CSC2421 Topics in Algorithms: Online and Other Myopic Algorithms Fall 2019

Allan Borodin

September 18, 2019

Week 2

Todays agenda:

- Update some comments from last week regarding bin packing.
- Discuss two related online maximization problems, time series search and one-way trading.
- Introduce randomization for online algorithms
- 1. Type of adversaries and their relation.
  2. A contrived extreme example of how much randomization can help: the bit guessing game
  3. A more natural extreme example of how much randomization can help: the proportional profit knasack problem
  4. Brief discussion of the why and how of de-randomization
- The Yao minimax principle
- Paging: the deterministic vs the randomized competitive ratio

What is now known about online bin packing

- The (asymptotic) competitive ratio for FirstFit and BestFit is 1.7. The initial result by Garey, Graham and Ullman [1972] showed that if ALG is FirstFit or BestFit then for every $\epsilon > 0$, $ALG \leq (1.7 + \epsilon)OPT$ for $OPT \geq \frac{51}{10\epsilon} + 3$. They also pointed out that for many values of OPT, it was actually the case that FirstFIt and BestFit satsfiy $ALG \leq (1.7)OPT$. The proof in our text is their result.
- After a series of results showing $FF \leq (1.7)OPT + c$ for better additibe constants c, Dósa and Sgall in 2013 (resp. 2014) showed that 1.7 was a strict competitive ratio for FirstFit (resp., BestFit).
- In 1980 Yao provided an online algorithm with competitive ratio $\frac{5}{3}$ and gave the first negative result in competitive analysis showing that no deterministic online algorithm has competitive ratio better than $\frac{3}{2}$.
- Currently the best know upper (resp. lower bound) for the bin packing competitive ratio are 1.57829 (resp 1.5014) due to Balogh et al [2018] (resp. Balogh, Békési and Galambos [2012]. And just this year, Dosa et al [2019] showed that $\frac{5}{3}$ is the best *strict* competitive ratio for bin packing.

The time-series search problem

We now consider our first maximization problem, the time-series search problem. Think of selling a US house when moving from the US to Canada. Every day there is a different exchange rate. Suppose we have n days to convert the US funds obtained from the sale. Suppose the exchange rates are $p_1, \ldots, p_n$. That is, one \$ US = p_i\$ Canadian on day i. We would like to sell on a day i so as to maximize the value p_i. Suppose we also know that $L \leq p_i \leq U$. (Note: such bounds might be dictated by the government or might be assumed from historical data.

If $\phi = \frac{U}{L}$ then any algorithm for picking the day to convert will achieve competitive ratio at most ϕ.

Reservation price policy (RPP) for converting given U and L

There is an online algorithm that achieves strict competitive ratio $\sqrt{\phi}$. Namely, let $p^* = \sqrt{UL}$. Then convert on the first day j (if any) such that $p_j \geq p^*$. If there is no such day, then convert on the last day. Furthermore $\sqrt{\phi}$ is a tight bound given U and L whereas if only ϕ is given, then ϕ is a tight bound.

The One-Way trading algorithm

The following problem generalizes the time series problem. We again have exchange rates $p_1, \ldots, p_n$ and want to convert US \$ to Canadian \$. But now we have the option to trade gradually. That is, on every day, we can choose a fraction f_i of the funds at that days exchange rate. The objective is to maximize $\sum_j f_j p_j$ subject to $\sum_j f_j = 1$. The time series problem forces $f_j \in \{0, 1\}$.

There is an algorithm that perhaps suprisingly obtains a competitive ratio $c(\phi) \log \phi$ where $c(\phi) \to 1$ as $\phi \to 1$. The ratio $\log \phi$ is asymptotically tight).

Assume $U/L = 2^k$ for some k.

More precise results can be found in the El-Yaniv, et al Algorithmica 2001 paper. In particular, they show that with knowledge of ϕ alone, one can also get a competitive ratio "arbitrarily close" to $\log \phi$.

The competitive algorithm for the one-way trading problem

Algorithm 6 The MIXTURE OF RPPS

procedure RESERVATION PRICE

  ▷ U, L, and $\phi = U/L = 2^k$ are known in advance

  $i^* \leftarrow -1$

  for $j \leftarrow 1$ to n **do**

    $i \leftarrow \max\{i \mid L2^i \leq p_j\}$

    if $i = k$ **then**

      $i \leftarrow k - 1$

    if $i > i^*$ **then**

      Trade fraction $(i - i^*)/k$ of savings on day j

      $i^* \leftarrow i$

  Trade all remaining savings on day n

Comments on time-series and one-way trading

- We will later discuss *the secretary problem*. This problem introduced the random order input model (ROM) where an adversary creates a nemesis set S of input items. The sequence of inputs is a random permutation of the items in S.
- In the secretary problem, the input is a sequence of items (e.g., secretaries or candidates for any position) where each input item is described by a value v_j. The secretary problem assumes the ROM model and the goal is to maximize the probability of choosing the best item. When the items can have arbitrary values, this is equivalent to maximizng the expected value of the solution. In terms of the objective and as online problems, the secretary problem and time series search are the same problem.
- Why did we refer to the one-way trading algorithm as mixture of RPPs? It turns out that the Mixture of RPPs is the de-randomization of an even simpler randomized algorithm.

Competitive ratio of maximization problems

In the two maximization examples just presented, we expressed the competitive ratio ρ so that $\rho \geq 1$. This is consistent with the following definition of the competitive ratio for a maximization problem:

$$\rho(ALG) = liminf_{OPT(\mathcal{I}) \to \infty} \frac{OPT(\mathcal{I})}{ALG(\mathcal{I})}$$

Note that the ratio here is inverted from the ratio in the definition for minimization problems as we know that OPT has to be at least as good as ALG maximization. We could instead maintain the following definition as in the minimization problems, namely:

$$\rho(ALG) = liminf_{OPT(\mathcal{I}) \to \infty} \frac{ALG(\mathcal{I})}{OPT(\mathcal{I})}$$

Using this defintion, competitive ratios will always satisfy $\rho \leq 1$.
There is no clear convention but I will try to use $\rho \geq 1$ when the ρ is is a function of the input $\mathcal{I}$ and use $\rho \leq 1$ when ρ is an absolute constant.

What have we been doing and where are we going?

We have basically covered Chapters 1 and 2 except for the line search problem and paging. Paging is an important topic and will be discussed later in this lectture and in aa more unified manner in subsequent lectures. Next is Chapter 3 where we begin a discussion of randomized online algorithms. The power of randomization is one of the most fundamental questions in complexity theory. The central open question in this regard is whether or not $P = BPP$ where BPP is a randomized version of P that allows for 2-sided errors.

Even if it turns out that say $P = BPP$, which some prominent complexity theorists conjecture, it is still possible that randomization provides better complexity bounds and/or may allow conceptually simpler algorithms for various problems.
For online algorithms, our goal is to see the extent to which randomization can improve bounds the competitiveness that can be achieved for various problems or when it may permit a conceptually simpler algorithm. For online algorithms, the randomneess is used in making a decision for each online input item.

Randomized algorithms

We think of randomized algorithms having the ability to access a distribution and instantiate a random variable whenever we want to make a randomized decision.

It is sufficient to just have access to *random bits* since we can approximate all the common distributions to any degree of accuracy using a fixed number of random bits. For example, we might want to say "choose a random real number in $[0, 1]$". This should be apparent as to how to do this.

We will therefore consider randomized algorithms (either online or offline) as having access to an infinite tape of random bits. For algorithms terminiating in some bounded number of steps, we will only use a bounded initial substring of the bits from the infinite tape.

Although we traditionally think of using randomness as we go along, we can alternatively view randomized algorithms (either online or offline) as a mixture of deterministic algorithms. Once we fix the random bits, we have a deterministic algorithm. So for a fixed n, we will have a finite number of deterministic algorithms each deriving from an initial instantiation of the bits.

Suppose there are at most 2^t computation paths (one for each setting of say t bits). A specific computation path will use some $t' \leq t$ of these bits. We can think of this being filled out by all the to $2^{t-t'}$ choices of bits meaning that this path is chosen with higher probability.

For the purpose of proving negative (i.e., inapproximation) results, the view of randomized algorithms as a mixture allows us to use the Yao pinciple (to be discussed).

Types of adversaries

While for deterministic online algorithms, there is only one type of adversary, when discussing randomized online algorithms, there are three types of adversaries.

- An *oblivious adversary*. This is the commonly assumed adversary and the one we assume. The adversary is oblivious in the sense that it does not see the random bits and decisions of the algorithm (and hence gives the algorithm a "fighting chance" at improved performace. That is, the adversary just provides a sequence $\{x_1, \ldots, x_n\}$ of inputs. The performance of the algorithm is now a random variable depending on the randomness used in the algorithms decisions.

  The competitive ratio (for a minimization algorithm ALG) in this setting is defined as :

  $$\rho(ALG) = \lim sup_{OPT(x_1, \ldots x_n) \to \infty} \frac{\mathbb{E}_{D_1, \ldots, D_n}(ALG(x_1, \ldots, x_n, D_1, \ldots, D_n))}{OPT(x_1, \ldots, x_n)}$$

  Note that the denominator is not a random variable. Why?

Adaptive adversaries

There are two types of adaptive adversaries.

- *Adaptive offline adversary*. This is the strongest adversary and hence makes it the hardest for the algorithm to exploit randomness. In this setting, after the algorithm creates each x_i, it sees the decision of the algorithm before creating x_{i+1}. Then after the entire sequence $x_1, \ldots, x_n$ is created, the adversary can then determine an optimal solution for this input.

  The competitive ratio (for a minimization algorithm ALG in this setting is defined as :

$$\rho(ALG) = \lim sup_{OPT(x_1, \ldots x_n) \rightarrow \infty} \frac{\mathbb{E}_{D_1, \ldots, D_n}(ALG(x_1, \ldots, x_n, D_1, \ldots, D_n))}{\mathbb{E}(OPT(x_1, \ldots, x_n))}$$

Note that the denominator is now a random variable. Why?

Adaptive adversaries continued

- *Adaptive online adversary*. This adversary is an intermediate type of adversary. It has the benefit of observing the decisions of the algorithm but must also make decisions online. (If we have time, I plan to discuss stochastic probing algorithms where the ide of the adversary having to behave online is also used as a more reasonable benchmark for defining a competitive type of performance ratio.

  The competitive ratio (for a minimization algorithm ALG in this setting is defined as :

$$\rho(ALG) = \lim sup_{OPT(x_1,\ldots x_n)\to\infty} \frac{\mathbb{E}_{D_1,\ldots,D_n}(ALG(x_1,\ldots,x_n,D_1,\ldots,D_n))}{\mathbb{E}(OPT(x_1,d_1,\ldots,x_n,d_n))}$$

  where $d_1,\ldots,d_n$ is the sequence of decisions made by the online adverssary.
  Note that the denominator is again a random variable.

The relationship between the adversaries

The following figure summarizes the ratlative power of these adversaries:

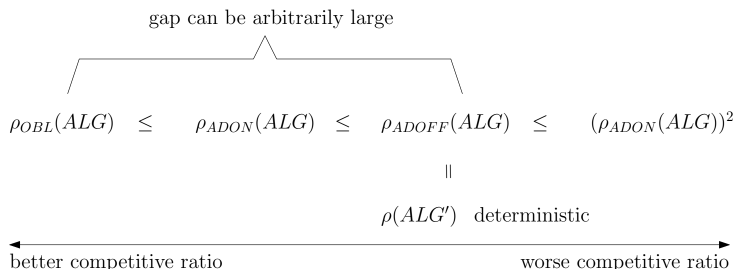

Figure 3.1: This figure summarizes the relationships between competitive ratios with respect to different types of adversaries.

The proportional knapsack problem: A natural example where randomness is very helpful

In the knapsack problem, we are given a set of items $\{s_i, v_i\}$ and a knapsack size bound B. The goal is to select a set S of items (or indices of items) that fit into the knapsack (i.e., $\sum_{i \in S} s_i \leq B$. (Without loss of generality (as in bin packing) we can assume $s_i \leq 1 = B$ for all i.)

It is not hard to show that no online (or even greedy algorithm) can have a constant approximation for the knapsack problem (although there is an FPTAS for this problem).

A special case of the knapsack problem is what we have called the proportional knapsack problem. It is also called the subset sum (optimization) problem. In this problem, we let $w_i = s_i = v_i$ for all i and then the problem is more simply to see what is the most size we can fit into the knapsack.

As we show, there is no deterministic constant competitive algorithm for the proportional knapsack problem.

The 1 random bit proportional knapsack algorithm

The following simple 1 random bit algorithm is competitive.

Algorithm 7 Simple randomized algorithm for Proportional Knapsack

---

procedure SimpleRandom
  Let $B \in \{0, 1\}$ be a uniformly random bit   ▷ W is the knapsack weight capacity
  if $B = 0$ **then**
    Pack items $w_1, \ldots, w_n$ greedily, that is if w_i still fits in the remaining weight knapsack capacity, pack it; otherwise, ignore it.
  else
    Pack the first item of weight $\geq W/2$ if there is such an item. Ignore the rest of the items.

$\rho_{OBL}(SimpleRandom) \leq 4OPT$

Aside: For any function $g(n)$, We give a rather artificial example where any deterministic algorithm obtain profit $= 1$, while there exists a randomized algorithm that achieves profit $\approx g(n)$.

A natural example where randomness does not help

Theorem 3.4.3 shows that any randomized algorithm ALG for one-way trading can be "de-randomized" while maintaing the same competitive ratio. In that proof we are viewing the randomized algorithm as having access to an arbitrary (perhaps even continuous) distribution and then being a mixture of deterministic algorithms where each of these deterministic algorithms comes from fixing some random r drawn from the distribution.

Since we are allowing continuous distributions we need to consider integrals. But if we are just viewing randomness as choosing a string of t bits, then the integrals can be replaced by summations over each of the possible 2^t random stings.

The proof shows how the mixture can be removed. Namely, suppose the random one-way algorithms ALG trades a fraction f_i of the remaining dollars on day i. That is, when r is fixed, the algorithm trades a fraction $f_i(r, p_1, \ldots, p_{i-1})$ on day i. Then we can create a deterministic one-way algorithm by "driving the mixture" into the fraction.

Randomization and de-randomizatiion for the time-series problem

Recall that an algorithm for time-series is a special case of a one-way trading algorithm. Consider the following randomized algorithm for the time-series problem when given U and L, upper and lower bounds on the conversion rates $\{p_i\}$.

Assume for simplicity that $L = 1$ and $U = 2^k$ for some $k > 0$. Partition the possible rates into k classes $C_0, C_1, \ldots, C_{k-1}$ where $C_j = \{p | 2^j < p \leq 2^{j+1}\}$. Randomly choose a $j \in \{0, 1, \ldots k-1\}$ and then convert on the first day i such that $p_i \in C_j$ and trade on last day otherwise (i.e., every day $p_i = 1$).

This is clearly a mixture of deterministic algorithms which by Theorem 3.4.3 can be de-randomized. It is easy to see that this achieves (in expectation) competitive ratio $\log \phi$ where $\phi = U/L$. Recall in contrast that no detreministic algorithm can do better than $\sqrt(\phi)$.

Claim: The de-randomization of the above algorithm is the mixture of RPPs algorithm given for the one-way trading algorithm.

De-randomization

Theorem 3.4.3 shows how to de-randomize any algorithm for the one-way trading problem. We also now know that for some problems (e.g., the proportional knapsack and time series problems), randomization is essential to obtain a good competitive ratio. That is, no deterministic online algorithm can come close to the performance of a specific randomized algorithm. This, of course, implies there is no possible way to de-randomize some algorithms so as to maintain (or come close to) the randomized competitive ratio.

Later in the text we will consider other de-randomizations, some of which will preserve the online framework while other de-randomizations will necessarily require that the de-randomized algorithm is no longer an online algorithm (e.g. the de-randomized algorithm is a "two-pass online" algorithm or an "online paralell stream").

NOTE: It is hard to define what de-randomization means. That is, when transforming randomized algorithm ALG into a deterministic algorithm ALG' , what is an allowable transformation?

Paging and the k-server problem

Paging (or caching) is, of course, one of the most studied problems, both in theory and in "practice". From the theoretical point of view, paging is the starting point for many generalizations and alternative measures (beyond the competitive ratio) of performance.

Arguably the most important generalization is the k-server problem which is defined a follows:

Let $\mathcal{M} = (M, d)$ be a metric space with at least $k + 1$ points. That is, $d(x, y)$ is a metric distance function satisfying:

- symmetry: $d(r, s) = d(s, r)$ for all r, s
- triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$ and
- $d(x, y) = 0$ iff $x = y$.

In the k-server problem, a request sequence $r_1, \ldots, r_n$ is a sequence of points $r_i \in M$ that must be served by one of k servers. If a request r_i is not occupied by a server, then an algorithm must move one of the k servers (say located at some location $r \in M$) at a cost of $d(r, r_i)$. The goal is to minimize the total cost of serving all requests.

Paging as a special case of the k-server problem

Paging is the special case of the uniform metric (i.e., $d(r, s) = 1$ for all $r \neq s$) when we have k cache locations and M consists of the main memory. Assume the cache is full. A request for a page in M not in the cashe is a *page fault* and we have to move that page into the cache and evict one of the pages in the cache. Each page fault has cost 1. (In one of the many variations of this basic paging model, there can be different costs associated with different external memory locations.)

In the k-server problem, we can assume without loss of generality that we never move a server except to serve a request and at all times the servers are located in different locations.

For paging, we can assume that we never evict a page unless there is a page fault. This is called *demand paging*.

Many possible paging algorithms

The following are some possible online paging algorithms: When a page fault occurs

- LRU (least recently used): evict the page whose most recent request was earliest.
- FIFO (first in first out): evict the page that has been in the cache for the longest time.
- LIFO (last in, first out): evict the page that has most recently been put in the cache.
- LFU: (least frequently used): evict the page that has been request the least since entering the cache
- FWF (flush when full): whenever the cache is full and a page fault occurs, flush out the entire cache and place the new request in the cache.

The good and the bad regading competitive analysis

A good measure of performance should help predict real behavior and also should be able to alllow us to gain insight as to which algorithm might be prefered.

We shall show that for *every* k-server problem (i.e. every metric space), no deterministic algorithm can be better than k-competitive. This then implies that no deterministic online paging algorithm can be better than k-competitive.

Experimental studies suggest that LRU (or some variant of it) is the best paging algorithm to use.

The good and bad news is that LRU, FIFO and FWF are all k-competitive while LIFO and LFU are not k-competitive. So while the competitve theory at least distinguishes LRU from LIFO and LFU, it does not distinguish itself from FIFO and FWF. Clearly FWF seems like a poor idea and in practice it is really a terrible idea. (See the figure on the next slide.)

Experimental results for competitiveness vs cache size

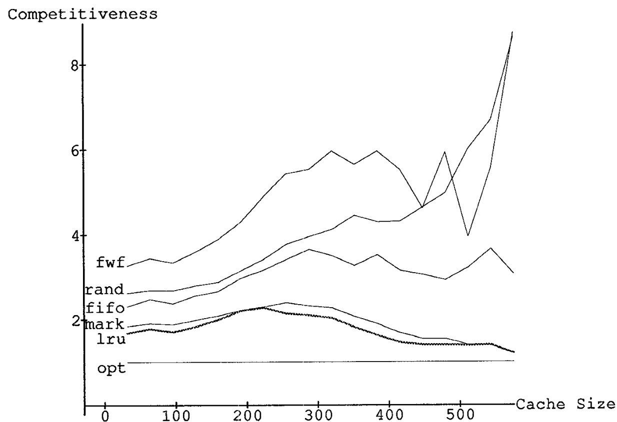

Figure: Figure from Young, SODA 1991

Some comments about paging

Some algorithms suffer *Belady's anomaly*, namely, that there are request sequences on which the algorithm will perform better (i.e., have less page faults) when there are $h < k$ cache pages.

LRU does not suffer Belady's anomaly but FIFO does.

The main criticism of competitive analysis (especially with regard to paging) is that adversarially generated request sequences are not found in practice. For paging, we know that page requests will tend to satisfy *locality of reference*; that is, the next page request will be "related" to immediately preceding page requests. This is a reasonable explanation for why LRU performs well in practice. This has led to a study of *Markov paging*.

LFD (longest forward distance; i.e, evict the page whose next request is latest) is an optimal *offline* algorithm that can be computed using dynamic programming.

Marking and conservative algorithms are all k-competitive

Consider cache size k and let $\sigma = r_1, r_2 \ldots$ be any request sequence. We can partition σ into *phases* as follows: phase 0 is the empty phase, phase i is the maximal request sequence following phase $i-1$ containing at most k *distinct* page requests.

Note that this k-phase partition is well-defined function of the request sequence and does not depend on any algorithm for paging.

Consider a k-phase partion for a sequence σ. We will *mark* pages as follows: At beginning of each phase, unmark all the pages in the cache. Then during a phase we mark a page when it is first requested during this phase. A *marking algoriithm* never evicts a marked page. In particular, the $k+1^{st}$ distinct page request r starts a new phase with r marked.

Every marking algorithm is k competitive

Consider a request sequence σ and it k-phase partition.

During a phase, a marking algorithm cannot fault twice on the same page and hence during a phase, any marking algorithm will fault at most k times.

On the other hand, counting the first request of a new phase, any algorithm, including OPT will fail at least once per phase.

LRU and FWF are marking algorithms and hence are k competitive.

FIFO is not a marking algorithm but it is *conservative* in the sense that on any consecutive subsequence of requests containing k or fewer distinct pages, the algorithm will incur k of fewer page faults. A similar proof shows that all conservative algorithms (and hence FIFO) are k competitive.

No online deterministic paging algorithm can be better than k competitive

Choose any set of $k+1$ pages $\{p_1, \ldots, p_{k+1}\}$.

- Initialize the cache arbitarily with k pages say the cache contains $p_1, \ldots, p_k$. Let the first request be the missing in cache page p_{k+1}. For every i, the next request r_{i+1} is the *unique page* not in cache. (Recall that we are assuming that we never evict a page unless there is a page fault so that this is a well defined request sequence.) Such an adversary is called a *cruel adversary*.

Two relatively easy extension

We mention two relatively easy extensions for adversarial deterministic online paging.

1. We have been charging 0 for a cache access and 1 for cache miss. This is an infinite ratio of fast to slow memory. Of course, the ratio between fast and slow is always finite. If a cache access costs 1 and a cache miss costs $s \geq 1$, the any marking algorithm has competitive ratio $\frac{k(s+1)}{k+s}$ and this is a tight bound.
2. One way to give the algorithm more of a "fighting chance" against a worst case adversary is to assume that the algorithm is allowed more cache pages than the adversary/OPT. Specifically suppose the algorithm has k pages and OPT has $h \leq k$ pages. The tight competitive ratio is then $\frac{k}{k-h+1}$

How much can randomization help in paging?

It turns out that like the time-series problem, randomization can provide an exponential improvement in the competitive ratio. Namely, there is a randomized paging algorithm that achieves competitive ratio H_k where $H_k = \sum_{i=1}^{k} \frac{1}{i}$. Note that $\ln k \leq H_k \leq \ln k + 1$.

The lower bound will utilize the *Yao minimax principle*, based on von Neuman's zero sum minimax theorem. Yao first applied this principle to establish time lower bounds. It has subsequently been used to establish lower bounds (for minimization problems) on the competitive ratio.

We will first informally state the minimax principle as it applies to the expected cost of an online algorithm for a minimization problem. (The principle applies much more braodly as we indicate in the text.) In particular, there is an analogous statement for a maximization problem. Then we will apply it to show that the competitive ratio for every randomized paging algorithm (for a cache of size k) is at least H_k.

The Yao Minimix Principle

Yao Principle applied to a minimization problem

Let X^n denote a distribution on sequences of n input items. The expected cost (with respect to the randomization) of a randomized algorithm on a worst case input sequence (i.e., given by an oblivious adversary) is at least as big as the expected cost (with respect to the distribution X^n) of the best deterministic algorithm.

NOTE: Proving that the princple as a theorem requires a precise statement. Instead we will see how it is proven within the paging example when we exhibit an appropriate distribution X^n on request sequences of length n.

A lower bound on the cost and competitive ratio for randomized paging

Let X^n be a distribution on input sequences $x_1, \ldots, x_n$ defined as follows: for each i, choose x_i uniformly and independently at radnom from $\{p_1, \ldots, p_{k+1}\}$.

We first consider the expected cost (i.e. page faults) of an arbitrary deterministic paging algorithm. Starting from any initial setting of the cache, the probability of each page request being a page fault is $\frac{1}{k+1}$. Hence the expected number of page faults for a random request sequence of length n is $\frac{n}{k+1}$.

We now have to calculate the expected cost of an optimal algorithm OPT (say LFD) for a random sequence of length n.

The expected cost of OPT on a random requeet sequence of length n

We consider the k-phase partition induced by the random input sequence. Say the partition is $B_1 B_2 \ldots B_{\ell(n)}$. Note that the lengths of these blocks and $\ell(n)$ are also random variables. (I am ignoring a final incomplete block.) They are not independent since we must have $\sum_{j=1}^{\ell(n)} |B_j| = n$. But as $n \to \infty$, the $|B_i|$ are acting almost as if they are independent identically distributed.

Let Z_i be the random variable equal to the number of requests before seeing the ith new page. In the text we argue that $\mathbb{E}[Z_i] = \frac{k+1}{k-i+2}$. and hence if B is a block, $\mathbb{E}[|B|] = \sum \mathbb{E}[Z_i] = (k+1)H_k$. Now we use an intuitive but technical result that $lim_{n\to\infty} \frac{n}{\mathbb{E}[\ell(n)]} = \mathbb{E}|B|$ (i.e., formalizing the meaning that the $|B_i|$ are basically acting as i.i.d. random variables.

We conclude that $\mathbb{E}[|\ell(n)|]$ (i.e. the expected number of blocks) is $(k+1)H_k$ and therefore the expected cost of OPT is $(k+1)H_k$

Concluding the proof for the lower bound on the competitive ratio for randomized paging

We have for any deterministic algorithm ALG_r that
We have $\mathbb{E}_{X^n} ALG_r(X^n) \geq H_k \mathbb{E}_{X_n}[OPT(X^n)]$
(I am ignoring a lower order term.)

Here now comes where we will use the Yao Principle.
Since a randomized algorithm is a mixture over deterministic algorithms, taking the expectation over each algorithm in the mixture, we have

$$\mathbb{E}_r \mathbb{E}_{X^n}[ALG_r(X^n)] \geq H_k \mathbb{E}_{X_n}[OPT(X^n)]$$

We now interchange the order of the expectations on the left side of the inequality, we have:

$$\mathbb{E}_{X^n} \mathbb{E}_r[ALG_r(X^n)] \geq H_k \mathbb{E}_{X_n}[OPT(X^n)]$$

There must then be one instantiation of the random input sequence (say x^n) that satisfies this inequality proving the desired result that:

$$\mathbb{E}_r[ALG_r(x^n)] \geq H_k[OPT(X^n)]$$

A randomized algorithm achieving competitive ratio $2H_k$

There are now two randomized algorithms that achieve the optimal competitive ratio H_k, for simplicty we will present the first randomized paging algorithm achieving ratio $2H_k$.

The algorithm is appropriately called MARK as it is a marking algorithm and indeed it was the first paging algorithm where marking algorithms were explicitly discussed. As in all marking algorithms, when a page in the cache is requested the page becomes marked. When a page fault occurs, an unmarked page is uniformly chosen at random chosen to be evicted. If there are no unmarked pages in the cache then all are unmarked and again a random page is evicted. In either case, the newly access page is brought into the cache and marked.

MARK is $2H_k$ competitive

We can assume MARK and OPT are in the same initial configuration with the cache full. AS usual, we need an upper bound on the expected number of page faults by MARK and a lower bound on the number of page faults by OPT.

For a request sequence, let $B_1, \ldots, B_\ell$ be the k-pahse partition for the request sequence. Consider the pages in block B_i. We classify these pages as *new* if they didn't appear in the previous block B_{i-1} or *old* if it is did appear in B_i.

We now bound the number of faults by MARK. The worst case for MARK is when new pages all occur before old pages. Every new page request in block B_i results in a page fault. Let m_i be the number of new pages in B_i so that we have $k - m_i$ old pages in B_i. Consider the start of phase i just before the page starting this phase is brought in. All k pages in B_i are unmarked. The m_i new pages will each cause a page fault.

Analysis of MARK continued

So we now want to bound the expected number of faults caused by first requests to the old $k - m_i$ pages. Consider the first request of an old page. It is still in the cache with probability $\frac{k-m_i}{k}$ and hence has probability $\frac{m_i}{k}$ of causing a page fault. The jth old page is in the cache with probability $\frac{k-m_i-(j-1)}{k-(j-1)}$ and thus causes a page fault with probability $\frac{m_i}{k-j+1}$.

Summing up the expected number of page faults in the ith phase is $m_i + \sum_{j=1}^{k-m_i} = m_i + m_i(H_k - H_{m_i} \leq m_i H_k$. The total number of page faults by MARK is $\sum_i m_i H_k$

It remains to lower biund the number of faultts by OPT. Counting the page requests in both B_{i-1} and B_I, there must be at least $k + m_i$ distinct page requests and hance at least m_i page faults by OPT. Hence the total number of page faults by OPT is at least $\frac{1}{2}(\sum_i m_i)$ since we are grouping OPT faults by pairs of phases.