CSCI 435: Algorithms and Complexity
6. Online Algorithms

- *secretary problem*
- *paging problem*
- *randomized online paging*

Online algorithms

Algorithmic design patterns.

- Greedy
- Divide-and-conquer
- Dynamic programming
- Network flow
- *Intractability*
- Randomization
- Online

Offline algorithm. One that receives all of its input up front.

Online algorithm. One that receives as input a sequence of requests $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ and serves each of these requests in order. When a request σ_i is served, the algorithm does not know the values of future requests σ_j for any $j > i$.

Online algorithms

Examples of online decision-making.

- An investor must decide whether to put their money into a mutual fund account, an investment certificate, or the stock market without knowing how the returns of any of these investment vehicles will fluctuate over the next year.

- A new skier must decide whether to rent or buy their skis without knowing how many days in the season will see snow on the ground.

Online algorithms: analysis

Competitive ratio. For any sequence of requests σ, let $\mathrm{ALG}(\sigma)$ denote the value produced by the online algorithm and let $\mathrm{OPT}(\sigma)$ denote the value produced by the best offline algorithm. Then the competitive ratio is

$$\frac{\mathrm{ALG}(\sigma)}{\mathrm{OPT}(\sigma)}.$$

The competitive ratio tells us how well the online algorithm performs compared to the optimal offline algorithm.

If, for all sequences of requests σ, we have that $\mathrm{ALG}(\sigma) \leq \alpha \cdot \mathrm{OPT}(\sigma)$, then our online algorithm is α-competitive.

Competitive analysis. To find the best possible online solution, we want to find $\max_\sigma(\mathrm{ALG}(\sigma)/\mathrm{OPT}(\sigma))$.

CSCI 435: Algorithms and Complexity
6. Online Algorithms

- *secretary problem*
- *paging problem*
- *randomized online paging*

Secretary problem

Motivation. You're the boss of a large company, and you need to hire a secretary. Your company is so popular that many people want a job there, but you're a busy person and can't read everyone's resumes.

Idea. Wing it!

- Meet with each candidate and make your decision after the interview.
- If you accept, that person is hired and the search ends.
- If you reject, that person leaves the building and is never seen again.

Objective function.

- Choosing the best candidate gives you a score of 1.
- Choosing anybody else gives you a score of 0.

Secretary problem: examples

Example 1. Suppose $n = 2$ candidates.
$\Longrightarrow$ 50% chance of success.

Example 2. Suppose $n = 3$ candidates.

- Always accept the first candidate.

$\Longrightarrow$ 33% chance of success.

- Always reject the first candidate and accept the second candidate if they're better than the first.
  We succeed if the candidates are ordered
  $$\{2 > 1 > 3\}, \{2 > 3 > 1\}, \{3 > 1 > 2\}.$$
  We fail if the candidates are ordered
  $$\{1 > 2 > 3\}, \{1 > 3 > 2\}, \{3 > 2 > 1\}.$$

$\Longrightarrow$ 50% chance of success.

Stopping rules

Naïve strategy. Choose a candidate at random (e.g., always the first candidate).
$\Longrightarrow$ 1/n chance of success.

Look, then leap. [Flood, 1958] Reject the first k candidates for some value of k, then choose the first candidate that is more qualified than any of the first k.
$\Longrightarrow$ 50% chance of success (when $n = 3$ and $k = 1$)
Better than the naïve 33% chance!

Caveat. There's no guarantee that the best candidate won't be included in the first k candidates who are rejected automatically.

Question. What value of k should we take to minimize the risk of accidentally rejecting the best candidate?

Stopping rules: analysis

Theorem. Using the "look, then leap" strategy, the optimal value of k is n/e, where $\mathrm{e} = 2.71828\ldots$ is Euler's constant.

Pf. Consider a uniformly random permutation of candidates.
Let i denote the index of the best candidate.
Clearly, if $i \leq k$, our strategy will fail.
If $i > k$, then our strategy will succeed if and only if the second-best candidate among the first i candidates is found in the first k positions of the sequence.

$$\Pr[\text{success} \mid \text{candidate } i \text{ is best}] = \Pr[\text{second-best of first } i \text{ candidates is among first } k \mid \text{candidate } i \text{ is best}]$$
$$= \frac{k}{i-1}. \quad \longleftarrow \text{because the permutation of candidates is uniformly random}$$

Stopping rules: analysis

Theorem. Using the "look, then leap" strategy, the optimal value of k is n/e, where $e = 2.71828\ldots$ is Euler's constant.

Pf (cont'd). We can write a direct expression for the probability of success:

$$\begin{aligned}
\Pr[\text{success}] &= \sum_{i=k+1}^{n} \Pr[\text{success} \mid \text{candidate } i \text{ is best}] \cdot \Pr[\text{candidate } i \text{ is best}] \\
&= \sum_{i=k+1}^{n} \frac{k}{i-1} \cdot \frac{1}{n} \\
&= \frac{k}{n} \cdot \sum_{i=k+1}^{n} \frac{1}{i-1} \\
&= \frac{k}{n} \cdot \sum_{i=k}^{n-1} \frac{1}{i} \\
&\approx \frac{k}{n} \cdot \ln\left(\frac{n-1}{k-1}\right)
\end{aligned}$$

← setting $x = k/n$, this expression has a local maximum at $1/e$

Rearranging this expression gives us $k = n/e$. ▪

Look, then leap

Corollary. [Bruss, 1984] Any algorithm for the secretary problem that uses the "look, then leap" stopping rule where the first n/e candidates are automatically rejected has a success probability of $1/e \approx 36.8\%$.

MATHEMATICAL GAMES

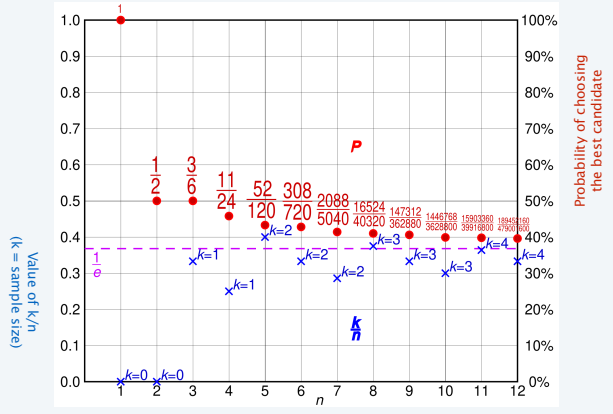
A fifth collection of "brain-teasers"

by Martin Gardner

3.

In 1958 John H. Fox, Jr., of the Minneapolis-Honeywell Regulator Co., and L. Gerald Marnie of the Massachusetts Institute of Technology devised an unusual betting game which they call Googol. It is played as follows: Ask someone to take as many slips of paper as he pleases, and on each slip write a different positive number. The numbers may range from small fractions of one to a number the size of a "googol" (1 followed by a hundred zeros) or even larger. These slips are turned face-down and shuffled over the top of a table. One at a time you turn the slips face-up. The aim is to stop turning when you come to the number that you guess to be the largest of the series. You cannot go back and pick a previously turned slip. If you turn over all the slips, then of course you must pick the last one turned.

Look, then leap



Reference: Wikipedia

CSCI 435: Algorithms and Complexity
6. Online Algorithms

- *secretary problem*
- *paging problem*
- *randomized online paging*

Computer memory hierarchy

Layers of memory. Computer memory is not monolithic.
- L1/L2 cache: fastest to access, smallest capacity.
- Random access memory (primary): fast to access, small capacity.
- Disk memory (secondary): slow to access, large capacity.
- Tape memory (tertiary/offline): slowest to access, largest capacity.

Accessing memory. Retrieving data stored in slow forms of memory incurs lengthy wait times.

Caching memory. Computers cache recently used data in pages to reduce wait times. But cache memory has the smallest capacity.

Page faults. When cache memory runs out of capacity and tries to access data that isn't paged, it must evict an existing page.

Paging problem

Formalization. Suppose we have a computer with two forms of memory: cache and disk. We have as much disk memory as we want, but only k pages of cache memory. When we try to access a page p_i, we check whether it's in the cache already.
- If it is, then no problem.
- If it isn't, then we incur a page fault: evict one page from the cache and replace it with p_i.

Paging problem. Given a sequence of page accesses $\{p_1, p_2, \ldots, p_n\}$ where $n \gg k$, what is the optimal strategy to minimize page faults?

Paging problem: offline algorithm

The MIN algorithm. Suppose we know the complete sequence of page accesses in advance. Then the paging problem becomes very easy!

```
PAGING-MIN (p_1, p_2, ..., p_n)
  S = ∅
  FOR (1 ≤ i ≤ n)
    IF (S contains p_i) CONTINUE
    ELSE
      IF (|S| < k)
        S = S ∪ p_i
      ELSE
        evict the page p_j in S whose access time is furthest in
        the future (or never)
```


Paging problem: offline algorithm

Example. Suppose $k = 4$ and we have five pages $\{p_1, p_2, p_3, p_4, p_5\}$.

Time	0	1	2	3	4	5	6	7	8	9
Request		p_3	p_1	p_4	p_2	p_5	p_2	p_1	p_2	p_3
Cached pages		p_3	p_3	p_3	p_3	p_3	p_3	p_3	p_3	p_3
		p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1
			p_4	p_4	p_5	p_5	p_5	p_5	p_5	p_5
				p_2	p_2	p_2	p_2	p_2	p_2	p_2

Page fault!
Evict the page accessed furthest in the future (or never)

Result. One page fault.

Paging problem: offline algorithm

Theorem. The MIN algorithm is the optimal offline paging algorithm.

Pf. By induction on the length n of the page access sequence.

- When $n = 1$, the theorem is trivially true.
- Suppose $n \geq 2$. Let us show that no offline algorithm A performs better than MIN.
  Denote the initial cache state by S_1.
  After accessing page p_1, the cache state becomes S_2.
  After accessing page p_2, the cache state becomes S_3, and so on.
  Denote the sequence of cache states A produces by $\{S_1, S_2, \ldots, S_n\}$.
  - If accessing p_1 does not incur a page fault, then $S_1 = S_2$ and we can shorten the cache state sequence length by 1. The theorem is then proved by induction.
  - If accessing p_1 incurs a page fault, then we consider two cases.

Paging problem: offline algorithm

Theorem. The MIN algorithm is the optimal offline paging algorithm.

Pf (cont'd). If accessing p_1 incurs a page fault, then we consider two cases.

- If there is a page $p \in S_1$ that is never accessed in the future, modify A to evict page p. Then A adopts the same strategy as MIN at time 1, and the theorem is proved by induction.
- If there is a page $p_j \in S_1$ that has latest future access time, but A evicts page p_i with earlier access time instead, modify A to use the following strategy B:
  - Swap all removals of p_i with p_j and vice versa until cache states of A and B are equal, then continue following A's strategy.

  It is then possible to show that B performs no better than A or MIN, and the theorem is proved by induction. ▪

Paging problem: online algorithms

Problem. The MIN algorithm can't be used with online data, and we can never do as well as the MIN algorithm if we don't receive the whole sequence in advance.

Online strategies.

- FIFO (first-in-first-out). Evict the page that has been in the cache longest.
- LIFO (last-in-first-out). Evict the page that has been in the cache shortest.
- LRU (least recently used). Evict the page that was last accessed longest ago.
- LFU (least frequently used). Evict the page that was accessed least.
- MRU (most recently used). Evict the page that was last accessed shortest ago.
- MFU (most frequently used). Evict the page that was accessed most.

Remarks. MRU and MFU are particularly unintelligent strategies.

LIFO and LFU strategies

Non-competitiveness. The LIFO and LFU strategies are suboptimal for instances of online paging because we can make them perform arbitrarily badly with certain page access sequences.

LIFO. This strategy on a cache of size k incurs a page fault on every access to pages p_k and p_{k+1} after time k with the following sequence:

$$p_1, p_2, \ldots, p_{k-1}, p_k, p_{k+1}, p_k, p_{k+1}, p_k, \ldots$$

LFU. This strategy on a cache of size k incurs a page fault on every access to pages p_k and p_{k+1} after time $m \cdot (k-1)$ with the following sequence:

$$(p_1)^m, (p_2)^m, \ldots, (p_{k-1})^m, (p_k, p_{k+1})^{m-1}$$

Theorem. The LIFO and LFU strategies are not α-competitive for any α.

FIFO strategy

Example. Suppose $k = 4$ and we have five pages $\{p_1, p_2, p_3, p_4, p_5\}$.

Time	0	1	2	3	4	5	6	7	8	9
Request		p_3	p_1	p_4	p_2	p_5	p_2	p_3	p_2	p_1
Cached pages		p_3	p_3	p_3	p_3	p_5	p_5	p_5	p_5	p_5
			p_1	p_1	p_1	p_1	p_1	p_3	p_3	p_3
				p_4	p_4	p_4	p_4	p_4	p_4	p_1
					p_2	p_2	p_2	p_2	p_2	p_2
						Page fault!		(*)		(*)

Evict the page that has been in the cache longest

LRU strategy

Example. Suppose $k = 4$ and we have five pages $\{p_1, p_2, p_3, p_4, p_5\}$.

Time	0	1	2	3	4	5	6	7	8	9
Request		p_3	p_1	p_4	p_2	p_5	p_1	p_3	p_5	p_4
Cached pages		p_3	p_3	p_3	p_3	p_5	p_5	p_5	p_5	p_5
			p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1
				p_4	p_4	p_4	p_4	p_3	p_3	p_3
					p_2	p_2	p_2	p_2	p_2	p_4
						Page fault!		(*)		(*)

Evict the page that was last accessed longest ago

FIFO and LRU strategies

Competitiveness. The FIFO and LRU strategies are competitive for any cache size!

Theorem. The LRU strategy is k-competitive for any cache size k.

Pf. Consider the execution of LRU on a page access sequence

$$\sigma = \sigma_1, \sigma_2, \ldots, \underbrace{\sigma_i}_{\text{first page fault}}, \underbrace{\sigma_{i+1}, \ldots, \sigma_j}_{\text{first phase}}, \underbrace{\sigma_{j+1}, \ldots, \sigma_\ell}_{\text{second phase}}, \ \ldots$$

The sequence is divided into phases where each phase incurs exactly k page faults and ends in a page fault.

For example, the first phase of the above page access sequence ends at time

$$j = \min\{t \mid \text{the LRU strategy incurs } k \text{ page faults between requests } \sigma_{i+1} \text{ and } \sigma_t\}.$$

FIFO and LRU strategies

Competitiveness. The FIFO and LRU strategies are competitive for any cache size!

Theorem. The LRU strategy is k-competitive for any cache size k.

Pf (cont'd). **LRU incurs k page faults** per phase. How many does MIN incur?

1. If LRU faults twice on a page p in one phase:

   After the first fault on p, the page is cached and removed when it becomes least recently used. If p faults a second time, then all k cached pages must have been added between the first fault and second request, so $k + 1$ different pages were requested.

   ⇒ **MIN incurs at least 1 page fault**

2. If LRU faults on k different pages in one phase:

   Suppose page p was the last fault in the previous phase. Then p must be cached along with $k - 1$ other pages in this phase. During the phase, k requests are handled and none of them involve p, so p must be evicted to accommodate one of the k requests.

   ⇒ **MIN incurs at least 1 page fault**

$$C_{\text{LRU}}(\sigma) \leq k \cdot C_{\text{MIN}}(\sigma)$$

FIFO and LRU strategies

Competitiveness. The FIFO and LRU strategies are competitive for any cache size!

Theorem. The FIFO strategy is k-competitive for any cache size k.

Pf. Similar to the proof for LRU.

$$C_{\text{FIFO}}(\sigma) \leq k \cdot C_{\text{MIN}}(\sigma)$$

Optimality. FIFO or LRU are the best strategies out of all that we've seen.

Theorem. For all deterministic online paging algorithms A, there exists an access sequence σ where $C_{\text{A}}(\sigma) \geq k \cdot C_{\text{MIN}}(\sigma)$ for any cache size k.

Real world. In practice, LRU performs much better than FIFO.

CSCI 435: Algorithms and Complexity
6. Online Algorithms

- *secretary problem*
- *paging problem*
- *randomized online paging*

Downsides of online paging

Optimality? FIFO and LRU strategies are k-competitive, but this is terrible on a size-k cache. A competitive ratio closer to 1 is desirable.

Determinism. FIFO and LRU strategies are deterministic and susceptible to exploits from an adversary.

Oblivious adversaries. An adversary can create a special page access sequence based on knowledge of our paging strategy and distribution of inputs that will force worst-case performance. An *oblivious* adversary cannot change the page access sequence after the algorithm begins execution.

Paging problem: randomized online algorithms

Solution. Devise a strategy that doesn't exhibit deterministic behaviour.

Randomized online strategies.

- RANDOM. When the cache is full, evict a page at random.
- MARKING. Mark each page as it is accessed in the cache. When the cache is full, evict an unmarked page at random. When all cached pages are marked, unmark them all.

Oblivious adversaries. Impossible for an oblivious adversary to create a special page access sequence to force worst-case performance: they don't know how these strategies will behave in advance.

RANDOM strategy

Theorem. The RANDOM strategy is k-competitive for any cache size k.

Pf. Consider the performance of MIN vs. RANDOM on a size-k cache containing page p_{k+1} along with $k-1$ other pages from the set $\{p_1, \ldots, p_k\}$.

Suppose the page access sequence is

$$\sigma = \{p_1, p_2, \ldots, p_k, p_1, p_2, \ldots, p_k, p_1, \ldots\}.$$

Note that page p_{k+1} is never accessed.

- MIN: evicts page p_{k+1}, incurs 1 page fault.
- RANDOM: evicts a page at random, which may or may not be p_{k+1}.
  Then $\Pr[p_{k+1} \text{ is evicted}] = 1/k$, and each eviction is independent.
  We can model this using a geometric distribution with $p = 1/k$.
  This gives an expected value of $1/(1/k) = k$ page faults.

$$C_{\text{RANDOM}}(\sigma) \leq k \cdot C_{\text{MIN}}(\sigma)$$

MARKING strategy

Key idea. Combine the LRU and RANDOM strategies.
Remove pages at random, but ignore recently accessed pages.

```
PAGING-MARKING (p_1, p_2, ..., p_n)
  FOR each request for a page p
    IF page p is cached
      mark p; CONTINUE
    IF all cached pages are marked
      unmark all pages
    choose a cached unmarked page q uniformly at random
    evict page q
    cache and mark page p; CONTINUE
```


MARKING strategy

Theorem. The MARKING strategy is $2H_k$-competitive for any cache size k, where $H_k = \sum_{i=1}^{k} 1/i$.

Pf. Consider the page access sequence $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, partitioned into phases $(\sigma_{\ell'}, \ldots, \sigma_r)$ in such a way that for each phase (excluding the last):

- exactly k distinct pages are accessed in each phase;
- adding the request σ_{r+1} increases the number of distinct pages accessed to $k+1$.

By how MARKING works, this partitioning ensures each phase begins with no pages marked and ends with all pages marked.

Terminology:

- Page p accessed during phase i is called *old* if the page was previously accessed during phase $i-1$.
- Page p accessed during phase i is called *new* otherwise.

Accessing an *old* page incurs either 0 or 1 page faults, while accessing a *new* page always incurs 1 page fault.

MARKING strategy

Theorem. The MARKING strategy is $2H_k$-competitive for any cache size k, where $H_k = \sum_{i=1}^{k} 1/i$.

Pf (cont'd). Consider the worst case where all *new* pages are accessed before any *old* page in a given phase. Let m_i = number of *new* pages accessed in phase i. Then the number of *old* pages accessed in phase i is $k - m_i$.
After all m_i *new* pages are accessed, consider the subsequence of *old* pages $\{o_1, o_2, \ldots, o_{k-m_i}\}$. When accessing a page o_j, the cache holds $k - m_i - j + 1$ unmarked *old* pages out of the total $k - j + 1$ *old* pages not yet encountered in the phase. Then

$$\Pr[o_j \text{ is drawn randomly}] = \frac{k - m_i - j + 1}{k - j + 1}$$

and accessing page o_j incurs a page fault with probability

$$1 - \frac{k - m_i - j + 1}{k - j + 1} = \frac{m_i}{k - j + 1}.$$

MARKING strategy

Theorem. The MARKING strategy is $2H_k$-competitive for any cache size k, where $H_k = \sum_{i=1}^{k} 1/i$.

Pf (cont'd). The expected number of *old* page faults is

$$\sum_{j=1}^{k-m_i} \left(1 - \frac{k-m_i-j+1}{k-j+1}\right) = \sum_{j=1}^{k-m_i} \frac{m_i}{k-j+1}$$
$$= m_i \cdot \sum_{j=1}^{k-m_i} \frac{1}{k-j+1}$$
$$= m_i \cdot \sum_{j=m_i+1}^{k} \frac{1}{j}$$

so the expected number of total MARKING page faults is

$$\underbrace{m_i}_{\text{new pages}} + \underbrace{m_i \cdot \sum_{j=m_i+1}^{k} \frac{1}{j}}_{\text{old pages}} \leq m_i \cdot H_k.$$

MARKING strategy

Theorem. The MARKING strategy is $2H_k$-competitive for any cache size k, where $H_k = \sum_{i=1}^{k} 1/i$.

Pf (cont'd). In every two consecutive phases, a total of $k + m_i$ distinct pages are accessed, meaning MIN incurs at least m_i page faults.

Taking the sum over all phases and adjusting for double-counting, MIN incurs a total of $(1/2) \cdot \sum_i m_i$ page faults.

Combining this with the bound on the previous slide gives a competitive ratio of $2 \cdot H_k$.

$$C_{\text{MARKING}}(\sigma) \leq 2 \cdot H_k \cdot C_{\text{MIN}}(\sigma)$$

MARKING strategy

Optimality. In practice, the performance of MARKING may be better than the $2 \cdot H_k$ competitive ratio. But it can't be too much better than that.

Theorem. No randomized online paging algorithm can be better than H_k-competitive for any cache size k.

Competitive Paging Algorithms

Amos Fiat,*[1] Richard M. Karp,†[2] Michael Luby,‡[3]
Lyle A. McGeoch,§ Daniel D. Sleator,¶[4] and Neal E. Young#[5]

The *paging problem* is that of deciding which pages to keep in a memory of k pages in order to minimize the number of page faults. We develop the *marking algorithm*, a randomized on-line algorithm for the paging problem. We prove that its expected cost on any sequence of requests is within a factor of $2H_k$ of optimum. (Where H_k is the kth harmonic number, which is roughly $\ln k$.) The best such factor that can be achieved is H_k. This is in contrast to deterministic algorithms, which cannot be guaranteed to be within a factor smaller than k of optimum. An alternative to comparing an on-line algorithm with the optimum off-line algorithm is the idea of comparing it to several other on-line algorithms.