# The Worst Page-Replacement Policy

**Kunal Agrawal** KUNAL_AG@MIT.EDU
**Jeremy T. Fineman** JFINEMAN@MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge, MA 02139

## 1. Introduction

Consider a computer system with a two-level memory hierarchy consisting of a small fast memory of size $k$ and a large slow memory. Memory is divided into fixed-size pages. Each memory access indicates an access into a particular page of memory. If the page is located in fast memory, the access has no cost. If the page is located only in slow memory, the access induces a ***page fault***, whereby the page must be moved from slow memory into fast memory (possibly evicting a page that is currently stored in fast memory). A page fault has a cost of one.

Research in the area of page-replacement strategies focuses on strategies that reduce the number of page faults. If all the page requests are known a priori (***offline***), the optimal strategy is to replace the page whose next request occurs furthest in the future [Belady, 1966]. An ***online*** strategy must make its decisions at the time that each page request arrives, without any knowledge of the future accesses.

Since the performance of any online strategy depends on the input sequence, Sleator and Tarjan introduce ***competitive analysis*** [Sleator & Tarjan, 1985] to analyze these strategies. An online strategy $A$ is $c$-competitive if there exists a constant $\beta$ such that for every input sequence $\sigma$,

$$A(\sigma) \leq c \cdot \mathrm{OPT}(\sigma) + \beta \,,$$

where $A(\sigma)$ is the cost incurred by the algorithm $A$ on the input sequence $\sigma$, and $\mathrm{OPT}(\sigma)$ is the cost incurred by the optimal offline strategy for the sequence $\sigma$. Sleator and Tarjan prove that there is no online strategy for page replacement that is better than $k$-competitive, where $k$ is the memory size. Moreover, the ***least-recently-used (LRU)*** heuristic, whereby the page evicted is always the one least recently used, is $k$-competitive. If the offline strategy operates on a memory that is twice the size of that used by the online strategy, LRU is 2-competitive.

In this paper we are interested in finding a "reasonable" online strategy that causes as many page faults as possible. We assume that the fast memory is initially empty. A ***reasonable*** strategy[1] follows two rules:

1. It is only allowed to evict a page from fast memory when the fast memory is full.
2. It is only allowed to evict a page from fast memory when that page is being replaced by the currently requested page, and the currently requested page does not already reside in fast memory.

Although this problem has no practical motivation, it is fun and theoretically interesting.

The optimal offline strategy $\mathrm{OPT}$ for the problem of maximizing page faults discards the page that will be requested next. An online strategy $A$ is $c$-competitive if there exists a constant $\beta$ such that for every input sequence $\sigma$,

$$A(\sigma) \geq \mathrm{OPT}(\sigma)/c - \beta \,,$$

where $A(\sigma)$ is the number of page faults incurred by the algorithm $A$ on the input sequence $\sigma$, and $\mathrm{OPT}(\sigma)$ is the number of page faults incurred by the optimal offline strategy on the sequence $\sigma$.

Throughout this paper, when we talk about strategies being competitive, we mean with respect to the offline strategy that maximizes page faults. An optimal strategy is therefore the "worst" page-replacement policy.

The rest of this paper is organized as follows. Section 2 proves that there is no deterministic, competitive, online algorithm to maximize page faults, and that no (randomized) algorithm is better than $k$-competitive. Section 3 gives an algorithm that is $k$-competitive, and hence optimal. Section 4 proves that a direct-mapping strategy is also the worst possible strategy under the assumption that page locations are random. Most proofs are omitted for space reasons.

## 2. Lower bounds

This section gives lower bounds on the competitiveness of online strategies for maximizing page faults.

The following lemma states that there is no deterministic online strategy that is competitive with the offline strategy.

[1] Once unreasonable strategies are allowed, one could design a strategy that uses only one location on the fast memory. This strategy will perform optimally for maximizing page faults, but it doesn't make much sense in a real system.

**Lemma 1** *Consider any deterministic strategy $A$ with a fast-memory size $k \geq 2$. For any $\varepsilon > 0$ and constant $\beta$, there exists an input $\sigma$ such that $A(\sigma) < \varepsilon \cdot \mathrm{OPT}(\sigma) - \beta$.*

*Proof.* Consider a sequence $\sigma$ that begins by requesting pages $v_1, v_2, \ldots, v_{k+1}$. After page $v_k$ is requested, all strategies have a fast memory containing pages $v_1, \ldots, v_k$. At the time $v_{k+1}$ is requested, one of the pages must be evicted from the fast memory. Suppose that the deterministic strategy $A$ chooses to evict page $v_i$. Then consider the sequence $\sigma = v_1, v_2, \ldots, v_k, v_{k+1}, v_j, v_{k+1}, v_j, v_{k+1}, \ldots$, that alternates between $v_{k+1}$ and $v_j$ for some $j$ with $1 \leq j \leq k$ and $i \neq j$. After $v_{k+1}$ is requested, both $v_j$ and $v_{k+1}$ are in $A$'s fast memory. Thus, $A$ incurs only the first $k+1$ page faults. The offline strategy OPT incurs a page fault on every request (by evicting page $v_j$ when $v_{k+1}$ is requested and vice versa). Extending the length of the sequence proves the lemma. □

Lemma 1 also holds even if we introduce resource augmentation. That is, even if the deterministic strategy is allowed a smaller fast memory of size $k_{on} \geq 2$ than the fast memory $k_{off} \geq k_{on}$ used by the offline strategy, there is still no competitive deterministic strategy.

The following lemma states that no randomized strategy is better than expected $k$-competitive when both the online and offline strategies have the same fast-memory size $k$. Moreover, when the offline strategy uses a fast memory of size $k_{off}$ and the online strategy has a fast memory of size $k_{on} \leq k_{off}$, no online strategy is better than $k_{off}/(k_{off} - k_{on} + 1)$. We omit the proof due to space limitations.

**Lemma 2** *Let $k_{off}$ be the fast-memory size of the offline strategy and $k_{on} \leq k_{off}$ be the fast-memory size of the online strategy. Consider any (randomized) online strategy $A$. For any $c < k_{off}/(k_{off} - k_{on} + 1)$ and constant $\beta$, there exists an input $\sigma$ such that $E[A(\sigma)] < \mathrm{OPT}(\sigma)/c - \beta$.*

## 3. Most-recently used

This section describes two $k$-competitive strategies. The first strategy uses one step of randomization followed by the deterministic "most-recently-used" (MRU) heuristic. The second strategy uses more randomization to achieve the optimal result even when the offline and online strategies have different fast-memory sizes.

Since least-recently-used (LRU) is optimal with respect to an offline strategy that minimizes page faults, it is reasonable to expect MRU to be optimal for maximizing page faults. This strategy, however, is deterministic, and Lemma 1 states that no deterministic strategy is competitive. Instead we consider a ***randomized MRU*** strategy, where the first page evicted (when the $(k+1)$th distinct page is requested) is chosen at random. All subsequent requests follow the MRU strategy. This strategy avoids the alternating-request problem from the proof of Lemma 1.

**Theorem 3** *Randomized MRU is expected $k$-competitive, where $k$ is the fast-memory size.*

This result does not match the lower bound from Lemma 2. In particular, it does not generalize to the case in which the online and offline strategies have different fast-memory sizes. We have a strategy called "reservoir MRU" that uses more randomization. The main idea behind our ***reservoir MRU*** strategy is to keep a reservoir of $k_{off} - 1$ page, where each previously requested page is in the reservoir with equal probability.[2] The reservoir MRU strategy works as follows. For the first $k_{on}$ distinct requests, the fast memory is not full, thus there are no evictions. After this time, if there is a request for a previously requested page $v_i$, and the page is not in fast memory, then the most recently requested page is evicted. When the $n$th new page is requested, for any $n > k_{on}$, the most recently requested page is evicted with probability $1 - (k_{off} - 1)/(n-1)$. Otherwise, a fast-memory location (other than the most-recently-used page's) is chosen uniformly at random, and the page at that location is evicted.

The following theorem matches the lower bound given by Lemma 2, and hence reservoir MRU is optimal.

**Theorem 4** *Reservoir MRU is expected $k_{off}/(k_{off} - k_{on} + 1)$-competitive, where $k_{off}$ is the fast-memory size of the offline strategy, and $k_{on} \leq k_{off}$ is the fast-memory size for reservoir MRU.*

This theorem means that when the offline strategy and reservoir MRU have the same fast-memory size $k$, reservoir MRU is $k$-competitive. When reservoir MRU has fast-memory size $k$ and the offline strategy has fast-memory size $(1 + 1/c)k_{on}$, reservoir MRU is $(c+1)$-competitive, which is analogous to Sleator and Tarjan's [Sleator & Tarjan, 1985] result for LRU.

## 4. Direct Mapping

This section considers a particular page-replacement strategy used in real systems called "direct mapping" and proves that under some assumption of randomness, the direct-mapping strategy is $k$-competitive.

In a direct-mapping strategy [Patterson & Hennessy, 1998], each page is mapped to a particular location on the fast memory. If the page is requested and it is not on the fast memory, it evicts the page in that location. This strategy does not follow the rules of a reasonable strategy setup

[2] This technique is inspired by reservoir sampling [Vitter, 1985], which is where we came up with the name.

in Section 1. In particular, it may evict a page from the fast memory before the fast memory is full. We still think it is interesting to consider this strategy in the context of maximizing page faults as it is a strategy commonly implemented for the normal page-replacement problem (where the goal is reversed).

The direct-mapping strategy we consider is as follows. Each time a new page is requested, that page is mapped to a random location on the fast memory uniformly at random. Every time that page is requested again, it maps to the same location. The following theorem states that this strategy is $k$-competitive.

**Theorem 5** *The direct-mapping strategy is $k$-competitive, where $k$ is the fast-memory size of the online and offline strategies.*

In real direct-mapping strategies, pages are not typically mapped at random. Instead, they are mapped based on the lower order bits in the address. Theorem 5 states that as long as the pages are located at random locations in memory, then (deterministic) direct-mapping is the worst possible strategy. This assumption of random locations may seem a bit pessimistic in a real program. On a machine with some time sharing, however, each application may be located at a random offset in memory, so as long as the applications don't access too many pages during a time slice, the theorem still applies.

## References


Belady, L. A. (1966). A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, *5*, 78–101.

Patterson, D. A., & Hennessy, J. L. (1998). *Computer organization & design: The hardware / software interface*. San Francisco, CA: Morgan Kaufmann. Second edition.

Sleator, D. D., & Tarjan, R. E. (1985). Amortized efficiency of list update and paging rules. *Communications of the ACM*, *28*, 202–208.

Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, *11*, 37–57.