overlap).

Nonetheless, *the competitive ratio of the FWF algorithm is also k.* The reason is that the proof of Theorem 2.1 still applies verbatim — in particular, the FWF algorithm faults exactly k times per block. Indeed, this algorithm could only have been discovered by worst-case analysts asking the question: what are the minimal properties required for the proof of Theorem 2.1 to work?

4 Online Paging: Beyond Competitive Analysis

So is the FWF algorithm just as good as LRU? Obviously not. Thus competitive analysis, in the case of online paging, is at best partially successful in achieving even the weaker goal stated in the previous section. In particular, it fails to meaningfully differentiate between some "obviously better" and "obviously worse" online paging algorithms. This issue was noticed immediately, and it led to a fair amount of introspection about competitive analysis and spawned a sizable literature on ways to rectify its drawbacks. The rest of this lecture surveys most of the "first-generation" approaches (mostly from the early 1990s). Good overviews of the "second-generation" results (from the 21st century) are in [1, 9]; some of these are explored on the homeworks.

There have been three genres of results that attempt to glean more meaningful theoretical results about online paging.

1. Randomized algorithms. There are some cool ideas and algorithms for randomized online paging algorithms, and the competitive ratios are much better: $O(\log k)$ is achievable and also the best possible. See [4, Chapter 4] for a survey. We won't discuss these randomized algorithms, as most of them are rather different from the standard ones used "in practice"; and because the online paging problem seems solvable empirically (e.g., by LRU) without resorting to randomization.[6]

[5]Strictly speaking this is outside our demand paging model, but you get the idea.

[6]A caveat: worst-case analysis of randomized algorithms can still explain why deterministic algorithms perform well on "real instances" of a problem. For example, the proof that randomized Quicksort is fast (with high probability) on every input also shows that deterministic Quicksort is fast on almost every input. Thus the worst-case analysis of randomized QuickSort provides an excellent theoretical explanation of the speed

2. Comparing two (deterministic) algorithms using an alternative to competitive analysis — in particular, in a way that eschews comparison with an offline algorithm. Early attempts along these lines were ad hoc and failed to gain traction, but there has been some interesting work in this direction in the past few years (touched on in the homeworks). For example, analogs of the "order-oblivious instance-optimality" notion that we saw in Lecture #1 — there, in the context of running-time analyses of computational geometry algorithms — have also been explored (several years earlier) for paging and other online problems [6, 12].

3. Introducing a *model of data*. This lecture will focus on this direction. Our discussion also serves as a segue into the next five lectures, where integrating models of data with the spirit of worst-case analysis will be a recurrent theme.

5 Models of Data: A Preamble

5.1 Motivation

Why introduce a model of data? Recall the perspective we're adopting in this lecture — excellent empirical performance of LRU is an observable phenomenon, and we seek a good theoretical explanation. Think about what the systems gospel tell us: the LRU algorithm is exceptionally good in practice (and better than e.g. "first-in first-out" (FIFO)) *only because of a property of "real data"* — namely, locality of reference. If we accept this assertion, then we *cannot expect* an analysis framework to strongly advocate the LRU algorithm without a non-trivial data model.

5.2 A Digression

Warning 1 In the theoretical design and analysis of algorithms, there is no uncontroversial non-trivial model of data.

Note that "uncontroversial" is a sociological statement, not a mathematical one. *Every* model has drawbacks, and the key is to understand whether or not a model's weaknesses are so severe that they preclude a useful interpretation of results proved in the model. Certainly worst-case analysis — the "Murphy's Law" data model — has its drawbacks, but this analysis framework is not controversial among theoretical computer scientists. Its weaknesses are well understood and clearly explained in every algorithms or complexity textbook; researchers in the field agree (for the most part) on how to interpret results proved in the model; and despite its problems, it is widely accepted on the grounds that is has led to lots of useful results and also a beautiful theory.[7]

of deterministic QuickSort on "real data", for essentially any reasonable definition of "real data" (modulo the usual "already sorted" exception). It is unclear whether or not the randomized online paging algorithms with good competitive ratios are similar enough to algorithms like LRU to suggest a similar interpretation.

[7]Worst-case analysis certainly is controversial in other communities, where the many successes of the framework are not so well known.

As of yet, no non-trivial data model has achieved anywhere near the level of consensus acceptance as worst-case analysis. (Smoothed analysis, discussed in Lecture #7, might be the closest candidate.) Hopefully this will change soon — it seems necessary to enable useful progress on the algorithmic problems that resist understanding via worst-case analysis.

5.3 On Average-Case Analysis

The polar opposite of worst-case analysis is the classical approach of positing a specific distribution D over inputs z. One can interpret such a distribution literally, as if nature is really flipping coins according to D to generate the input z (a "frequentist" interpretation). Alternatively, the distribution D could numerically encode the "beliefs" of the algorithm designer about which inputs are more important than others (a "Bayesian" interpretation).

Accepting the existence of a distribution D, one can summarize the performance of an algorithm via its expectation $\mathbf{E}_{z \sim D}[\mathrm{cost}(A, z)]$.[8] Such a one-dimensional summary automatically gives a full ordinal ranking on algorithms, and unequivocally singles out the "optimal algorithm(s)". This stands in contrast to last lecture's view that an algorithm's performance is really a *vector*, indexed by inputs, and that two different algorithms typically have incomparable performance vectors. A distribution D over inputs is equivalent to a consistent way of trading off between the performance of an algorithm on different inputs.

We will avoid such "average-case analyses" as much as possible in this course. This bias is motivated by the following two hypotheses.

1. Positing a distribution D assumes implausibly rich knowledge about statistical properties of real data or about the appropriate trade-offs between different inputs.

2. Proposing an overly specific data model introduces the danger of "overfitting" a non-robust solution to the model, which is often uncertain anyways.

Of course, in some applications average-case analysis is perfectly appropriate — e.g., when reams of data are available and the input distribution is invariant over time. This course will focus on ways to formulate the conviction that "real data" has at least some structure and is not worst-case, but cannot be plausibly described by something as specific as an input distribution (online paging is a good example). We will pursue "weak" models of data that encourage robust and simple algorithmic solutions (like the LRU algorithm). In optimization terms, we want to *maximize the robustness of our data model, subject to the constraint of meaningful results.*

6 Models of Data in Online Paging

We now survey three early and different attempts to capture theoretically aspects of "real-world inputs" for online paging. Chapter 5 of [4] also surveys this material.

[8]Obviously, a function other than the expectation (e.g., to reflect risk) could also be used here.

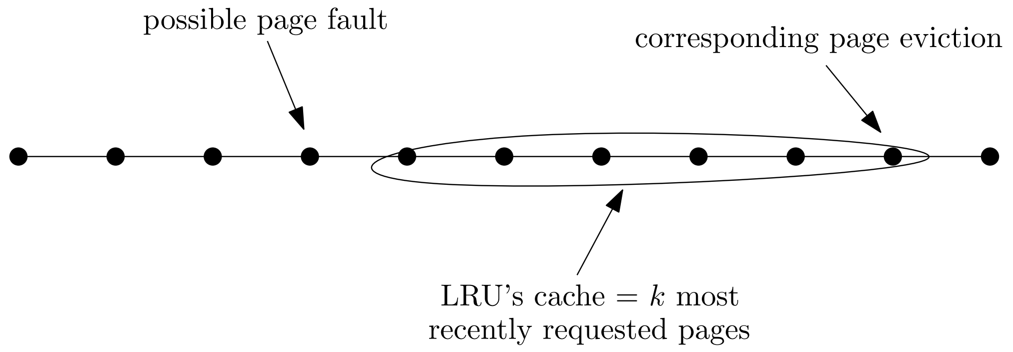

possible page fault
corresponding page eviction
LRU's cache = k most recently requested pages

Figure 2: Example 6.1. The LRU algorithm is optimal on a path access graph.

6.1 Access Graphs

Borodin et al. [5] proposed *access graphs* as a natural way to capture locality of reference in page request sequences. Karlin et al. [11] developed the model and results further (see [4, Chapter 5] for further references). Formally, an *access graph* is just an undirected graph where the nodes correspond to the set N of all pages. A request sequence σ is *legal* for an access graph if and only if every request p_t is a neighbor (in G) of the previous one p_{t-1}. This model is clearly simplistic but is reasonably well-motivated: traversals of a data structure (like a B-tree, say) could generate requests conforming to an access graph (that mirrors the data structure); and requests for program code could conform to an access graph related to information flow in the program.

The *competitive ratio* of an algorithm A for an access graph G is then

$$\max_{\text{legal } z} \frac{\text{cost}(A, z)}{\text{cost}(OPT, z)}.$$

Thus G defines a "promise problem", in that the algorithm can behave arbitrarily on inputs z that do no conform to the "promise" of being legal for G.

The standard online paging problem corresponds to the case where G is the complete graph (including self-loops). Thus to learn anything new we need to consider restricted types of graphs.

Example 6.1 (LRU on a Path Graph) We claim that if G is a path graph (so every request is either to the left or the right of the previous one), then the LRU algorithm is optimal (with competitive ratio 1). To see this, first note that, by induction, the cache of the LRU algorithm is the subpath of the more recently requested k pages. Thus, a page fault occurs at either the left or the right boundary of the current cache (see Figure 2). The LRU algorithm evicts the page at the opposite boundary. Among the pages in the cache, this is necessarily the one that will be requested furthest in the future (everything else must be requested first). Thus the LRU and (optimal) Furthest-in-the-Future algorithms coincide.

By contrast, the Flush-When-Full and FIFO algorithms still have competitive ratio $\Omega(k)$ when G is a path (exercise, see Homework #1). This toy example is evidence that the access graph model could work well: a path access graph is an extreme form of locality, LRU uses it perfectly, while FIFO's worst-case performance remains poor. The following theorem of Chrobak and Noga (which we won't prove) asserts that LRU's superiority over FIFO is not an artifact of path graphs, and holds for *every* model of locality (i.e., every access graph).

Theorem 6.2 (LRU is Better than FIFO [7]) *For every access graph G, the competitive ratio of LRU for G is at most that of FIFO for G (and it is strictly better for many graphs G).*

It would be nice to extend this "graph-by-graph" guarantee[9] to a class of algorithms that generalizes FIFO. This remains an open question. One might hope that LRU is as good as *every* other deterministic online algorithm for every access graph. This is not true, however — there are graphs for which there are deterministic online algorithms with much smaller competitive ratio than LRU.

Example 6.3 (LRU on a Cycle) Suppose G is a cycle of $k+1$ nodes and consider the cyclic (i.e., periodic) legal request sequence. The proof of Theorem 2.2 shows that the LRU algorithm has competitive ratio k for G.

On the other hand, there is a deterministic online algorithm that is tailored for the cycle access graph that has competitive ratio only $\approx \log_2 k$ (see Homework #1). A natural question is whether or not there is a "graph-independent" online algorithm that dominates LRU. This depends on how one defines "graph-independent". Fiat and Mendel [10] give an affirmative answer when graph-independent is defined to mean that every page eviction depends only on the sequence of requests so far, and not on any additional information (like a priori knowledge of G).

6.2 Markov Paging

We next discuss a natural probabilistic variant of access graphs, called *Markov paging* by Karlin et al. [11]. Here, we assume that the request sequence σ is a sample from a Markov chain, where the states correspond to the pages N. In other words, given that the last request was p_{t-1}, there is a probability distribution that governs what the next page p_t will be. Following [11], we assume that the Markov chain is known a priori to the algorithm designer. While such a chain is in principle learnable from a sufficiently long prefix of the request sequence, this is not an entirely satisfactory assumption and we will return to it later.

[9]One can draw a rough analogy between this result and last lecture's instance optimality. Recall in instance optimality one thinks of an algorithm's performance as a vector indexed by inputs, and looks for componentwise dominance. Here, components of the performance vector correspond to online paging *problems*, indexed by access graphs. The value of a component is the (worst-case) competitive ratio on sequences legal for that access graph, rather than the algorithm's performance on a single input.

We said in Section 5.3 that we'll try to avoid average-case analyses. While mostly true, that's exactly what we'll do in this section. So the goal is to identify an online algorithm that minimizes $\mathbf{E}[\mathrm{cost}(A, \sigma)]$, where σ is a sample (of a fixed length) from the (known) Markov chain. Recall that in this case, there is an online algorithm that can, without controversy, be deemed optimal. So what's left to do?

The problem is that an optimal algorithm — which reports the optimal page to evict given the current cache and the requested page — can take exponential space to describe (see Homework #1). The goal in this section is to design a tractable online algorithm with expected cost almost as small as that of the optimal one. By "tractable" we mean with running time polynomial in N, k, and the description of the Markov chain. Note this is not competitive analysis per se: there is no mention of an offline algorithm, and there is no input-by-input comparison of two algorithms (only expected costs are compared).

We discuss an algorithm by Lund et al. [15] that achieves this goal. Very roughly, the idea of the algorithm is to (on a cache miss) evict a page with probability proportional to its likelihood of being requested last. Thus the algorithm is randomized, although it's not clear that this is necessary for the above goal — for example, the optimal online algorithm is deterministic (see Homework #1). The formal definition of the algorithm (i.e., its behavior on a page fault) is given in Figure 3.

1. For each distinct pair of pages p, q in the cache, compute the probability $\alpha(p.q)$ that q will be requested before p (conditioned on the most recently requested page). Define $\alpha(p, p) = 0$ for all p. [This step can be implemented in polynomial time — see Homework #1.]

2. Compute a distribution D on the pages in the cache with the following property: for every fixed page p in the cache and for a random page q drawn from D, $\mathbf{E}_{q \sim D}[\alpha(p, q)] \leq \frac{1}{2}$. [Lemma 6.4 below shows that this can be done.]

3. Evict a random page q drawn from the distribution D.

Figure 3: The LPR algorithm.

Intuitively, the random page q chosen from D in the LPR algorithm is "likely to be last" according to pairwise comparisons between pages (i.e., the α's). Why not do something conceptually simpler along the same lines? Presumably, this is the simplest solution discovered so far that can be implemented in polynomial time and whose expected cost is analyzable.

Lemma 6.4 ([15]) *The distribution D in Step 2 of the LPR algorithm exists and can be computed in polynomial time.*

Proof: We use the Minimax Theorem for two-player constant-sum games. Recall that this says that in games of pure competition (a gain by one player involves an equal loss by the other), if mixed (randomized) strategies are allowed, then *the order of play doesn't matter.*

More formally, let M denote the matrix of the $\alpha(p,q)$'s. Imagine that the row (column) player picks a probability distribution x (y) over pages in the cache to maximize (minimize) $x^T My$, the expected payoff from the column player to the row player (assuming that the two players randomized independently). The Minimax Theorem asserts that:

$$\max_x \left(\min_y x^T My \right) = \min_y \left(\max_x x^T My \right),$$

where x, y range over all probability distributions. The LHS (RHS) is the utility (negative utility) the row (column) player can guarantee itself if it is forced to go first, assuming optimal subsequent play by the other player. It is clear that the LHS is at most the RHS (announcing your strategy up front can't help you) but in fact they are equal. This fact is equivalent to strong duality in linear programming, which you should know from a different course (see e.g. [8, Chapters 5,15]).

The lemma asserts that the RHS is at most 1/2 (note that in the inner maximum, by linearity it is enough to check only the k deterministic x's). So we can instead prove that the LHS is most 1/2. So fix a choice of x, we just need to exhibit a suitable y. Take $y = x$. By symmetry, the outcomes (p,q) and (q,p) are equally likely. Since $\alpha(p,q) + \alpha(q,p) = 1$ for distinct p,q (and equals 0 if $p = q$), it follows that $x^T My = x^T Mx \leq \frac{1}{2}$, as claimed.

We've shown that the desired distribution y exists. Computing such a distribution can be formulated as a polynomial-size linear program, which can be solved in polynomial time. ■

Lund et al. [15] build on Lemma 6.4 to prove the following guarantee for their algorithm.

Theorem 6.5 ([15]) *For every Markov chain MC, the expected cost of the corresponding algorithm LPR_{MC} (over inputs generated by MC) is at most 4 times that of an optimal online algorithm OPT_{MC} for MC:*

$$\mathbf{E}_{z\sim MC}[cost(LPR_{MC}, z)] \leq \mathbf{E}_{z\sim MC}[cost(OPT_{MC}, z)]. \tag{1}$$

We omit the proof. While it isn't *that* long or difficult (see the paper), it is on the opaque side.

Several open questions suggest themselves. First, it would be nice to have a simple or transparent proof of Theorem 6.5 — or something similar for a different algorithm — building on the intuitive guarantee of Lemma 6.4. Second, the upper bound could be improved. Even just for the LPR algorithm, the best known lower bounds are not much greater than 1. An upper bound of 2 certainly seems possible.

The bigger open question is to prove more robust guarantees via a less specific model of data. A first step would be to assume that the Markov chain is initially unknown. One could possibly preclude trivial learning of the Markov chain by focusing on request sequences that are not arbitrarily long. Second, one could look at more general probabilistic models than Markov chains — and this is done to some extent in [15]. What would really be interesting, though, would be to take a cue from the previous lecture on instance optimality.

Recall this entails proving that the proposed algorithm is (almost) as good as every other algorithm (possibly from a restricted class $\mathcal{C}$) on every single input. The question is: what's a meaningful definition of the class $\mathcal{C}$ of competing algorithms? When the motivation is to generalize a distributional model like Markov paging, there is an excellent candidate: an online algorithm lies in $\mathcal{C}$ if and only if it is *optimal for some Markov chain on the pages N.* (Or if this is too strong, for some Markov chain in a restricted class.) If there were a single algorithm A that is instance optimal with optimality ratio α with respect to this class $\mathcal{C}$, then in particular, for every Markov chain MC, its expected cost would be at most α times that of an optimal online algorithm for MC. (Exercise: make sure you understand the previous statement.) Thus instance optimality would imply, in particular, that the guarantee of Theorem 6.5 holds for the algorithm A *simultaneously* with respect to every possible underlying Markov chain — this would be a version of Theorem 6.5 in which the algorithm on the left-hand side of (1) is fixed while the Markov chain (and hence the algorithm OPT_{MC} on the right-hand side of (1)) varies.[10] Instance optimality is an even stronger guarantee, since it makes sense even when there is no underlying distribution — the instance-optimal algorithm is still always almost as good as every "reasonable" algorithm.

We study this connection between average-case analysis and instance optimality more systematically in Lecture #9.

6.3 Diffuse Adversaries

Our final model of data in online paging is the notion of a *diffuse adversary*. The idea comes from Koutsoupias and Papadimitriou [14].[11] The goal is essentially the same as the corollary of instance optimality we discussed last section. Namely, let $\mathcal{D}$ denote a set of distributions of sequences on a set N of pages. The goal is to design a single algorithm A (like LRU) which is simultaneously near-optimal for every distribution in $\mathcal{D}$:

$$\mathbf{E}_{z\sim D}[\mathrm{cost}(A,z)] \leq \alpha \cdot \mathbf{E}_{z\sim D}[\mathrm{cost}(OPT_D,z)]$$

for every $D \in \mathcal{D}$, where OPT_D denote the optimal online algorithm for the distribution D, and α is hopefully small (like $O(1)$).

This is a very nice idea. Of course, the question is then: what is a tractable yet well-motivated choice for $\mathcal{D}$? If $\mathcal{D}$ contains all of the point masses (i.e., some input occurs with probability 1) then we recover the standard definition of competitive analysis (as you should check). If $\mathcal{D}$ is a singleton, then the analysis framework reduces to average-case analysis. Intuitively, we would like to choose $\mathcal{D}$ as large as possible — committing as little as possible to what we know about the data — subject to getting useful information about the performance of different paging algorithms.

Koutsoupias and Papadimitriou [14] take $\mathcal{D}$ to be the *diffuse adversaries*, defined as the distributions D with the following property: for every page p and time t, no matter

[10]For sufficiently long input sequences, the LPR algorithm itself can be adapted to satisfy this guarantee by prepending a learning phase. So this corollary of instance optimality would be especially interesting for reasonable-length sequences.

[11]As does the title of these lectures.

what the history is before time t, the probability that the tth request is for p is at most a parameter ϵ. These distributions are incomparable to those generated by Markov chains. In Markov paging, the probability of a given page being requested at a given time can be arbitrarily small or large; but this probability can only depend on the previously requested page (not on the whole history so far, as with diffuse adversaries).

The motivation for diffuse adversaries is unclear, other than the intuition that they should be "sufficiently far from the point masses". In particular, would "real data" correspond to a diffuse adversary? A related drawback is the difficulty of proving that LRU is superior to (say) FIFO for diffuse adversaries; currently, this is known only for certain parameter values [14, 17]. This is perhaps unsurprising, given that diffuse adversaries need not exhibit locality of reference, which is the accepted explanation for LRU's good empirical performance. Exploring simultaneous approximation with respect to other classes $\mathcal{D}$ of input distributions is a largely open (and interesting) research question; see [2] for some partial results.

References

[1] S. Angelopoulos and P. Schweitzer. Paging and list update under bijective analysis. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1136–1145, 2009.

[2] L. Becchetti. Modeling locality: A probabilistic analysis of LRU and FWF. In *Proceedings of the 12th Annual European Symposium on Algorithms (ESA)*, volume 3221 of *Lecture Notes in Computer Science*, pages 98–109, 2004.

[3] L. A. Belady. A study of replacement algorithms for a virtual storage computer. *IBM Systems Journal*, 5(2):78–101, 1967.

[4] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[5] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50(2):244–258, 1995. Preliminary version in *STOC '91*.

[6] J. Boyar and L. M. Favrholdt. The relative worst order ratio for online algorithms. *ACM Transactions on Algorithms*, 3(2), 2007. Article 22. Preliminary version in *CIAC '03*.

[7] M. Chrobak and J. Noga. LRU is better than FIFO. *Algorithmica*, 23(2):180–185, 1999. Preliminary version in *SODA '98*.

[8] V. Chvátal. *Linear Programming*. Freeman, 1983.

[9] R. Dorrigiv and A. López-Ortiz. A survey of performance measures for on-line algorithms. *SIGACT News*, 36(3):67–81, 2005.

[10] A. Fiat and M. Mendel. Truly online paging with locality of reference. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 326–335, 1997.

[11] A. R. Karlin, S. J. Phillips, and P. Raghavan. Markov paging. *SIAM Journal on Computing*, 30(3):906–922, 2000. Preliminary version in *FOCS '92*.

[12] C. Kenyon. Best-fit bin-packing with random order. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 359–364, 1996.

[13] J. Kleinberg and É. Tardos. *Algorithm Design*. Addison-Wesley, 2005.

[14] E. Koutsoupias and C. H. Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30(1):300–317, 2000. Preliminary version in *FOCS '94*.

[15] C. Lund, S. Phillips, and N. Reingold. Paging against a distribution and IP networking. *Journal of Computer and System Sciences*, 58(1):222–232, 1999. Preliminary version in *FOCS '94*.

[16] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commincations of the ACM*, 28(2):202–208, 1985. Preliminary version in *STOC '84*.

[17] N. E. Young. On-line paging against adversarially biased random inputs. *Journal of Algorithms*, 37(1):218–235, 2000. Preliminary version in *SODA '98*.