Institut für Informatik
Fachbereich Informatik und Mathematik

GOETHE
UNIVERSITÄT
FRANKFURT AM MAIN

Online Paging for Flash Memory Devices

Annamária Kovács, Ulrich Meyer, Gabriel Moruz
and Andrei Negoescu

2009

Frankfurter Informatik-Berichte

Institut für Informatik • Robert-Mayer-Str.11-15 • D-60325 Frankfurt am Main, Germany

ISSN 1868-8330

Online Paging for Flash Memory Devices

Annamária Kovács* Ulrich Meyer*,† Gabriel Moruz*,†
Andrei Negoescu*

Abstract

We propose a variation of online paging in two-level memory systems where pages in the fast cache get modified and therefore have to be explicitly written back to the slow memory upon evictions. For increased performance, up to α arbitrary pages can be moved from the cache to the slow memory within a single joint eviction, whereas fetching pages from the slow memory is still performed on a one-by-one basis. The main objective in this new α-paging scenario is to bound the number of evictions. After providing experimental evidence that α-paging can improve the performance of flash-memory devices in the context of translation layers we turn to the theoretical connections between α-paging and standard paging. We give lower bounds for deterministic and randomized α-paging algorithms. For deterministic algorithms, we show that an adaptation of LRU is strongly competitive, while for the randomized case we show that by adapting the classical Mark algorithm we get an algorithm with a competitive ratio larger than the lower bound by a multiplicative factor of approximately 1.7.

1 Introduction

In recent years flash memory is becoming increasingly popular as a viable storage support, especially for mobile computing. Flash memory devices are lighter, more shock-resistant, and consume less power than traditional hard-disks. For these reasons, flash memory is an appealing solution for end-user storage, partly even replacing traditional hard-disks. Motivated by the fact that, unlike traditional hard-disks, flash memory achieves the best performance when writes are done in blocks of size larger than the read block sizes [2], in this paper we consider paging algorithms for these devices. The key difference to traditional paging is that when a page fault occurs and the memory is full, instead of evicting only one page, up to α pages can be jointly evicted before the new page is loaded, for some fixed parameter $\alpha \geq 1$. The goal is to minimize the number of evictions.

*Institut für Informatik, Goethe-Universität Frankfurt am Main, Germany.

†Partially supported by the DFG grant ME 3250/1-1, and by MADALGO – Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

Flash memory. Flash memory consists of an array of memory cells, divided into a number of *blocks* of α consecutive *pages*, where each page is a group of consecutive memory cells. Reading and writing are done on a page basis, but overwriting single pages is usually not possible. Instead, overwriting is done by *erasing a whole block* and then writing the new data. Since each block can sustain only a limited number of erase operations, typical flash memory devices include a wear-leveling mechanism that ensures an even usage of the blocks in time.

Because erase operations are slow, in applications that modify (i.e., overwrite) pages on disk in an unstructured way hardly any performance gain is obtained when replacing hard-disks by flash memory. Frequently, this problem can be resolved using an additional intermediate software-layer, i.e. different than the wear-leveling mechanism, that bundles up to α write requests (pages) and writes them jointly to new consecutive locations, thus exploiting the improved performance when writing in larger blocks. To subsequently find the respective data under their new locations on the flash-device, an internal-memory translation-table for page locations has to be maintained, too. Also, occasionally device space needs to be reclaimed by compressing blocks with respect to outdated pages. Such software translation-layers can be found both in algorithmic research (e.g. [2]) and commercial products (e.g. EasyCo's Managed Flash Technology [9]). Instead of actually transferring data blocks back and forth between main memory and flash device it is even more efficient to buffer as many blocks as possible in internal-memory. This is classically done using paging algorithms. Motivated by the asymmetry between reads and writes in flash devices, we adapt classical paging by having evictions done in groups of up to α pages.

Most other previous algorithmic works for flash memory focused on memory management and wear-leveling, i.e. another block re-mapping *within* the flash device to avoid a premature block wear-out, and flash-tailored file-systems (see e.g. [11] for an overview). Typically the software translation-layer with its write-page bundling has a positive effect with respect to efficiency. Recently, thorough benchmarks for flash memories were conducted [3, 8], and based on their findings computational models exploiting the characteristics of these devices were proposed [2]. Other works use flash memory for model checking [5], route planning on mobile devices [12, 17], or on flash-aware R-trees and dictionaries [15, 19, 20].

Paging algorithms. Online algorithms are not provided with the input in advance and therefore must serve input requests as they arrive. To measure the efficiency of such algorithms, Sleator and Tarjan [18] considered comparing their cost against the cost of an optimal offline algorithm, i.e. an algorithm that knows the input sequence in advance and processes it optimally. The resulting measure, denoted later *competitive ratio* [14], states that an online algorithm A is c-competitive if $A(\sigma) \leq c \cdot OPT(\sigma) + b$ for any input sequence σ, where b is a constant, and $A(\sigma)$ and $OPT(\sigma)$ are the costs of A and an optimal offline

algorithm respectively (if A is randomized, $A(\sigma)$ is the expected cost of A).

Over the last decades, paging has been extensively studied in a variety of settings. In classical paging, we are provided with a two-level memory, a fast memory that can hold up to k pages and a disk that can store infinitely many pages. Given as input a sequence of pages, an algorithm must decide which pages to store in the memory so that it incurs as few page faults as possible, where a page fault occurs when some page does not reside in the memory when requested. In [18] it was proved that the competitive ratio of any deterministic algorithm is at least k, and that popular algorithms such as FIFO and LRU match this bound. Fiat et al. [10] proved a competitive ratio of at least H_k for any randomized paging algorithm, where $H_k = \sum_{i=1}^{k} 1/i$ is the k-th harmonic number. They gave an algorithm, denoted Mark, which is $(2H_k - 1)$-competitive. This bound was further improved in [16], where a H_k-competitive algorithm was proposed. More recently, Achlioptas et al. [1] gave another H_k-competitive algorithm which is more practical. For a detailed view on paging algorithms, we refer the interested reader to comprehensive surveys [4, 7].

Our results. We propose α-paging as an adaptation of classical paging to improve the practical behavior of flash memory devices in the context of software translation layers like EasyCo's Managed Flash Technology. It is similar to classical paging, except that arbitrary sets of up to α pages are jointly evicted. Since in practice writes are typically more expensive than reads, we count the number of such joint evictions instead of page faults. More specifically, we are provided with a fast memory that can hold k pages and a slow memory which can hold infinitely many pages. The input consists of a sequence σ of pages to be served by the algorithm. For some request of page p, if it is not in the memory we say that a page fault occurs. Evicting pages from the fast to the slow memory is done in groups of at most α arbitrary pages. Therefore, each eviction increases the amount of free slots in the memory by up to α. As previously specified, the cost of the algorithm is given by the number of evictions performed. More generally, at any step, jointly evicting x pages costs $\lceil x/\alpha \rceil$.

We show that in our model it is easy to adapt classical paging algorithms, such as the optimal offline MIN [6], LRU, and Mark [10]. However, due to the fact that up to α pages are jointly evicted instead of only one, competitive ratios achieved by these algorithms are different and their analysis becomes significantly more involved. We prove lower bounds on the competitive ratio for randomized and deterministic online algorithms. In particular, the competitive ratios of deterministic and randomized algorithms cannot be smaller than k/α and $(H_{k+\alpha-1} - H_{\alpha-1})/(H_{2\alpha-1} - H_{\alpha-1})$ respectiviely, which are generalizations of the respective lower bounds for $\alpha = 1$. We show that, like in classical paging, our adaptation of LRU matches the deterministic lower bound. For our randomized version of Mark we prove that it achieves a competitive ratio of $((H_k - H_{2\alpha-1})/(H_{3\alpha-1} - H_{2\alpha-1})) + 3$. For large enough values of k and α this bound is by a factor of about 1.7 larger than the lower bound, whereas the classical Mark has a competitive ratio twice the lower bound.

Outline. The remainder of the paper is structured as follows. In Section 2 we motivate and discuss α-paging. We then prove lower bounds for α-paging algorithms, both deterministic and randomized, in Section 3. We propose deterministic and randomized algorithms in Sections 4 and 5, respectively.

2 α-paging

In this section we first give empirical results that motivate the α-paging setting, and then we turn to discuss generic properties of α-paging algorithms viewed as generalizations of classical paging algorithms.

Motivation. As shown in [3], flash memory based solid-state disks have a significantly different behavior than traditional hard-disks. While both devices read and write in blocks of data, hard-disks use the same block size for reading and for writing, but for the solid-state disks the best performance is achieved when writing is done in blocks significantly larger than the ones used for reading. This is due to the intrinsic constructive details of flash memory, where reading is done on a page basis, while writing in blocks matching the erase block size prevents writes at random locations and together with it periodic reorganization of the data. In [2] it was shown that models using different block sizes for reading and writing achieve accurate predictions for the running times of a variety of algorithms exhibiting diverse I/O patterns.

We conduct experiments to demonstrate the practical relevance of writing in large blocks of data. We perform random writes in a very large array (about 1.5 the size of the memory in our case) in two different settings. In the first setting we write the modified page immediately, whereas in the second one we employ a translation layer, as in [2], which groups together modified pages and writes them on the flash disk as a large block. This way, *any* pages can be grouped together in neighboring physical locations on the flash disk, regardless of the addresses from where they were loaded in memory. We measure the running time when varying the amount of random writes. For our experiments, data is read in blocks of sizes 128KB and 4KB respectively, and written back to the flash disk in blocks of size 4MB when buffered. We note that for this particular disk the best performance is achieved when the block size for reading is 128KB, and writing in blocks of 4MB ensures good performance as well.

The results of our experiments are shown in Figure 1. We note that in both cases when writing large buffers, corresponding to evicting many pages at once, achieves significantly better performance than when writing data non-buffered. The improvements in running times are of about 250% when the read-block size is 128KB and this figure increases to about 1800% when reading in smaller blocks of 4KB. This confirms that evicting large blocks, i.e. groups of pages, yields significant performance improvements.

α-paging and classical paging. We note that α-paging is a generalization of classical paging. Every paging algorithm in the classical model is a valid α-

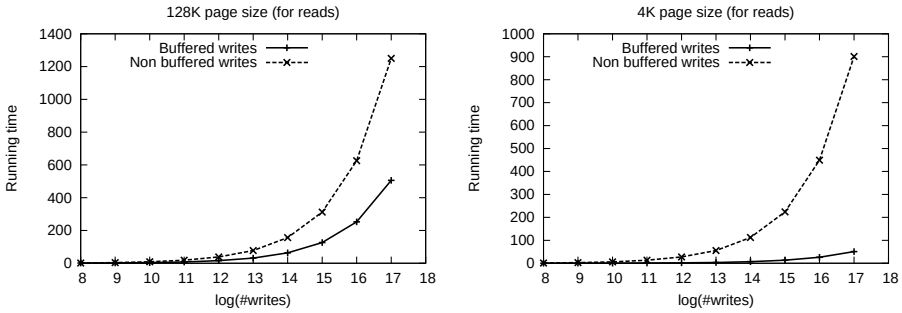

Figure 1: Running times (in seconds) for buffered and non-buffered random writes on SSDs, when reading in blocks of 128 KB (left) and 4 KB(right).

paging algorithm and vice versa, by performing identical page replacements in both models. We denote by $A^\alpha(\sigma)$ the cost of some algorithm A when processing the request sequence σ in the α-paging model (note that $\alpha = 1$ corresponds to classical paging). Since an eviction in the α-model corresponds to at most α evictions in the classical model, we have that $A^\alpha(\sigma) \leq A^1(\sigma) \leq \alpha \cdot A^\alpha(\sigma)$. This inequality also holds for the cost of the optimal offline algorithm denoted by $OPT^\alpha(\sigma)$, which adapts its decisions to the value of α. Given a c-competitive online algorithm A in the α-model for fixed $\alpha \geq 1$, we obtain that A has a competitive ratio of at most $\alpha \cdot c$ in the classical model:

$$\frac{A^1(\sigma)}{OPT^1(\sigma)} \leq \frac{\alpha \cdot A^\alpha(\sigma)}{OPT^1(\sigma)} \leq \frac{\alpha \cdot A^\alpha(\sigma)}{OPT^\alpha(\sigma)} \leq \alpha \cdot c$$

Lemma 1 *If c is a lower bound on the competitive ratio in classical paging then c/α is a lower bound on the competitive ratio in α-paging.*

We conclude the section with the observation that similarly to the case of classical paging, we can restrict our attention to *lazy* algorithms. We call an α-paging algorithm *lazy*, if it performs an eviction only when the memory is full, and a page fault occurs; in this case it evicts at most α pages. The proof of Lemma 2 is a step-by-step modification of a general algorithm into a lazy one, and it can be found in Appendix A.

Lemma 2 *For any α-paging algorithm A, a lazy algorithm B exists such that $B(\sigma) \leq A(\sigma)$ for every input sequence σ.*

3 Lower bounds

Recall that, for the competitive ratio of online paging algorithms, lower bounds of k and H_k were given in the deterministic [18] and in the randomized [10] settings, respectively. In what follows, we generalize these bounds for α-paging

algorithms. For deterministic α-paging, the result in Corollary 1 follows immediately from Lemma 1.

Corollary 1 *Every deterministic online algorithm for α-paging has a competitive ratio of at least k/α.*

For randomized α-paging, using the result in Lemma 1 yields a lower bound of H_k/α. In Lemma 3 this bound is significantly improved.

Lemma 3 *Every randomized online α-paging algorithm has a competitive ratio of at least $(H_{k+\alpha-1} - H_{\alpha-1})/(H_{2\alpha-1} - H_{\alpha-1})$.*

Proof. Let c_r be the claimed lower bound. By Yao's minimax principle for cost minimization problems [7], it suffices to prove that there exists a set of request sequences and a probability distribution over these inputs, such that the expected cost of any deterministic online algorithm is at least c_r times more than the expected cost of an optimal offline algorithm. Further, the offline cost has to be unbounded. We consider input sequences which first request pages $(1, \ldots, k+1)$ followed by n requests to pages in $\{1, \ldots, k+\alpha\}$, drawn uniformly at random.

We first analyze the performance of a deterministic online algorithm A, which evicts only groups of α pages. We consider the page requests that cause A to perform evictions, and calculate the expected number l_A of requests between two evictions. This happens after requesting α pages from the $2\alpha-1$ pages that are not in the memory. Having requested $i-1$ such pages, the probability to request the i'th is $(2\alpha - i)/(k+\alpha)$, and therefore $l_A = \sum_{i=1}^{\alpha}(k+\alpha)/(2\alpha - i) = (k+\alpha)(H_{2\alpha-1} - H_{\alpha-1})$. If A evicts groups of less than α pages then the value of l_A decreases. We obtain that the expected amount of evictions performed by any arbitrary online algorithm is at most n/l_A.

We now describe an offline algorithm S. We split the input in consecutive intervals $(I_0, \ldots, I_l)$, each of them, except for I_l, maximized (in the given order) with respect to the property of containing k pairwise distinct pages. If the first eviction in an interval I_j occurs, S evicts α pages including all pages not requested in I_j and thus no further eviction is needed in I_j.

Having already requested $i-1$ distinct pages in an interval, the probability that some requested page is the i'th distinct page in the specified interval is $(k+\alpha-i+1)/(k+\alpha)$. We obtain that the expected length l_S of an interval I_j, with $1 \leq j < l$, is $l_S = (\sum_{i=1}^{k+1}(k+\alpha)/(k+\alpha-i+1)) - 1$, which sums to $l_S = (k+\alpha)(H_{k+\alpha-1} - H_{\alpha-1})$. Thus, we have that S performs expected n/l_S evictions. By standard arguments, an optimal offline algorithm performs at least $0.5n/l_S$ evictions which implies that the optimal cost is unbounded. The quotient $(n/l_A)/(n/l_S)$ solves to the claimed lower bound.

□

4 Deterministic α-paging

In this section we discuss deterministic α-paging algorithms. We give in Lemma 4 a lower bound on the number of evictions done by any offline algorithm.

Lemma 4 *Consider an arbitrary input sequence σ that we split into intervals $I_0, \ldots, I_l$, so that I_j contains k pairwise distinct pages and is maximal with respect to this property, for all $j = 0, \ldots, l-1$. Then any offline algorithm performs at least l evictions.*

Proof. Let A be an algorithm and $I_{j_1}, \ldots, I_{j_n}$ be all intervals where A performs no eviction. We first prove that for each pair $(I_{j_i}, I_{j_{i+1}})$ there exists an interval $I_{x'}$, with $j_i < x' < j_{i+1}$, such that A performs at least two evictions while processing $I_{x'}$. Assume that there exists some pair $(I_{j_i}, I_{j_{i+1}})$, such that each interval I_x performs one eviction, for all x, with $j_i < x < j_{i+1}$. Since I_{j_i} does no eviction, after its processing the memory is full and contains all pages requested in I_{j_i}. If some interval I_x starts with a full memory containing all pages in I_{x-1}, then the first page in I_x triggers an eviction, since by definition it is not requested in I_{x-1}. If there occurs no other eviction in I_x, after processing I_x the memory is full and contains all pages requested in I_x, since I_x contains k pairwise distinct pages. Therefore, if all I_x, with $j_i < x < j_{i+1}$, perform only one eviction, then the first request in $I_{j_{i+1}}$ causes an eviction, which is a contradiction. Therefore, between two intervals where A performs no evictions there exists one interval where it performs at least two evictions and this concludes the proof. □

We propose an adaptation of the optimal offline algorithm MIN [6] from the classical paging, that we denote α-MIN, and prove that it achieves optimality also in α-paging. Upon a page request that is not in the memory, the MIN algorithm evicts the page whose first request occurs furthest away in the future. Similarly, upon a page fault when the memory is full, α-MIN evicts the α pages whose first requests occur furthest away in the future.

Lemma 5 *The α-MIN algorithm is optimal for α-paging.*

The proof of Lemma 5 is included in Appendix A. Like in the classic case, we modify an optimal algorithm step by step to eventually obtain α-MIN. Similarly to MIN, we adapt the classical LRU to the α-paging setting and obtain α-LRU which, when the memory is full and the requested page is not in memory, evicts the α least recently requested pages from the memory. We show that this algorithm achieves a competitive ratio of k/α, which is optimal.

Lemma 6 *α-LRU is k/α-competitive.*

Proof. We split the input sequence into consecutive intervals $(I_0, \ldots, I_l)$, each of them being maximal in requesting k pairwise distinct pages, except for I_l. We first prove that for each I_j, with $1 \leq j < l$, α-LRU performs at most k/α evictions. The first $k - \alpha$ pairwise distinct page requests do not evict any page previously requested in I_j, since there exist α pages requested less

recently. These $k-\alpha$ pages cause at most $k/\alpha-1$ evictions. If the remaining α pairwise distinct pages cause an eviction, then the memory contains only pages requested in I_j after this eviction, and thus no further eviction is possible in I_j. We conclude that while processing I_j α-LRU performs at most k/α evictions. Since by Lemma 4 any optimal offline algorithm performs amortized one eviction for I_j, we conclude that α-LRU is k/α-competitive. □

Essentially, the improvement of the competitive ratio by a factor of α (compared to classical paging) is due to the fact that for worst case inputs α-LRU takes full advantage of the possibility of evicting α pages at the cost of one, while the optimal offline algorithm has basically no gain. We note that this observation does not hold for every input sequence.

5 Randomized α-paging

We introduce an adaptation of the classical Mark algorithm, denoted α-Mark. Similarly to Mark, α-Mark keeps track of an interval splitting where each interval consists of exactly k pairwise distinct page requests. Additionally, α-Mark assigns priorities to pages, and pages are evicted based on these priorities.

α-Mark. Each page p is marked upon request. Assume that p causes a page fault, and the memory is full, containing x unmarked pages. If $x \geq 1$, then α-Mark evicts the subset of $\min\{x, \alpha\}$ unmarked pages with lowest priorities. In case $x = 0$, all pages get unmarked and are assigned k pairwise different priorities uniformly at random before choosing the pages to be evicted.

The analysis is based on interval splitting, where an interval ends just before all pages get unmarked. The key difference in analyzing the performance of Mark and α-Mark in such an interval is that the probability that some page p causes a page fault is determined solely by the input for classical Mark, whereas in the case of α-Mark it depends also on the random decisions before the request of p. Additionally, page faults can increase and decrease the probability of future page faults in an interval, such that an important simplifying assumption about the structure of the intervals (see Lemma 7) is not obvious like for classical Mark. To prove that this assumption holds also for α-Mark we use priorities instead of choosing a set of unmarked pages uniformly at random to be evicted. The probability for a page fault can be bounded independent on the previous random decisions, which simplifies the analysis. However, this leads to a competitive ratio of about $3.5(H_k - H_\alpha)$, whereas the improved analysis, as presented here, yields a competitive ratio of about $2.46(H_k - H_{2\alpha-1})$, which is 1.7 times the lower bound for large values of k and α.

Intervals. Suppose that the algorithm splits the input into the consecutive intervals $I_1, \cdots, I_l$, each containing k pairwise distinct pages, maximal with respect to this property. We call the pages which are requested in I_j but not in I_{j-1} *new pages* and denote their number by n_j. Pages which are requested

in I_{j-1} are called *old pages*. In I_j exactly $o_j = k - n_j$ old pages are requested. When I_j starts, the memory contains all old pages, all of them are unmarked and have distinct priorities.

Lemma 7 *Let n_j be the number of pairwise distinct new pages requested in I_j. The expected number of evictions done by α-Mark in I_j does not decrease if we assume that:*

- *every page is requested only once in I_j;*
- *all n_j new pages are requested before all the old pages.*

Proof. When a page is requested the second time in an interval, it is a marked page in the memory. Removing the second request of it neither changes the number of evictions nor changes the state of the algorithm. For the rest of the proof we assume that each page is requested only once. In order to prove the second assumption we fix the priorities assigned to old pages at the beginning of each interval and obtain a deterministic algorithm D. We show that D maximizes its cost if all new pages are requested before the old pages in I_j. Let p_n be the last requested new page in I_j and p_o the last requested old page before the request of p_n. We construct σ' from σ by switching the requests of p_n and p_o in I_j and show that $D(\sigma) \leq D(\sigma')$. The cost of D changes iff p_o causes a page fault only for the request sequence σ'. Fix the scenarios S immediately after the request of p_n in σ, and S' just before the request of p_o in σ'. If τ and τ' are the corresponding subsequences of remaining pages to be requested in I_j we have $\tau' = p_o\tau$ where both contain only requests of old pages. Since before both S and S' the same amount of page faults occurred, priorities are fixed, and p_o induces a page fault in S' the following holds: If page q (requested in τ) triggers the next eviction, then there exists another page q' requested before q in τ' such that q' triggers an eviction in S'. Inductively argument it can be proven that each further eviction performed in S can be injectively mapped to a further eviction in S'.

□

Expected cost of α-Mark. For some input σ, we bound the expected number of evictions done by α-Mark in an interval I_j, $(j \geq 1)$.

Lemma 8 *Consider an interval I_j, $j \geq 1$, in which n_j new pages are requested, and let $m_j = \lceil n_j/\alpha \rceil$. The expected number of evictions done by α-Mark in I_j is at most*

$$m_j + 1 + \frac{H_k - H_{n_j-1}}{H_{n_j+\alpha-1} - H_{n_j-1}} .$$

Proof. By Lemma 7, we assume that I_j consists of n_j requests of new pages followed by $o_j = k - n_j$ old page requests, and every page is requested only once in I_j. The new pages cause m_j evictions and all further evictions are caused by old pages.

We denote by $S(h)$ a state that occurs immediately after an eviction caused by an old page, where h is the number of old pages not yet requested (including the n_j old pages which are not requested at all) in I_j. Let y be the number of old pages not in memory for some $S(h)$. We have that $y = n_j + \alpha - 1$, since the memory contains n_j new pages, and $\alpha - 1$ free slots (as well as $k - h$ marked, and $h - n_j - \alpha + 1$ unmarked old pages).

For $S(h)$, let $(p_1, p_2, \ldots, p_h)$ be the old pages not yet requested in the order they appear in I_j; the pages not requested at all in I_j are at the end in arbitrary order. If p_i causes the next eviction then $S(h-i)$ occurs immediately after processing p_i. Let $E(h)$ be the expected number of evictions from state $S(h)$ until the end of I_j, and let $pb(i)$ be the probability that page p_i causes the next eviction. We have that:

$$n_j \leq h \leq n_j + \alpha - 1 : \quad E(h) = 0 \ , \tag{1}$$

$$h > n_j + \alpha - 1 : \quad E(h) = \sum_{i=\alpha}^{h-n_j} pb(i) \cdot (1 + E(h-i)) \ . \tag{2}$$

The base case is correct because there are $\alpha - 1$ empty locations in the memory and at most $\alpha - 1$ pages are requested until the end of I_j. For the recursive part we note that pages p_i, with $i < \alpha$, cannot cause an eviction because $\alpha - 1$ page faults must fill the memory before an eviction could take place. On the other hand, pages p_i, with $i > h - n_j$, are not requested at all in I_j and thus cannot cause evictions.

The next eviction is caused by p_i iff the y old pages not in memory at $S(h)$ contain p_i, moreover exactly $\alpha - 1$ pages from $(p_1, p_2, \ldots, p_{i-1})$, and $y - \alpha$ pages from $(p_{i+1}, \ldots, p_h)$. The total amount of subsets of size y chosen from $(p_1, \ldots, p_h)$ with this property is $\binom{i-1}{\alpha-1} \cdot \binom{1}{1} \cdot \binom{h-i}{y-\alpha}$. Since all not yet requested pages have the same probability not to reside in memory, we obtain:

$$pb(i) = \frac{\binom{i-1}{\alpha-1} \cdot \binom{h-i}{y-\alpha}}{\binom{h}{y}} \ .$$

For $h > n_j + \alpha - 1$, using $y = n_j + \alpha - 1$ and $\sum pb(i) \leq 1$, we obtain:

$$E(h) \leq 1 + \frac{1}{\binom{h}{n_j+\alpha-1}} \sum_{i=\alpha}^{h-n_j} \binom{i-1}{\alpha-1} \cdot \binom{h-i}{n_j-1} \cdot E(h-i) \ .$$

In Appendix B, Theorem 2 we prove that the function $f(h)$, defined as

$$f(h) = \frac{H_h - H_{n_j-1}}{H_{n_j+\alpha-1} - H_{n_j-1}} \ ,$$

satisfies nearly the same recurrence:

$$f(h) = 1 + \frac{1}{\binom{h}{n_j+\alpha-1}} \sum_{i=\alpha}^{h-n_j+1} \binom{i-1}{\alpha-1} \cdot \binom{h-i}{n_j-1} \cdot f(h-i) \ .$$

We prove by induction for all h, $n_j \le h \le k-1$ that $f(h) \ge E(h)$ holds. In the base case, for $n_j \le h \le n_j + \alpha - 1$ we have $E(h) = 0$, and since $f(h)$ is nonnegative the assumption follows. Assume now that the assumption is true for $n_j, n_j+1, \ldots, h$, where $h \ge n_j + \alpha - 1$. Using the recursive identity of $f(h)$, we have:

$$\begin{aligned} f(h+1) &= 1 + \frac{1}{\binom{h+1}{n_j+\alpha-1}} \sum_{i=\alpha}^{h+1-n_j+1} \binom{i-1}{\alpha-1} \cdot \binom{h+1-i}{n_j-1} \cdot f(h+1-i) \\ &\ge 1 + \frac{1}{\binom{h+1}{n_j+\alpha-1}} \sum_{i=\alpha}^{h+1-n_j} \binom{i-1}{\alpha-1} \cdot \binom{h+1-i}{n_j-1} \cdot f(h+1-i) \\ &\ge 1 + \frac{1}{\binom{h+1}{n_j+\alpha-1}} \sum_{i=\alpha}^{h+1-n_j} \binom{i-1}{\alpha-1} \cdot \binom{h+1-i}{n_j-1} \cdot E(h+1-i) \\ &\ge E(h+1). \end{aligned}$$

The first inequality results from removing the last (nonnegative) term of the sum and the second follows from the induction hypothesis. Since the new pages in I_j cause m_j evictions, and the expected number of evictions caused by old pages is at most $1 + f(k-1) \le 1 + f(k)$, the proof concludes. □

Competitive ratio. Let σ be the input sequence, $(I_0, I_1, \ldots, I_l)$ the interval splitting, and consider n_j, the number of new pages requested in some interval I_j ($j \ge 1$), with $m_j = \lceil n_j/\alpha \rceil$.

Lemma 9 *Let $OPT(\sigma)$ be the number of evictions done by an optimal offline algorithm when processing σ. Then, $OPT(\sigma) \ge \max\left\{\frac{1}{2}\sum_{j=1}^{l} m_j, l\right\}$.*

Proof. By Lemma 4, OPT does at least l evictions. For some $j > 1$, the intervals I_j and I_{j-1} contain $k + n_j$ pairwise distinct pages, thus at least n_j of them are filled in memory slots resulted by evictions *during* I_{j-1} and I_j. Therefore, the number of evictions in these two intervals is at least m_j. We get the following:

$$OPT(\sigma) \ge \sum_{j \text{ odd}, j \ge 1} m_j, \qquad OPT(\sigma) \ge \sum_{j \text{ even}, j \ge 1} m_j \ .$$

Since $\max\{a, b\} \ge \frac{1}{2}(a+b)$, we obtain that $OPT(\sigma) \ge \frac{1}{2}\sum_{j \ge 1} m_j$.

□

Theorem 1 *The competitive ratio of α-Mark is at most $3 + \frac{H_k - H_{2\alpha-1}}{H_{3\alpha-1} - H_{2\alpha-1}}$.*

Proof. Let M_j be the upper bound from Lemma 8 on the expected number of evictions done by α-Mark in interval I_j:

$$M_j = m_j + 1 + F(n_j), \qquad F(n_j) = \frac{H_k - H_{n_j-1}}{H_{n_j+\alpha-1} - H_{n_j-1}} \ .$$

We can assume w.l.o.g., that $n_j = m_j \cdot \alpha$, since it has no influence on the lower bound for OPT in Lemma 9, while in the upper bound for α-Mark M_j is increasing in n_j.

If $\frac{1}{2} \cdot \sum_{j=1}^{l} m_j < l$, then when incrementing some m_j the sum $\sum M_j$ increases. However, this does not affect the lower bound of OPT and thus we can assume that $\sum_{j=1}^{l} m_j \geq 2l$. Let $m = \sum_{j=1}^{l} m_j$. For fixed $m \geq 2l$, if the upper bound on the number of evictions done by α-Mark is maximal, then $m_j \geq 2$ for all j (see Appendix C, Lemma 7). Denoting C the competitive ratio of α-Mark, we have:

$$C \leq \frac{\sum_{j=1}^{l} M_j}{\frac{1}{2}\sum_{j=1}^{l} m_j} \leq 3 + 2\frac{\sum_{j=1}^{l} F(m_j \cdot \alpha)}{\sum_{j=1}^{l} m_j} \leq 3 + 2 \max_j \left\{ \frac{F(m_j \cdot \alpha)}{m_j} \right\} .$$

Since $\frac{F(x \cdot \alpha)}{x}$ is decreasing in x, and $m_j \geq 2$, we get $C \leq 3 + \frac{H_k - H_{2\alpha-1}}{H_{3\alpha-1} - H_{2\alpha-1}}$. □

By Lemma 3, for large enough k and α, approximating $H_x \approx \ln x + \gamma$, $\gamma \approx 0.57$, any randomized algorithm has a competitive ratio $C_{lb} \geq \ln(k/\alpha+1)/\ln(2)$. Similarly, the competitive ratio of α-Mark satisfies $C \leq \ln(k/\alpha)/\ln(1.5)$, and thus the gap between the upper bound and the lower bound is approximately $\ln 2/\ln(1.5) \approx 1.7$, as opposed to a tight factor of 2 in the case of classical Mark.

References

[1] D. Achlioptas, M. Chrobak, and J. Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234(1-2):203–218, 2000.

[2] D. Ajwani, A. Beckmann, R. Jacob, U. Meyer, and G. Moruz. On computational models for flash memory devices. In *Proc. 8th International Symposium on Experimental Algorithms*, pages 16–27, 2009.

[3] D. Ajwani, I. Malinger, U. Meyer, and S. Toledo. Characterizing the performance of flash memory storage devices and its impact on algorithm design. In *Proc. 7th International Workshop on Experimental Algorithms*, pages 208–219, 2008.

[4] S. Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1–2):3–26, 2003.

[5] J. Barnat, L. Brim, S. Edelkamp, D. Sulewski, and P. Šimeček. Can flash memory help in model checking? In *Proc. 13th International Workshop on Formal Methods for Industrial Critical Systems*, pages 159–174, 2008.

[6] L. A. Belady. A study of replacement algorithms for virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[7] A. Borodin and R. El-Yaniv. *Online computation and competitive anlysis*. Cambridge University Press, 1998.

[8] L. Bouganim, B. P. Jónsson, and P. Bonnet. uFLIP: Understanding Flash IO Patterns. In *Proc. 4th biennial conference on innovative data systems (CIDR)*, 2009.

[9] EasyCo. Managed flash technology. `http://www.easyco.com/mft/`.

[10] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

[11] E. Gal and S. Toledo. Algorithms and data structures for flash memories. *ACM Computing Surveys*, 37(2):138–163, 2005.

[12] A. V. Goldberg and R. Werneck. Computing point-to-point shortest paths from external memory. In *Proc. 7th Workshop on Algorithm Engineering and Experiments*, pages 26–40, 2005.

[13] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete mathematics*. Addison-Wesley, 1994.

[14] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.

[15] Y. Li, B. He, Q. Luo, and K. Yi. Tree indexing on flash disks. In *Proc. 25th International Conference on Data Engineering*, 2009. 1303-1306.

[16] L. A. McGeoch and D. D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

[17] P. Sanders, D. Schultes, and C. Vetter. Mobile route planning. In *Proc. 16th Annual European Symposium on Algorithms*, pages 732–743, 2008.

[18] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[19] C.-H. Wu, L.-P. Chang, and T.-W. Kuo. An efficient R-tree implementation over flash-memory storage systems. In *Proc. 11th ACM International Symposium on Advances in Geographic Information Systems*, pages 17–24, 2003.

[20] C.-H. Wu, T.-W. Kuo, and L.-P. Chang. An efficient B-tree layer implementation for flash-memory storage systems. *ACM Transactions on Embedded Computing Systems*, 6(3), 2007.

A Laziness and α-MIN proof

Notation. In the sequel, let $\sigma(i,j)$ denote the subsequence $(p_i, p_{i+1}, \ldots, p_j)$ of an input $\sigma = (p_1, \ldots, p_n)$, and $M_j(Alg)$, stand for the set of pages in the memory after step j of an algorithm Alg (for a given σ).

By a *general* α-paging algorithm we mean an algorithm where after any page request p, an arbitrary number e of pages can be evicted from memory at the cost of $\lceil e/\alpha \rceil$, and then some pages that fit into memory are loaded, including p whenever p caused a page fault.

Next, we prove Lemma 2, that will be used directly in the proof of Lemma 5.

Lemma 2 *For any general α-paging algorithm A, a lazy algorithm B exists such that $B(\sigma) \leq A(\sigma)$ holds for every input sequence σ.*

Proof. We modify the non-lazy actions of A step by step, and obtain the algorithms $A = B_0, B_1, \ldots, B_i, \ldots$, so that for every $i \in \mathbb{N}$, and every input σ of length $n \geq i$

(i) B_i performs only lazy actions in steps $1 \ldots i$;

(ii) B_i coincides with B_{i-1} on $\sigma(1, i-1)$;

(iii) if $n > i$ then $M_{i+1}(A) = M_{i+1}(B_i)$ and B_i coincides with A on $\sigma(i+2, n)$;

(iv) $B_i(\sigma) \leq A(\sigma)$;

(v) if A is an online algorithm then B_i is an online algorithm as well.

Having defined the sequence $B_1, B_2, \ldots$, we define the ith step of the lazy algorithm B on some input σ to be the ith step of B_i on the same input. Now if σ has length n (where we do not need to know n in advance!), then B (is well-defined and) coincides with B_n for this input, by (ii). Thus, B is lazy, $B(\sigma) \leq A(\sigma)$ for every σ, and B is online if A is online.

It remains to define B_i, for given A and B_{i-1}. Fix an arbitrary input sequence σ. Intuitively, B_i postpones to step $i+1$ all 'unnecessary' evictions and loadings made by A until step i. Notice that we need to specify only the steps i and $i+1$ of B_i conforming to the requirements. Assume $M_{i-1}(B_i) = M_{i-1}(B_{i-1})$ and page p_i is requested. If p_i causes a page fault, and the memory is full, then B_{i-1} evicts a set $E \neq \emptyset$ of pages. Let B_i evict a page set E_i so that $E_i \subset E$ and $|E_i| = \alpha$ if $|E| > \alpha$, and $E_i = E$ if $|E| \leq \alpha$. Finally, B_i loads page p_i. If, on the other hand, in step i no eviction is *necessary*, then the lazy action of B_i is trivial (load p_i if needed).

The (non-lazy) eviction and loading in step $i+1$ is then determined by $M_{i+1}(B_i) = M_{i+1}(A)$. Namely, let $E_{i+1} = M_i(B_i) \setminus M_{i+1}(A)$, and the set of pages loaded by B_i be $L_{i+1} = M_{i+1}(A) \setminus M_i(B_i)$. This is clearly doable as for memory space, and it loads the page p_{i+1} when needed.

Next we show that the cost of B_i in steps $(i, i+1)$ does not exceed that of B_{i-1}. This, in turn, will prove $B_i(\sigma) \leq B_{i-1}(\sigma)$, since B_i and B_{i-1} coincide on

the rest of the input sequence. In particular, by the definition of B_{i-1} we have $M_{i+1}(B_{i-1}) = M_{i+1}(A)$.

If B_i made no eviction in step i, then B_i has a cost of $\lceil |E_{i+1}|/\alpha \rceil$, and B_{i-1} has at least the same cost, since altogether it evicted at least the same set of pages during the steps i and $i+1$. If B_i made an eviction at step i, whereas B_{i-1} evicted the set E with $|E| \leq \alpha$, then both incur a cost of 1 at step i, moreover $M_i(B_i) \subseteq M_i(B_{i+1})$ implies that B_i has no larger cost at step $i+1$ than B_{i-1}. Finally, if $|E| > \alpha$, then we have $E = E_i \cup E'$, and $\lceil |E|/\alpha \rceil = 1 + \lceil |E'|/\alpha \rceil$ for some E', and a cost of 1 for B_i. Let E'' be the set of pages evicted by B_{i-1} at step $i+1$. It is easy to see that $E_{i+1} \subseteq E' \cup E''$, so the cost of B_i in the two steps is not more than $1 + \lceil |E'|/\alpha \rceil + \lceil |E''|/\alpha \rceil$, i.e., the cost of B_{i-1}.

We conclude the proof with the observation that for any $j \in \mathbb{N}$, the jth step of B_i depends only on the first j steps of A, which implies (v).

□

Lemma 4 *The α-MIN algorithm is optimal for α-paging.*

Proof. We fix an input sequence σ of page requests. We prove that any algorithm A makes at least as many evictions on input σ as α-MIN does. By Lemma 2, we can restrict the discussion to lazy algorithms.

We consider an A s.t. for σ the number of evictions $A(\sigma)$ is minimum; moreover, among these algorithms, A has the maximum $l \leq n$ such that on $\sigma(1, l)$, A performs the same steps (i.e., same evictions) as α-MIN. Assume by contradiction that $l < n$, meaning that $A \neq \alpha$-MIN. Observe that after step l of both algorithms, the memory contains exactly the same pages, and in step $l+1$ both A and α-MIN have to do an eviction, in which they evict distinct page sets. We assume, finally, that A is selected so that the symmetric difference of these two evicted sets is the smallest possible.

In what follows, we define an algorithm B (for input σ) that carries out the same steps on $\sigma(1, l)$, as A (and α-MIN), and in step $l+1$ the symmetric difference of the evicted pages of B and of α-MIN is strictly smaller than that of A and α-MIN. In addition, $|B(\sigma)| \leq |A(\sigma)|$. This will contradict the above definition of A being optimal on σ and 'most similar' to α-MIN.

For any algorithm Alg processing σ, let $E_j(Alg)$ be the sets of $\leq \alpha$ evicted pages in step j, (given that an eviction took place in step j), and let $Alg(j)$ denote the number of evictions before step $j+1$. Since A differs from α-MIN in step $l+1$, either A evicts a subset of the pages evicted by α-MIN (in this case a trivial modification of A yields B), or there exist $i > h > l$, and $\{p, q\} \subset M_l(A)$ such that $\sigma_i = p$ is the first request of page p and $\sigma_h = q$ is the first request of page q after l; furthermore, $q \in E_{l+1}(A) \setminus E_{l+1}(\alpha\text{-MIN})$ and $p \in E_{l+1}(\alpha\text{-MIN}) \setminus E_{l+1}(A)$.

Since B performs the same steps as A on $\sigma(1, l)$, it also must evict α pages, after σ_{l+1} was requested. Let $E_{l+1}(B) = E_{l+1}(A) \setminus \{q\} \cup \{p\}$ (note that so B reduces the symmetric difference as intended). In the input subsequence $\sigma(l+2, h-1)$ the pages p and q are not requested, and B makes the same evictions, in the same steps, as A does, except that in case A evicts p, then B

evicts q instead of p. If the latter occurs, then B and A can coincide on the rest of the sequence, and we are done. Otherwise, before the request $\sigma_h = q$, it still holds that $M_{h-1}(B) = M_{h-1}(A) \setminus \{p\} \cup \{q\}$. Thus, $\sigma_h = q$ causes *no* page fault for B. After step h, it is straightforward to define the lazy actions of B so that after each step $h \leq j < i$, either $B(j) < A(j)$, and $M_j(A) \subseteq M_j(B) \cup \{p\}$, or $B(j) = A(j)$ and $M_j(A) = M_j(B) \cup \{p\}$ holds. The difference of $M_j(A)$ and $M_j(B)$ can be resolved when either p is requested in step i, or p is evicted by A before step i. For the rest of the sequence B coincides with A.

□

B Expected cost of α-Mark

Theorem 2 *For all integers h, n, α, with the properties that $h > n + \alpha - 1$ and $n, \alpha > 0$, the following equality holds:*

$$\frac{H_h - H_{n-1}}{H_{n+\alpha-1} - H_{n-1}} = 1 + \frac{1}{\binom{h}{n+\alpha-1}} \sum_{i=\alpha}^{h-n+1} \binom{i-1}{\alpha-1}\binom{h-i}{n-1} \frac{H_{h-i} - H_{n-1}}{H_{n+\alpha-1} - H_{n-1}}$$

Definition 1 *Let $d : \mathbb{N} \to \mathbb{R}$ be an arbitrary function. For $z, x \in \mathbb{N}$ and $z > x$ we recursively define $f_x(z)$:*

$$f_0(z) = d(z), \qquad \forall x > 0 : \quad f_x(z) = \sum_{i=x}^{z-1} f_{x-1}(i)$$

The next Lemma gives us an recursive identity for $f_x(z)$, for any function $d(z)$:

Lemma 5

$$\forall x > 0 : \qquad f_x(z) = \sum_{i=1}^{z-x} \binom{z-i-1}{x-1} d(i)$$

Proof. We prove the equality by induction on x. For $x = 1$ we get by definition:

$$f_1(z) = d(1) + ... + d(z-1) = \sum_{i=1}^{z-1} \binom{z-i-1}{0} d(i)$$

We assume that the formula is correct for $1, \ldots, x-1$ and get:

$$f_x(z) = \sum_{l=x}^{z-1} f_{x-1}(l) = \sum_{l=x}^{z-1} \sum_{i=1}^{l-x+1} \binom{l-i-1}{x-2} d(i) = \sum_{i=1}^{z-x} \left(d(i) \sum_{l=x+i-1}^{z-1} \binom{l-i-1}{x-2} \right)$$

$$= \sum_{i=1}^{z-x} \left(d(i) \sum_{l=x-2}^{z-i-2} \binom{l}{x-2} \right) = \sum_{i=1}^{z-x} \binom{z-i-1}{x-1} d(i)$$

□

Definition 2

$$d(i) = \binom{i}{n-1} \cdot H_i,$$

where $n > 0$ is the fixed parameter defined in Theorem 2.

We get by the next lemma a second identity of $f_x(z)$ using the properties of the function $d(i)$ defined before.

Lemma 6

$$\forall z, x \geq 0, \quad z > n+x-1 : \qquad f_x(z) = \binom{z}{n+x-1} (H_z - (H_{n+x-1} - H_{n-1}))$$

Proof. We prove the lemma by induction on x. For $x = 0$ we have:

$$f_0(z) = \binom{z}{n-1} (H_z) = d(z)$$

Now assume that the equation is true for $0, \ldots, x-1$:

$$f_x(z) = \sum_{i=x}^{z-1} f_{x-1}(i) = \sum_{i=x}^{z-1} \binom{i}{n+x-2} (H_i - (H_{n+x-2} - H_{n-1}))$$

$$= \sum_{i=0}^{z-1} \binom{i}{n+x-2} (H_i - (H_{n+x-2} - H_{n-1}))$$

In the case that $n = 1$ we added a term with $i = x-1$ where the binomial is nonzero, but the second term in the sum is $H_{x-1} - H_{x-1} + H_0 = 0$. Using the identity $\sum_{i=0}^{z-1} \binom{i}{x} H_i = \binom{z}{x+1} \left(H_z - \frac{1}{x+1}\right)$ from [13] we get:

$$f_x(z) = \binom{z}{n+x-1} (H_{n+x-2} - H_{n-1}) + \binom{z}{n+x-1} \left(H_z - \frac{1}{n+x-1}\right)$$

This proves the lemma since $H_{n+x-2} + \frac{1}{n+x-1} = H_{n+x-1}$ □

Proof. For the proof of Theorem 2 we use the two identities from the previous Lemmas for $f_\alpha(h)$.

$$f_\alpha(h) = \binom{h}{h+n-1} (H_h - (H_{n+\alpha-1} - H_{n-1})) = \sum_{i=1}^{h-\alpha} \binom{h-i-1}{h-1} \binom{i}{n-1} \cdot H_i$$

$$H_h - H_{n-1} = H_{n+\alpha-1} - H_{n-1} - H_{n-1} + \frac{1}{\binom{h}{h+n-1}} \sum_{i=1}^{h-\alpha} \binom{h-i-1}{h-1} \binom{i}{n-1} \cdot H_i$$

$$H_h - H_{n-1} = H_{n+\alpha-1} - H_{n-1} + \frac{1}{\binom{h}{h+n-1}} \sum_{i=\alpha-1}^{h-\alpha} \binom{h-i-1}{\alpha-1} \binom{i}{n-1} \cdot (H_i - H_{n-1})$$

$$\frac{H_h - H_{n-1}}{H_{n+\alpha-1} - H_{n-1}} = 1 + \frac{1}{\binom{h}{h+n-1}} \sum_{i=\alpha-1}^{h-\alpha} \binom{h-i-1}{\alpha-1}\binom{i}{n-1} \cdot \frac{(H_i - H_{n-1})}{H_{n+\alpha-1} - H_{n-1}}$$

$$\frac{H_h - H_{n-1}}{H_{n+\alpha-1} - H_{n-1}} = 1 + \frac{1}{\binom{h}{h+n-1}} \sum_{i=\alpha}^{h-\alpha+1} \binom{i-1}{\alpha-1}\binom{h-i}{n-1} \cdot \frac{H_{h-i} - H_{n-1}}{H_{n+\alpha-1} - H_{n-1}}$$

□

C Competitive ratio of α-Mark

Consider the functions $F(x) = H_k - H_{x\alpha-1}$, $D(x) = H_{(x+1)\alpha-1} - H_{x\alpha-1}$, and $M(x) = x + 2 + F(x)/D(x)$.

Lemma 7 *Given integers m, l, with $(m \geq 2l)$, consider $m_1, \ldots, m_l$ positive integers such that $\sum_{j=1}^{l} m_j = m$. If $\sum_{j=1}^{l} M(m_j)$ is maximal under the constraint $m_j \geq 1$, then $m_j \geq 2$, $\forall j \geq 1$.*

Proof. Assume there exists some $m_u = 1$. This implies that there exists some m_v with $m_v > 2$. We prove that $\sum_{j=1}^{l} M(m_j)$ is not maximal.

$$\sum_{j=1}^{l} M(m_j) = \sum_{j=1}^{l} m_j + 2 + F(m_j)/D(m_j) = m + 2l + \sum_{j=1}^{l} F(m_j)/D(m_j)$$

Setting $m_u = m_u + 1$ and $m_v = m_v - 1$, we obtain that $\sum_{j=1}^{l} M(m_j)$ increases by $\delta = \left(\frac{F(2)}{D(2)} - \frac{F(1)}{D(1)}\right) - \left(\frac{F(m_v)}{D(m_v)} - \frac{F(m_v-1)}{D(m_v-1)}\right)$. By Lemma 8, we have that $\delta > 0$ for $m_v \geq 3$, which contradicts the maximality of $\sum_{j=1}^{l} M(m_j)$.

□

Lemma 8 *For $x > 1$, the following holds:*

$$\frac{F(x+1)}{D(x+1)} - \frac{F(x)}{D(x)} < \frac{F(x)}{D(x)} - \frac{F(x-1)}{D(x-1)}$$

We first prove that, if we have $1/D(x-1) + 1/D(x+1) \leq 2/D(x)$, the result holds, and then we prove this inequality.

Assume $1/D(x-1) + 1/D(x+1) \leq 2/D(x)$. We obtain immediately:

$$F(x)\left(\frac{1}{D(x+1)} + \frac{1}{D(x-1)}\right) \leq \frac{2F(x)}{D(x)}\,. \tag{3}$$

We note that $D(x) > D(x+1)$, for $x \geq 1$, which yields $1 - D(x)/D(x+1) < 0$. Writing $D(x-1)/D(x-1) - D(x)/D(x+1) < 0$ and summing to (3), we obtain:

$$\frac{F(x) - D(x)}{D(x+1)} + \frac{F(x) + D(x-1)}{D(x-1)} < \frac{2F(x)}{D(x)}\,.$$

Since $F(x+1) = F(x) - D(x)$ and $F(x-1) = F(x) + D(x-1)$, we get that:

$$\frac{F(x+1)}{D(x+1)} + \frac{F(x-1)}{D(x-1)} < 2\frac{F(x)}{D(x)} ,$$

and the result follows.

It remains to prove that $1/D(x+1) + 1/D(x-1) \leq 2/D(x)$, which is equivalent to $D(x)D(x-1) + D(x)D(x+1) \leq 2D(x-1)D(x+1)$. Denoting $f_x^i = x\alpha + i$, we note that $D(x) = \sum_{i=0}^{\alpha-1} 1/f_x^i$, and thus we need to prove:

$$\sum_{i=0}^{\alpha-1}\sum_{j=0}^{\alpha-1}\left(\underbrace{\frac{1}{f_x^i f_{x-1}^j} + \frac{1}{f_x^i f_{x+1}^j}}_{Fl(i,j)}\right) \leq 2\sum_{i=0}^{\alpha-1}\sum_{j=0}^{\alpha-1}\underbrace{\frac{1}{f_{x-1}^i f_{x+1}^j}}_{Fr(i,j)} .$$

It suffices to prove that $Fl(i,j) + Fl(j,i) \leq 2(Fr(i,j) + Fr(j,i))$, for all (i,j) satisfying $0 \leq i, j < \alpha$. Expanding $Fl(i,j)$ and $Fr(i,j)$, we get:

$$\frac{f_{x+1}^i f_x^j + f_x^i f_{x-1}^j - 2f_x^i f_x^j}{f_{x+1}^i f_{x-1}^j} + \frac{f_{x-1}^i f_x^j + f_x^i f_{x+1}^j - 2f_x^i f_x^j}{f_{x-1}^i f_{x+1}^j} \leq 0 .$$

Doing the calculations, for $E = (x+1)(x-1)\alpha^2 + ij + x\alpha i + x\alpha j$,this is equivalent to:

$$\frac{j-i}{E + \alpha(j-i)} - \frac{j-i}{E - \alpha(j-i)} \leq 0 .$$

Since this inequality is true for all (i,j), with $0 \leq i, j < \alpha$, the proof concludes.