Analyzing Adaptive Cache Replacement Strategies

Mario E. Consuegra², Wendy A. Martinez¹, Giri Narasimhan¹, Raju Rangaswami¹, Leo Shao¹, and Giuseppe Vietri¹

¹School of Computing and Information Sciences, Florida International University, Miami, FL 33199, USA. {walem001,giri,raju,gviet001}@fiu.edu

²Google Inc., Kirkland, WA, USA.

April 25, 2017

Abstract

Adaptive Replacement Cache (ARC) and CLOCK with Adaptive Replacement (CAR) are state-of-theart "adaptive" cache replacement algorithms invented to improve on the shortcomings of classical cache replacement policies such as LRU, LFU and CLOCK. By separating out items that have been accessed only once and items that have been accessed more frequently, both ARC and CAR are able to control the harmful effect of single-access items flooding the cache and pushing out more frequently accessed items. Both ARC and CAR have been shown to outperform their classical and popular counterparts in practice. Both algorithms are complex, yet popular. Even though they can be treated as online algorithms with an "adaptive" twist, a theoretical proof of the competitiveness of ARC and CAR remained unsolved for over a decade. We show that the competitiveness ratio of CAR (and ARC) has a lower bound of N+1 (where N is the size of the cache) and an upper bound of 18N (4N for ARC). If the size of cache offered to ARC or CAR is larger than the one provided to OPT, then we show improved competitiveness ratios. The important implication of the above results are that no "pathological" worst-case request sequences exist that could deteriorate the performance of ARC and CAR by more than a constant factor as compared to LRU.

1 Introduction

Megiddo and Modha [MM03,MM04] engineered an amazing cache replacement algorithm that was self-tuning and called it Adaptive Replacement Cache or ARC. Later, Bansal and Modha [BM04] designed another algorithm called Clock with Adaptive Replacement (CAR). Extensive experimentation suggested that ARC and CAR showed substantial improvements over previously known cache replacement algorithms, including the well-known Least Recently Used or LRU and CLOCK. On the theoretical side, the seminal work of Sleator and Tarjan [ST85] showed that LRU can be analyzed using the theory of online algorithms. They showed that LRU has a competitiveness ratio of N (where N is the size of the cache). More surprisingly, they also showed that with no prefetching, no online algorithm for cache replacement could achieve a competitiveness ratio less than N, suggesting that under this measure, LRU is optimal. In other words, there exist worst-case request sequences that would prevent any algorithm from being better than N-competitive. While these results are significant, they highlight the difference between theory and practice. Sleator and Tarjan's techniques analyze online algorithms in terms of their worst-case behavior (i.e., over all possible inputs), which means that other algorithms with poorer competitiveness ratios could perform better in practice. Another way to state this is that the results assume an oblivious adversary who designs inputs for the online algorithms in a way that make them perform as poorly as possible. The upper bound on performance ratio merely guarantees that no surprises are in store, i.e., there is no input designed by an adversary that can make the algorithm perform poorly.

Given a fixed size cache, the **cache replacement problem** is that of deciding which data item to evict from the cache in order to make room for a newly requested data item with the objective of maximizing cache hits in the future. The cache replacement problem has been referred to as a *fundamental and practically important online problem in computer science* (see Irani and Karlin [Hoc97], Chapter 13) and a "fundamental metaphor in modern computing" [MM04].

The LRU algorithm was considered the most optimal page replacement policy for a long time, but it had the drawback of not being "scan-resistant", i.e., items used only once could pollute the cache and diminish its performance. Furthermore, LRU is difficult to implement efficiently, since moving an accessed item to the front of the queue is an expensive operation, first requiring locating the item, and then requiring data moves that could lead to unacceptable cache contention if it is to be implemented consistently and correctly. The Clock algorithm was invented by Frank Corbató in 1968 as an efficient one-bit approximation to LRU with minimum overhead [Cor68] and continues to be used in MVS, Unix, Linux, and Windows operating systems [Fri99]. Like LRU, Clock is also not scan-resistant because it puts too much emphasis on "recency" of access and pays no attention to "frequency" of access. So there are sequences in which many other algorithms can have significantly less cost than the theoretically optimal LRU. Since then, many other cache replacement strategies have been developed and have been showed to be better than LRU in practice. These are discussed below in Section 2.

An important development in this area was the invention of adaptive algorithms. While regular "online" algorithms are usually designed to respond to input requests in an optimal manner, these self-tuning algorithms are capable of adapting to changes in the request sequence caused by changes in the workloads. Megiddo and Modha's ARC [MM03] is a self-tuning algorithm that is a hybrid of LFU and LRU. Bansal and Modha's CAR is an adaptivehybrid of LFU and CLOCK [BM04]. Experiments show that ARC and CAR outperform LRU and CLOCK for many benchmark data sets [BM04]. Versions of ARC have been deployed in commercial systems such as the IBM DS6000/DS8000, Sun Microsystems's ZFS, and in PostgreSQL.

Unfortunately, no **theoretical analysis** of the adaptive algorithms, \mathbf{A}_{RC} and \mathbf{C}_{AR} , exist in the literature. The main open question that remained unanswered was whether or not there exist "pathological" request sequences that could force \mathbf{A}_{RC} or \mathbf{C}_{AR} to perform poorly. In this document we show that these two algorithms are O(N)-competitive, suggesting that they are not much worse than the optimal \mathbf{L}_{RU} . We also prove a surprising lower bound on the competitiveness that is larger than N.

The main contributions of this paper are as follows:

- 1. For completeness, we provide proofs that LRU and CLOCK are N-competitive.
- 2. We prove a lower bound on the competitiveness of \mathbf{A} RC and \mathbf{C} AR of N+1, proving that there are request sequences where they cannot outperform \mathbf{L} RU and \mathbf{C} LOCK.
- 3. We show that \mathbf{A}_{RC} is 4N-competitive.
- 4. We show that \mathbf{C}_{AR} is 18N-competitive.
- 5. We obtain precise bounds for the competitiveness of all four algorithms when the sizes of the caches maintained by them are different from that maintained by **O**PT.
- 6. We show that if the size of the cache is twice that of the one allowed for the optimal offline algorithm, then the competitiveness ratio drops to a small constant.

We use the method of potential functions to analyze the algorithms. However, the main challenges in solving these problems is that of carefully designing the potential function for the analysis. We discuss the role of the adaptive parameter in the potential function. The contributions of this paper are summarized in Table 1. In this table, N is the size of the cache maintained by the algorithm, while N_O is the size of the cache maintained by **OPT**. The table provides lower bounds (LB) and upper bounds (UB) on the competitiveness ratio when the cache sizes are equal, i.e., $N = N_O$; it also provides upper bounds when they are not equal.

Algorithm	Compet. Ratio	Compet. Ratio	Compet. Ratio UB	[Ref]
	LB	UB	w/ Unequal Sizes	
\mathbf{L}_{RU}	N	N	$N/(N-N_O+1)$	[ST85]
\mathbf{A}_{RC}	N+1	4N	$12N/(N-N_O+1)$	This paper
Clock	N	N	$N/(N-N_O+1)$	This paper
Car	N+1	18N	$18N/(N-N_O+1)$	This paper

Table 1: Summary of Results

After providing relevant background on cache replacement algorithms in Section 2, we discuss the lower bounds on the competitiveness ratios of ARC and CAR in Section 3. Next we prove upper bounds on the competitiveness ratios of LRU, CLOCK, ARC, and CAR in Section 4. Concluding remarks can be found in Section 5.

2 Previous Work on Cache Replacement Strategies

Below we give brief descriptions of the four algorithms being discussed in this paper, after which we mention a large collection of other closely related cache replacement algorithms.

The LRU Algorithm: LRU evicts the least recently used entry. It tends to perform well when there are many items that are requested more than once in a relatively short period of time, and performs poorly on "scans". LRU is expensive to implement because it requires a queue with move-to-front operations whenever a page is requested.

The CLOCK Algorithm: On the other hand, CLOCK was designed as an efficient approximation of LRU, which it achieves by avoiding the move-to-front operation. CLOCK's cache is organized as a single "circular" list, instead of a queue. The algorithm maintains a pointer to the "head" of the list. The item immediately counterclockwise to it is the "tail" of the list. Each item is associated with a "mark" bit. Some of the pages in the cache are marked, and the rest are unmarked. When a page hit occurs that page is marked, but the contents of the cache remain unchanged. When a page fault occurs, in order to make room for the requested page, the head page is evicted if the page is unmarked. If the head page is marked, the page is unmarked and the head is moved forward clockwise, making the previous head as the tail of the list. After a page is evicted, the requested page is unmarked and placed at the tail of the list. CLOCK is inexpensive to implement, but is not scan-resistant like LRU.

The ARC Algorithm To facilitate our discussion, we briefly describe the ARC algorithm. As mentioned before, it combines ideas of recency and frequency. ARC's cache is organized into a "main" part (of size N) and a "history" part (of size N). The main part is further divided into two lists, T_1 and T_2 , both maintained as LRU lists (i.e., sorted by "recency"). T_1 focuses on "recency" because it contains pages with short-term utility. Consequently, when an item is accessed for the first time from the disk, it is brought into T_1 . Items "graduate" to T_2 when they are accessed more than once. Thus, T_2 deals with "frequency" and stores items with potential for long-term utility. Additionally, ARC maintains a history of N more items, consisting of B_1 , i.e., items that have been recently deleted from T_1 , and T_2 , i.e., items that have been recently deleted from T_2 . History lists are also organized in the order of recency of access. The unique feature of ARC is its self-tuning capability, which makes it scan-resistant. Based on a self-tuning parameter, T_2 , the size of T_3 may grow or shrink relative to the size of T_3 . The details of the algorithm are fairly complex and non-intuitive. Detailed pseudocode for ARC (Figure 4 from [MM03]) is provided in the Appendix for convenience.

It is worth noting that **A**RC is considered a "universal" algorithm in the sense that it does not use any *a priori* knowledge of its input, nor does it do any offline tuning. Furthermore, **A**RC is continuously adapting, since adaptation can potentially happen at every step.

It must be noted that our results on ARC assume the "learning rate", δ , to be equal to 1, while the ARC algorithm as presented by Megiddo and Modha recommended a "faster" learning rate based on experiments on real data. The learning rate is the rate at which the adaptive parameter p is changed as and when needed.

The CAR Algorithm Inspired by ARC, CAR's cache is organized into two main lists, T_1 and T_2 , and two history lists, B_1 and B_2 . Inspired by CLOCK, both T_1 and T_2 are organized as "circular" lists, with each item associated with a mark bit. The history lists, B_1 and B_2 are maintained as simple FIFO lists. We let t_1, t_2, b_1, b_2 denote the sizes of T_1, T_2, B_1, B_2 , respectively. Also, let $t := t_1 + t_2$. Let lists L_1 (and L_2 , resp.) be the list of size ℓ_1 (ℓ_2 , resp.) obtained by concatenating list B_1 to the end of "linearized" T_1 (concatenating B_2 to the tail of T_2 , resp.). Note that circular lists are linearized from head to tail. We let T_1^0 and T_2^0 (T_1^1 and T_2^1 , resp.) denote the sequence of unmarked (marked, resp.) pages in T_1 and T_2 , respectively.

The following invariants are maintained by CAR for the lists:

- 1. $0 \le t_1 + t_2 \le N$
- $2. \ 0 \le \ell_1 = t_1 + b_1 \le N$
- 3. $0 \le \ell_1 + \ell_2 = t_1 + t_2 + b_1 + b_2 \le 2N$
- 4. $t_1 + t_2 < N \implies b_1 + b_2 = 0$
- 5. $t_1 + t_2 + b_1 + b_2 \ge N \implies t_1 + t_2 = N$
- 6. Once $t_1 + t_2 = N$ and/or $\ell_1 + \ell_2 = 2N$, they remain true from that point onwards.

CAR maintains an adaptive parameter p, which it uses as a target for t_1 , the size of list T_1 . Consequently, N-p is the target for t_2 . Using this guiding principle, it decides whether to evict an item from T_1 or T_2 in the event that a miss requires one of the pages to be replaced. The replacement policy can be summarized into two main points:

- 1. If the number of items in T_1 (barring the marked items at the head of the list) exceeds the target size, p, then evict an unmarked page from T_1 , else evict an unmarked page from T_2 .
- 2. If $\ell_1 = t_1 + b_1 = N$, then evict a history page from B_1 , else evict a history page from B_2 . Since the details of the algorithm are complex, the actual pseudocode is provided (Figure 2 from [BM04]) in the Appendix.

Other Cache Replacement Algorithms The DuelingClock algorithm [JIPP10] is like Clock but keeps the clock hand at the newest page rather than the oldest one, which allows it to be scan-resistant. More recent algorithms try to improve over Lru by implementing multiple cache levels and leveraging history. In [OOW93] the Lru-K algorithm was introduced. Briefly, the Lru-K algorithm estimates interarrival times from observed requests, and favors retaining pages with shorter interarrival times. Experiments have shown Lru-2 performs better than Lru, and that Lru-K does not show increase in performance over Lru-2 [OOW93], but has a higher implementation overhead. It was also argued that Lru-K is optimal under the independence reference model (IRM) among all algorithms A that have limited knowledge of the K most recent references to a page and no knowledge of the future [OOW93].

In essence, the LRU-K algorithm tries to efficiently approximate Least Frequently Used (LFU) cache replacement algorithm. As K becomes larger, it gets closer and closer to LFU. It has been argued that LFU cannot adapt well to changing workloads because it may replace currently "hot" blocks instead of "cold" blocks that had been "hot" in the past. LFU is implemented as a heap and takes $O(\log N)$ time per request.

Another cache replacement algorithm is Lirs [JZ02]. The Lirs algorithm evicts the page with the largest IRR (inter-reference recency). It attempts to keep a small ($\approx 1\%$) portion of the cache for HIR (high inter-reference) pages, and a large ($\approx 99\%$) portion of the cache for LIR (low inter-reference) pages. The Clock-Pro algorithm approximates Lirs efficiently using Clock [JCZ05]. The **2**Q [JS94] algorithm

is scan-resistant. It keeps a FIFO buffer A_1 of pages that have been accessed once and a main LRU buffer A_m of pages accessed more than once. 2Q admits only hot pages to the main buffer. The buffer A_1 is divided into a main component that keeps the pages in A_1 that still reside in cache, and a history component that remembers pages that have been evicted after one access. The relative sizes of the main and history components are tunable parameters. 2Q has time complexity of O(1). Another algorithm that tries to bridge the gap between recency and frequency is LRFU [LCK+01]. This is a hybrid of LRU and LFU and is adaptive to changes in workload. The time complexity ranges from O(1) for LRU to $O(\log n)$ for LFU.

3 Lower Bounds on Competitiveness Ratio for Arc and Car

This section presents our results on the lower bounds for ARC and CAR. We also show that the adaptive parameter is critical to both ARC and CAR by showing that their non-adaptive versions have an unbounded competitiveness ratio.

3.1 Lower Bound for ARC

First, we show a lower bound on the competitiveness ratio for ARC.

Theorem 1. The competitiveness ratio of Algorithm ARC has a lower bound of N + 1.

Proof. We show that we can generate an unbounded request sequence that causes N+1 page faults on \mathbf{A}_{RC} for every page fault on \mathbf{O}_{PT} . The sequence only involves 2N+1 pages denoted by $1,\ldots,2N+1$. Our example, will take the contents of the cache managed by \mathbf{A}_{RC} from configurations 1 through configuration 5, which are shown in Table 2. Note that configuration 1 and configuration 5 are essentially the same to the extent that the value of p is 0 in both, and the number of pages in each of the four parts of the cache are identical.

Configuration	p	T_1	T_2	B_1	B_2
1	0	Ø	$1, \ldots, N$	Ø	$N+1,\ldots,2N$
2	0	2N+1	$2, \ldots, N$	Ø	$N+2,\ldots,2N,1$
3	0	Ø	$2,\ldots,N,1$	2N + 1	$N+2,\ldots,2N$
4	1	Ø	$3, \ldots, N, 1, 2N+1$	Ø	$N+2,\ldots,2N,2$
5	0	Ø	$1,2N+1,2,\ldots,N-1$	Ø	$N+2,\ldots,2N,N$

Table 2: Example for Lower Bound on ARC's competitiveness

We note that we can obtain configuration 1 from an empty cache with the following request sequence: $2N, 2N, 2N - 1, 2N - 1, \ldots, 2, 2, 1, 1$. Consider the first half of the above request sequence, which contains a total of 4N requests to 2N new pages, each page requested twice in succession. The first time a page is requested from the first N new pages, it will be put in T_1 . The second time the page is requested, it will get moved to T_2 . In the second half, if a page not in A_{RC} is requested, R_{EPLACE} will be called, which will move a page from T_2 to B_2 , and the new page will be placed in T_1 . When the same page is requested again, it simply gets moved to T_2 . The value of p remains unchanged in this process. It is clear that we get Configuration 1 as a result of the request sequence.

We design our request sequence by following the steps below.

- 1. Make one request to a page 2N + 1 not in ARC. We will assume that this is a brand new page and therefore also causes a fault for OPT and for ARC. The page 2N + 1 will go into T_1 and a page in T_2 will be demoted to B_2 . The contents of ARC is given by Configuration 2 in Table 2.
- 2. Request any page in B_2 . This decreases the value of p but since p is zero it will remain unchanged. Since the size of T_1 is more than p **A**RC will call REPLACE, which will act on T_1 , hence 2N + 1 will be demoted to B_1 . Upon requesting page 1 in B_2 , we get Configuration 3 in Table 2.

- 3. The next step is to request 2N + 1 again, which will move to T_2 , p is increased and a page in T_2 is demoted to B_2 . Configuration 4 reflects the contents of the cache at this stage.
- 4. Finally we make N-2 requests to any pages from B_2 . By requesting the pages $2, 3, \ldots, N$, we end up in Configuration 5 from Table 2.

The steps outlined above cause N+1 page faults for \mathbf{A}_{RC} and at most one page fault for \mathbf{O}_{PT} . Since we are back to the initial configuration we can repeat this process over again. This concludes the proof that the competitiveness ratio of \mathbf{A}_{RC} is at least N+1.

3.2 Lower Bound for CAR

Now we prove a similar lower bound for CAR.

Theorem 2. The competitiveness ratio of Algorithm CAR has a lower bound of N+1.

Proof. We show that we can generate an infinite request sequence that causes N+1 page faults in CAR for every page fault on OPT. The sequence only involves 2N+1 pages denoted by $1, \ldots, 2N+1$. Our example, will take the contents of the cache managed by CAR from configurations 1 through N+2 as shown in Table 3. Note that a superscript of 1 on any page in $T_1 \cup T_2$ indicates that it is marked. All others are unmarked. Also note that configuration 1 and configuration N+2 are essentially the same upon relabeling.

First, we show that configuration 1 is attainable, by showing that it can be reached from an empty cache. This is formalized in the following lemma.

Lemma 1. We can obtain configuration 1 starting from an empty cache with the following request sequence: $2N, 2N, 2N - 1, 2N - 1, \ldots, 2, 2, 1, 1$.

Proof. The first half of the above request sequence calls each of the N pages 2N, 2N - 1, ..., N + 1 twice in succession. The first time they are called, they are moved into T_1 unmarked. The second time the same page is called it gets marked, but remains in T_1 . At the end of the first half, all the N pages requested end up in T_1 and are all marked.

The next call to new page N, will trigger a call to REPLACE, which will move all the marked pages in T_1 to T_2 leaving them unmarked. It will also move one page from T_2 to B_2 . Finally, the requested page N will be moved to T_1 and left unmarked. When requested again, it simply gets marked. When the next page, i.e., N-1 is requested, it moves marked page N to T_2 , moves one more page from T_2 to T_2 . As the rest of the pages from the request sequences are requested, the previous requested page gets moved to T_2 , which in turn demotes one of its pages to T_2 . At the end of the process, we get a marked page T_2 in T_2 . Pages T_2 , which in turn are in T_2 , unmarked, and pages T_2 , and up in T_2 . This is exactly what we need for configuration T_2 .

Continuing on the proof of Theorem 2, we show what happens when, starting from configuration 1, CAR processes the following request sequence.

Page 2N + 1: A page in T_2 is demoted to B_2 , which loses a page; the marked page from T_1 is moved to T_2 and the new page is moved into T_1 .

MRU page in B_2 : This should have decremented p but remains unchanged since it is already zero. Since the size of T_1 is more than p CAR will call REPLACE and 2N + 1 will be demoted to B_1 , resulting in configuration 3 in Table 3.

Page 2N + 1: It will move to T_2 , p is increased and a page in T_2 is demoted to B_2 . See configuration 4 in Table 3.

MRU page from B_2 , repeat N-2 times: It results in configuration N+2 in Table 3.

Config.	p	B_1	T_1	T_2	B_2	
1	0	Ø	1^1	$2,\ldots,N$	$N+1,\ldots,2N$	
2	0	Ø	2N + 1	$1,\ldots,N-1$	$N,\ldots,2N-1$	
3	0	2N+1	Ø	$N, 1, \ldots, N-1$	$N+1,\ldots,2N-1$	
4	1	Ø	Ø	$2N+1, N, 1, \dots, N-2$	$N-1, N+1, \dots, 2N-1$	
5	0	Ø	Ø	$N-1, 2N+1, N, 1, \dots, N-3$	$N-2, N+1, \dots, 2N-1$	
	0					
N+2	0	Ø	Ø	$2, \ldots, N-1, 2N+1, N$	$1, N+1, \ldots, 2N-1$	

Table 3: Example for Lower Bound on CAR's competitiveness

The request sequence detailed above generates N+1 faults for CAR while only N different pages are requested. Thus, OPT could limit itself to at most one fault in this stretch. OPT will fault once during each stretch if the next page is picked to be one that is farthest used in the future. Repeating the above steps an unbounded number of times with appropriate relabeling proves that the competitiveness ratio of CAR is lower bounded by N+1.

3.3 Non-Adaptive ARC and CAR are not Competitive

It is particularly interesting to note that the non-adaptive version of **C**AR and **A**RC (called *Fixed Replacement cache*) [MM03] are not competitive. The following two theorems prove that the competitiveness ratios can be unbounded in this case.

Theorem 3. Algorithm **C**AR with fixed p is not competitive.

Proof. Suppose that algorithm \mathbf{C}_{AR} has p fixed instead of being adaptive and 0 . Recall that <math>p is the target size of T_1 and N-p is the target size of T_2 . We design a request sequence such that with less than N pages we can generate an infinite number of page faults for \mathbf{C}_{AR} . The sequence is described as follows:

- **Step 1:** Fill up T_2 with N-p unmarked pages as described above in the proof of Theorem 2.
- **Step 2:** Request the MRU page in B_2 . The requested page goes to the tail of T_2 as an unmarked page. Since the size of T_2 is greater than p we discard the head of T_2 .
- Step 3: Request the MRU page in B_2 which is actually the page discarded in Step 2 from T_2 . This step is similar to Step 2 and we can continue to repeat this infinitely often, since the page that moves from B_2 to T_2 get's unmarked and the page that moves from T_2 to B_2 goes to MRU.

Therefore, we can cycle infinitely many times through N-p+1 pages triggering an infinite number of faults, while **O**PT can avoid faults altogether during the cycle.

Theorem 4. Algorithm \mathbf{A}_{RC} with fixed p is not competitive.

Proof. Suppose that algorithm \mathbf{A} RC has p fixed instead of being adaptive and 0 . Recall that <math>p is the target size of T_1 and N-p is the target size of T_2 . We design a request sequence such that with less than N pages we can generate an infinite number of page faults for \mathbf{A} RC. The first step is to fill up T_2 (size of $T_2 = N - p$). Next we request the MRU page in B_2 . Every time we request a page from B_2 , it goes into the top of T_2 and thus it increases the size of T_2 beyond its target size. It follows that \mathbf{A} RC will call Replace and move a page from T_2 to the MRU position in T_2 . If the MRU page from T_2 is repeatedly requested, we will cycle through T_2 pages, every time incurring a page fault for T_2 while T_2 or avoid faults altogether during the cycle.

4 Analyzing LRU using potential functions

4.1 The generic approach

The standard approach used here is as follows. First, we define a carefully crafted potential function, Φ . As per the strategy of analyzing competitiveness ratios suggested by Sleator and Tarjan [ST85], we then try to show the following inequality:

$$C_A + \Delta \Phi \le f(N) \cdot C_O + g(N), \tag{1}$$

where C_A and C_O are the costs incurred by the algorithm and by **O**PT, respectively, $\Delta\Phi$ is the change in potential, f(N) is some function of N, the size of the cache.

In all of our proofs, we assume that the work involves simultaneously maintaining **O**PT's cache as well as the algorithm's cache. So we can break down the work into two steps, one where only **O**PT serves and one where only the algorithm serves. When only **O**PT serves, there are 2 cases: first when **O**PT has a hit and next when **O**PT has a miss. Next, we consider the cases when the algorithm serves, once when it has a hit and once when it has a miss. In each case, our goal is to prove the inequality (1) mentioned above, which establishes that f(N) is the competitiveness ratio of algorithm A. There may be an additive term of g(N) which is a function of the misses needed to get to some initial configuration for the cache.

4.2 Analyzing LRU using potential functions

Assuming that the size of cache given to the competing **O**PT algorithm is $N_O \leq N$, the following result was proved by Sleator and Tarjan [ST85] (Theorem 6) for **L**RU.

Theorem 5. [ST85] Algorithm LRU is $(\frac{N}{N-N_O+1})$ -competitive.

Here we present a complete proof of this well-known result because we believe it is instructive for the other proofs in this paper.

Proof. While this was not used in the proof in Sleator and Tarjan [ST85], a potential function that will facilitate the proof of the above theorem is:

$$\Phi = \frac{\sum_{x \in D} r(x)}{N_L - N_O + 1},\tag{2}$$

where D is the list of items in LRU's cache but not in OPT's cache, and r(x) is the rank of item x in LRU's list with the understanding that the LRU item has rank 1, while the MRU item has rank equal to the size of the cache [Alb96].

We now show the following inequality:

$$C_A + \Delta \Phi \le \left(\frac{N}{N - N_O + 1}\right) \cdot C_O + O(N),\tag{3}$$

where C_A and C_O are the costs incurred by the algorithm and by **O**PT, respectively, $\Delta\Phi$ is the change in potential, f(N) is some function of N, the size of the cache.

We assume that the work involves simultaneously maintaining \mathbf{O} PT's cache as well as \mathbf{L} RU's cache. So we can break down the work of \mathbf{L} RU into two steps, one where only \mathbf{O} PT serves and one where only \mathbf{L} RU serves. When only \mathbf{O} PT serves, there are 2 cases: first when \mathbf{O} PT has a hit and next when \mathbf{O} PT has a miss. In either case, the cost for \mathbf{L} RU is 0, since only \mathbf{O} PT is serving. When \mathbf{O} PT has a hit, the cost for \mathbf{O} PT is also 0. Furthermore, since \mathbf{L} RU's cache remains untouched, and no changes take place in the contents of \mathbf{O} PT's cache, the ranks of the items in \mathbf{L} RU remain unchanged. Thus, $\Delta \Phi = 0$. Therefore, the inequality in (3) is trivially satisfied in this case.

When **O**PT has a miss, $C_O = 1$, as before. The item evicted by **O**PT can contribute the rank of that item to increase at most by N_L , making the increase in potential function to be bounded by $\frac{N_L}{N_L - N_O + 1}$. Thus, the inequality in (3) is satisfied.

Next, we consider the step where \mathbf{L}_{RU} serves the request. As with \mathbf{O}_{PT} , when \mathbf{L}_{RU} is serving, the cost for \mathbf{O}_{PT} is 0. We again consider two cases: first when \mathbf{L}_{RU} has a hit and next when \mathbf{L}_{RU} has a miss. When \mathbf{L}_{RU} has a hit, the cost for \mathbf{L}_{RU} is 0. The contents of \mathbf{L}_{RU} 's cache may change. The item that was accessed is moved to the MRU position. However, this item is already in \mathbf{O}_{PT} 's cache and therefore cannot contribute to a change in potential. Several other items may move down in the cache, thus contributing to a decrease in potential of at most (N-1). In the worst, case the increase in potential is at most 0. Therefore, the inequality in (3) is again satisfied.

Finally, we consider the case when LRU has a miss. As before, $C_L = 1$. Following the previous arguments, an item would be brought into MRU (which is already present in **O**PT's cache), a bunch of items may be demoted in rank, and the LRU item will be evicted. The only action that can contribute to an increase is caused by the item that is brought into the MRU location. However, this item is already present in **O**PT's cache, and hence cannot contribute to an increase. All the demotions and eviction can only decrease the potential function. Note that before the missed item is brought into LRU's cache, the contents of LRU's and **O**PT's cache agree in at most $N_O - 1$ items, since **O**PT just finished serving the request and the item that caused the miss is already in **O**PT's cache. Thus there are at least $N_L - N_O + 1$ items that contribute their ranks to the potential function. These items either get demoted in rank or get evicted. Either way, the potential function will reduce by a minimum value of $N_L - N_O + 1$, although it could more if there are more items that are in LRU and that are not in **O**PT's cache. Thus the total change in potential has to be at most $N_L - N_O + 1$, and we have

$$C_L + \Delta \Phi \le 1 - \frac{(N_L - N_O + 1)}{(N_L - N_O + 1)} \le 0 = \frac{N_L}{N_L - N_O + 1} \cdot C_O.$$

Summarizing the costs, we have the following:

Step	C_L	$\Delta\Phi$	C_O		
Opt Serves Request					
OPT has a hit	0	0	0		
Opt has a miss	0	$\leq N_L$	1		
LRU Serves Request					
LRU has a hit	0	≤ 0	0		
LRU has a miss	1	$\leq N_L - N_O + 1$	0		

The analysis of LRU states that if the sizes of LRU's and OPT's caches are N_L and N_O respectively, and if $N_L \geq N_O$, then the competitiveness ratio of LRU is $\frac{N_L}{N_L - N_O + 1}$. Thus LRU is 2-competitive if the size of LRU's cache is roughly twice that of OPT's cache.

4.3 Analyzing the competitiveness of Clock

Our result on the competitiveness of Clock is formalized in the following theorem. While this result appears to be known, we have not been able to locate a full proof and we believe this is of value. We therefore present it for the sake of completeness.

Theorem 6. Algorithm Clock is $(\frac{N}{N-N_O+1})$ -competitive.

Proof. Let M_0 denote the subsequence of unmarked pages in CLOCK, ordered counterclockwise from head to tail. Let M_1 denote the subsequence of marked pages in CLOCK, ordered counterclockwise from head to tail. Let q be any page in CLOCK's cache. Let $P^0[q]$ denote the position of an unmarked page q in the

ordered sequence M_0 , and let $P^1[q]$ denote the position of a marked page q in M_1 . Finally, let R[q] denote the rank of page q defined as follows:

$$R[q] = \begin{cases} P^0[q] & \text{if } q \text{ is unmarked,} \\ P^1[q] + |M_0| & \text{otherwise.} \end{cases}$$
 (4)

Thus, if q is an unmarked page at the head, then R[q] = 1. By the above definition, the following lemmas are obvious.

Lemma 2. If q is any page in CLOCK's cache, then $R[q] \leq N$.

Lemma 3. If a marked page q at the head of Clock's cache is unmarked and moved to the tail, then R[q] does not change in the process.

Let D be the set of pages that are in the cache maintained by \mathbf{C}_{LOCK} , but not in the cache maintained by \mathbf{O}_{PT} . We define the potential function as follows:

$$\Phi = \sum_{q \in D} R[q] \tag{5}$$

We prove one more useful lemma about the ranks as defined above.

Lemma 4. If an unmarked page at the head of Clock's cache is evicted from Clock's cache, and if there is at least one page in D, then Φ decreases by at least 1 in the process.

Proof. All pages, marked or unmarked, will move down by at least one position (reducing the rank of each by at least 1). The decrease in potential for at least one page that is in D will contribute to Φ , guaranteeing that $\Delta \Phi \leq -1$.

Let C_{Clock} and C_{Opt} be the costs incurred by the algorithms CLOCK and OPT, and let $S = \sigma_1, \sigma_2, \ldots, \sigma_m$ be an arbitrary request sequence. Let S' denote the initial subsequence of requests that take place prior to the cache becoming full. Note that exactly N faults are incurred in S', after which the cache remains full. Let S'' be the subsequence of S that comes after S'.

Let $C_{\mathbf{C}lock}$ and $C_{\mathbf{O}pt}$ be the cost incurred by the algorithms $\mathbf{C}_{\mathsf{LOCK}}$ and \mathbf{O}_{PT} respectively. We will prove that for every individual request, $\sigma \in \mathcal{S}''$:

$$C_{Clock}(\sigma) + \Delta \Phi \le N * C_{Opt}(\sigma)$$
 (6)

As before, we assume that request σ is processed in two distinct steps: first when **O**PT services the page request and, next when **C**LOCK services the request. We will show that inequality (6) is satisfied for both the steps.

When only **O**PT acts in this step, $C_{clock} = 0$. If **O**PT does not fault on this request, then $C_{OPT} = 0$. No change occurs to the contents of the cache maintained by **O**PT as well as **C**LOCK, and the clock hand does not move. Thus, $\Delta \Phi = 0$, satisfying inequality 6.

If **O**PT faults on request σ , then $C_{OPT} = 1$ and $C_{Clock} = 0$. The contents of the cache maintained by **O**PT does change, which could affect the potential function. The potential could increase due to the eviction of a page in **O**PT. Since by Lemma 2 the rank of the evicted page cannot exceed N, the potential will change by at most N. Thus, inequality 6 is satisfied.

Next we consider what happens when C_{LOCK} services the request. For this case $C_{OPT}=0$. If C_{LOCK} does not fault, then $C_{clock}=0$ and the requested page may change from an unmarked status to a marked one. However, since the page is already in the cache maintained by OPT it is not in D and is therefore not considered for the potential function calculations in 5. Thus, inequality 6 is satisfied.

Finally, we consider the case when CLOCK faults, in which case $C_{Clock} = 1$ and $C_{Opt} = 0$. To satisfy inequality 6, $\Delta\Phi$ needs to be less or equal to -1. When CLOCK has a miss, if the head page happens to be marked, then CLOCK will repeatedly unmark the marked head page, moving it to the tail position, until an unmarked head page is encountered. The unmarked head page is then evicted. Each time a marked head page becomes an unmarked tail page, by Lemma 3 its rank does not change. When finally an unmarked head page is evicted, we know that there is at least one page in OPT's cache that is not in CLOCK's cache (i.e., the page that caused the fault). Since there are N pages in the cache maintained by CLOCK, at least one of those pages is guaranteed not to be part of the cache maintained by OPT. Since there is at least one page in D, by Lemma 4 it is clear that evicting an unmarked head page will decrease the potential function by at least one, which will pay for the CLOCK's page fault.

We have therefore showed that for every request σ , inequality 6 is satisfied. Since there can be at most N faults for the requests in S', summing up the above inequality for all requests, $\sigma \in S$, we get

$$C_{\mathbf{C}lock}(\mathcal{S}) \leq N * C_{\mathbf{O}pt}(\mathcal{S}) + N.$$

This completes the proof of the theorem and the competitiveness analysis of the \mathbf{C}_{LOCK} algorithm.

4.4 Analyzing the Competitiveness of ARC

In this paper, we prove two different upper bounds for the competitiveness of ARC. These two proofs use very different potential function. The first one allows for the sizes of the caches maintained by ARC and OPT to be different, while the second one does not allow for it, but provides a tighter bound. We provide both results below.

Our first result on the competitiveness of ARC is formalized in the following theorem:

Theorem 7. Algorithm **A**RC is
$$\left(\frac{12N}{N-N_O+1}\right)$$
-competitive.

Proof. Let $P_X[q]$ be the position of page q in an arbitrary ordered sequence of pages X. When the set is obvious, we will drop the subscript and denote $P_X[q]$ simply by P[q]. The set of history pages T_1 , T_2 , B_1 , and B_2 will be treated as an ordered sequence of pages ordered from its LRU position to its MRU position. Let $\mathbf{O}pt$ and $\mathbf{C}ar$ be the set of main pages stored in the caches for algorithms $\mathbf{O}PT$ and $\mathbf{A}RC$ respectively. Let $D = \mathbf{A}rc \setminus \mathbf{O}pt$. As before, we associate each page with a rank value R[q], which is defined as follows:

$$R[q] = \begin{cases} 2P_{B_1}[q] & \text{if } q \in B_1\\ 2P_{B_2}[q] & \text{if } q \in B_2\\ 4P_{T_1}[q] + 2b_1 & \text{if } q \in T_1\\ 4P_{T_2}[q] + 2b_2 & \text{if } q \in T_2 \end{cases}$$

$$(7)$$

Finally, we define the potential function as follows:

$$\Phi = p + 2t_1 + 2\left(\frac{\sum_{q \in D} R[q]}{N - N_O + 1}\right) - 3|\mathbf{A}rc|$$
(8)

The initial value of Φ is 0. If the following inequality (9) is true for any request σ , where $\Delta\Phi$ is the change in potential caused by serving the request, then when summed over all requests, it proves Theorem 7.

$$C_{\mathbf{A}rc}(\sigma) + \Delta\Phi \le \frac{12NC_{\mathbf{O}pt}(\sigma)}{N - N_O + 1}.$$
(9)

As before, we assume that request σ is processed in two distinct steps: first when **O**PT serves and, next when **A**RC serves. We will show that inequality (9) is satisfied for each of the two steps.

Step 1: Opt serves request σ

Since only **O**PT acts in this step, $C_{\mathbf{A}rc} = 0$, and $T_1 \cup T_2$ does not change. There are two possible cases: either **O**PT faults on σ or it does not. If **O**PT does not fault on this request, then it is easy to see that $C_{\mathbf{O}pt} = 0$ and $\Delta \Phi = 0$, thus satisfying inequality (9).

If **O**PT faults on request σ , then $C_{\mathbf{O}pt}=1$ and some page q, is evicted from the cache maintained by **O**PT will belong to D after this step and thus its rank will contribute to the potential function, which will increase by two times the rank of q. The maximal positive change in potential will occur when q is the MRU page of either T_1 or T_2 . In this case the rank of q is given by: $R[q]=4P[q]+b_1$ ($R[q]=4P[q]+b_2$). The maximum possible values for each of the terms P[q] and b_1 will be N, hence the maximum possible rank of q will be 4N+2N=6N. Therefore resulting potential change is at most $\frac{12N}{N-N_O+1}$.

Step 2: Arc serves request σ

We break down the analysis into four cases. Case 2.1 deals with the case when ARC finds the page in its cache. The other three cases assume that ARC faults on this request because the item is not in $T_1 \cup T_2$. Cases 2.2 and 2.3 assume that the missing page is found recorded in the history in lists B_1 and B_2 , respectively. Case 2.4 assumes that the missing page is not recorded in history.

Case 2.1: ARC has a page hit Clearly, the page was found in $T_1 \cup T_2$, and $C_{Arc} = 0$. We consider the change of each of terms in the potential function individually.

- 1. As per the algorithm, p can only change when the page is found in history. (See lines 3 through 10 of $\mathbf{A}_{RC}(x)$.) Since the page is not found in \mathbf{A}_{RC} 's history, $\Delta p = 0$.
- 2. If the hit happens in T_1 , the page will move to the top of T_2 (See line 2 of $\mathbf{A}_{RC}(x)$.), which will result in a decrease in t_1 . If the hit happens in T_2 , the size of t_1 will remain the same. The overall change in t_1 will be 0.
- 3. Since **O**PT has already served the page, the page is in **O**PT's cache. Therefore, even if the page's rank could change when moved from T_1 to MRU position of T_2 , this rank will not affect the potential since the page is not in D.

We, therefore, conclude that $\Delta \Phi = 0$, satisfying inequality (9).

Next we will analyze the 3 cases when the requested page is not in ARC's cache. Since $C_{Arc} = 1$, the change in potential must be ≤ -1 in each case in order for inequality (9) to be satisfied.

Case 2.2: ARC has a page miss and the missing page is in B_1 We consider the two cases, first when REPLACE moves an item from T_1 to B_1 and second when it moves an item from T_2 to B_2 .

- 1. Case 1: We consider the change in potential function by analyzing each of the 3 terms.
 - Value of p will either increase by 1 or stay the same in case p = N, we will account for the worst case which is when $\Delta p = 1$.
 - A new page is being added to MRU of T_2 , and REPLACE is taking the LRU page of T_1 to B_1 , then $2\Delta t_1 = -2$.
 - The page that moved from B_1 to T_2 is not in D, therefore the change in its rank will not affect the potential, the other pages will could only decrease their rank, meaning that $2\Delta \sum_{q \in D} R[q] \leq 0$.

Since p increases by at most 1 and t_1 decreases by at least 2 the total change in potential is at most -1.

- 2. Case 2: Once again. we consider the change in potential function by analyzing each of the three terms.
 - Value of p will either increase by 1 or stay the same in case p = N, we will account for the worst case which is when $\Delta p = 1$.

- A new page is added to MRU of T_2 , and REPLACE moves the LRU page of T_2 to B_2 . Thus, there is no change in T_1 .
- The page that moved from B_1 to T_2 is not in D, therefore the change in its rank will not affect the potential. Since $t_1 + t_2 = N$, it is guaranteed that at least $N N_O + 1$ pages are not in **O**PT. For the pages that are in T_1 , their ranks will decrease by at least 2 since b_1 decreases by 1, and for the pages in T_2 their ranks will decrease by at least 2 as well since b_2 increases by 1 but the LRU page in T_2 will move to B_2 , reducing P[q] for all the pages in T_2 . The term $2\frac{\sum_{q\in D} R[q]}{N-N_O+1}$ decreases by at least -4.

Since p increases by at most 1 and $2\frac{\sum_{q\in D}R[q]}{N-N_O+1}$ decreases by at least -4 the total change in potential is at most -3.

Case 2.3: ARC has a page miss and the missing page is in B_2 When the missing page is in B_2 , ARC makes a call to REPLACE (Line 5) and then executes Lines 18-19. Thus, p is decremented except if it is already equal to 0. We consider two sub cases: $\Delta p \leq -1$ and $\Delta p = 0$.

 $\Delta p \leq -1$: As in Case 2.2, the call to REPLACE has no effect on t_1 . REPLACE will not increment the rank using a similar analysis as in 2.2 and change in p will at least be -1. The change in the potential function is at most -1.

 $\Delta p = 0$: Unlike the sub case above when p decreases by 1, the change in p cannot guarantee the required reduction in the potential. We therefore need a tighter argument. We know that there is a call to REPLACE. Two cases arise and are discussed below.

- REPLACE moves an item in T_1 to B_1 : Since the LRU page of T_1 is moved to the MRU position of B_1 , $2\Delta t_1 = -2$ and there is no movement of a page in D that could increase the rank. Therefore the total change in the potential function is at most -2.
- REPLACE moves an item in T_2 to B_2 : p=0 indicates that T_2 has N pages, therefore is guarantee that at least $N-N_O+1$ pages will not be part of **O**PT, contributing to the change in potential. The page being moved from T_2 to B_2 will decrease it's rank by at least 2, and the rest of the pages in T_2 will move down one position (P[q] will decrease by 1) while B_2 will remain the same, resulting in a change in the potential function of at most -4.

Thus, in each case the potential function decreased by at most -2.

Case 2.4: ARC has a page miss and the missing page is not in $B_1 \cup B_2$

- 1. $t_1 + b_1 = N$; $t_1 < N$; The LRU page in B_1 is evicted. Assume REPLACE moves a page from T_1 to B_1 and a new page is brought into T_1 ($\Delta t_1 = 0$, $\Delta b_1 = 0$, $\Delta t_2 = 0$, $\Delta b_2 = 0$).
 - The term p is not affected.
 - The term t_1 is not affected.
 - Since $t_1 + b_1 = N$, at least $N N_o + 1$ pages in $T_1 \cup B_1$ are not in $\mathbf{O}pt$. If the page is in $B_1 \setminus \mathbf{O}pt$ then its rank decreases by 2; if the page is in $T_1 \setminus \mathbf{O}pt$ its rank decreases by 4.
- 2. $t_1 + b_1 = N$; $t_1 < N$; The LRU page in B_1 is evicted. Assume REPLACE moves a page from T_2 to B_2 and a new page is brought into T_1 ($\Delta t_1 = 1$, $\Delta b_1 = -1$, $\Delta t_2 = 1$, $\Delta b_2 = 1$).
 - The term p is not affected.
 - The term t_1 is increased by 1.

- Since $t_1 + t_2 = N$, at least $N N_o + 1$ pages in $T_1 \cup T_2$ that are not in **O**PT. If a page, q, is in $T_1 \setminus \mathbf{O}pt$ then its rank decreases by 2 $(\Delta R[q] = \Delta 4 * P[q] + \Delta 2 * b_2 = -2)$; if the page, q, is in $T_2 \setminus \mathbf{O}pt$ its rank decreases by 2 $(\Delta R[q] = \Delta 4 * P[q] + \Delta 2 * b_2 = -2)$.
- 3. $t_1 + b_1 < N$; $t_1 + t_2 + b_1 + b_2 = 2N$; Assume that the LRU page in B_2 is evicted and REPLACE moves a page from T_1 to B_1 and a new page is brought into T_1 ($\Delta t_1 = 0$, $\Delta b_1 = 1$, $\Delta t_2 = 0$, $\Delta b_2 = -1$).
 - The term p is not affected.
 - The term t_1 is not affected.
 - Here we used the fact that $t_2 + b_2 > N$, then at least $N N_o + 1$ pages in $T_2 \cup B_2$ are not in **Opt**. If a page, q, is in $T_2 \setminus \mathbf{Opt}$ then its rank decreases by $2 (\Delta R[q] = \Delta 4 * P[q] + \Delta 2 * b_2 = 4 * (0) + 2(-1) = -2)$; if the page, q, is in $B_2 \setminus \mathbf{Opt}$ its rank decreases by $2 (\Delta R[q] = \Delta 2 * P[q] = 2 * (-1) = -2)$.
- 4. $t_1 + b_1 < N$; $t_1 + t_2 + b_1 + b_2 = 2N$; Assume that the LRU page in B_2 is evicted and REPLACE moves a page from T_2 to B_2 and a new page is brought into T_1 ($\Delta t_1 = 1$, $\Delta b_1 = 0$, $\Delta t_2 = 1$, $\Delta b_2 = 0$).
 - \bullet The term p is not affected.
 - The term t_1 is increased by 1.
 - Here we used the fact that $t_2 + b_2 > N$, then at least $N N_o + 1$ pages in $T_2 \cup B_2$ are not in $\mathbf{O}pt$. If a page, q, is in $T_2 \setminus \mathbf{O}pt$ then its rank decreases by $2 (\Delta R[q] = \Delta 4 * P[q] + \Delta 2 * b_2 = 4 * (0) + 2(-1) = -2)$; if the page, q, is in $B_2 \setminus \mathbf{O}pt$ its rank decreases by $2 (\Delta R[q] = \Delta 2 * P[q] = 2 * (-1) = -2)$.
- 5. $t_1 + b_1 < N$; $t_1 + t_2 + b_1 + b_2 < 2N$; In this case, no pages are evicted from history. Assume that REPLACE moves a page from T_1 to B_1 and a new page is brought into T_1 ($\Delta t_1 = 0$, $\Delta b_1 = 1$, $\Delta t_2 = 0$, $\Delta b_2 = 0$)
 - The term p is not affected.
 - The term t_1 is increased by 1.
 - Here we cannot say that the rank decreases. Hence the rank term is at most 0.
 - The term $|\mathbf{A}rc|$ increases by 1.
- 6. $t_1 + b_1 < N$; $t_1 + t_2 + b_1 + b_2 < 2N$; In this case, no pages are evicted from history. Assume REPLACE moves a page from T_2 to B_2 and a new page is brought into T_1 ($\Delta t_1 = 1$, $\Delta b_1 = 0$, $\Delta t_2 = -1$, $\Delta b_2 = 1$)
 - The term p is not affected.
 - The term t_1 is not affected.
 - Here we cannot say that the rank decreases. Hence the rank term is at most 0.
 - The term $|\mathbf{A}rc|$ increases by 1.

Wrapping up the proof of Theorem 7: Combining the four cases (2.1 through 2.4) proves that inequality (9) is satisfied when ARC serves request σ . This completes the proof of Theorem 7, establishing that the upper bound on the competitiveness of ARC is 12N for the cases where the sizes of OPT and ARC are the same. By analyzing cases where the size of ARC is greater than OPT we can observe that since ARC will be $\frac{12N}{N-N_O+1}$ the greater the size of ARC's cache relative to the size of OPT's cache, smaller will be the competitiveness of ARC.

4.5 Alternative Analysis of Competitiveness of ARC

Below, we prove an improved upper bound on the competitiveness ratio of \mathbf{A}_{RC} . As seen below, the potential function is considerably different. Let C_A and C_O be the costs incurred by the algorithms \mathbf{A}_{RC} and \mathbf{O}_{PT} .

We start with some notation and definitions. If X is the set of pages in a cache, then let MRU(X) and LRU(X) be the most recently and least recently used pages from X. Let $MRU_k(X)$ and $LRU_k(X)$ be the k most recently and k least recently used pages from X.

Let lists L_1 (and L_2) be the lists obtained by concatenating lists T_1 and B_1 (T_2 and B_2 , resp.). Let list L be obtained by concatenating lists L_1 and L_2 . We let $\ell_1, \ell_2, t_1, t_2, b_1, b_2$ denote the sizes of $L_1, L_2, T_1, T_2, B_1, B_2$, respectively. Finally, let $t := t_1 + t_2$ and $\ell := \ell_1 + \ell_2$.

At any instant of time during the parallel simulation of **O**PT and **A**RC, and for any list X, we let $MRU_k(X)$ be denoted by TOP(X), where k is the largest integer such that all pages of $MRU_k(X)$ are also in the cache maintained by OPT. We let L'_1, L'_2, T'_1, T'_2 denote the TOPs of L_1, L_2, T_1, T_2 , respectively, with sizes $\ell'_1, \ell'_2, t'_1, t'_2$, respectively. We let b'_1 and b'_2 denote the sizes of the $B'_1 = L'_1 \cap B_1$ and $B'_2 = L'_2 \cap B_2$, respectively. Note that if $b'_1 > 0$ ($b'_2 > 0$, resp.), then all of T_1 (T_2 , resp.) is in **O**PT. Finally, we let $\ell' := \ell'_1 + \ell'_2$. The **A**RC algorithm ensures that $0 \le t \le N$, $0 \le \ell \le 2N$ and $0 \le \ell_1 \le N$, thus making $0 \le \ell_2 \le 2N$.

We assume that algorithm X being analyzed is provided an arbitrary request sequence $\sigma = \sigma_1, \sigma_2, \dots, \sigma_m$. We define the potential function as follows:

$$\Phi = p - (b_1' + 2 \cdot t_1' + 3 \cdot b_2' + 4 \cdot t_2'). \tag{10}$$

The main result of this section is the following theorem:

Theorem 8. Algorithm ARC is 4N-competitive.

We say that the cache is full if t = N and either $t_1 + b_1 = N$ or $t_2 + b_2 \ge N$. We will prove the above theorem by proving the following inequality for any request σ that is requested after the cache is full:

$$C_A(\sigma) + \Delta \Phi \le 4N \cdot C_O(\sigma) + 2N, \tag{11}$$

where ΔX represents the change in any quantity X. Summing up the above inequality for all requests would prove the theorem as long as the number of faults prior to the cache becoming full is bounded by the additive term 2N.

We make the following useful observation about a full cache.

Lemma 5. When the request sequence requests the N-th distinct page, we have t = N, and this remains an invariant from that point onward. No items are discarded from the cache (main or history) until either $t_1 + b_1 = N$ or $\ell_1 + \ell_2 = 2N$. By the time the request sequence requests the 2N-th distinct page, we have either $t_1 + b_1 = N$ or $\ell_1 + \ell_2 = 2N$.

Proof. Once the request sequence requests the N-th distinct page, it is obvious that we will have t = N, since until then, no item is evicted from $T_1 \cup T_2 \cup B_1 \cup B_2$. (Note that REPLACE only moves items from the main part to the history, i.e., from $T_1 \cup T_2$ to $B_1 \cup B_2$.) Also, until then, p does not change. From that point forward, the algorithm never evicts any item from $T_1 \cup T_2$ without replacing it with some other item. Thus, t = N is an invariant once it is satisfied. The history remains empty until the main cache is filled, i.e., t = N.

From the pseudocode it is clear that items are discarded from the cache in statements 14, 17, and 21; no discards happen from the cache until either $t_1 + b_1 = N$ (statement 12) or $\ell_1 + \ell_2 = 2N$ (statement 20). If $\ell_1 + \ell_2 = 2N$ is reached, since $t_1 + b_1 \leq N$, we are guaranteed that $t_2 + b_2 \geq N$ and $b_1 + b_2 = N$, both of which will remain true from that point onward. Thus, by the time the 2N-th distinct page is requested, we have reached either $t_1 + b_1 = N$ or $\ell_1 + \ell_2 = 2N$.

We assume that request σ is processed in two distinct steps: first when **O**PT services the page request and, next when **A**RC services the request. We will show that inequality (11) is satisfied for each of the two steps.

Step 1: Opt services request σ

Since only **O**PT acts in this step, $C_A = 0$, and the contents of **A**RC's cache does not change. There are two possible cases: either **O**PT faults on σ or it does not. Assume that page x is requested on request σ .

If **O**PT does not fault on this request, then $C_O = 0$. Since the contents of the cache maintained by **O**PT does not change, and neither do the lists L_1 and L_2 , we have $\Delta \Phi = 0$, and $C_A(\sigma) + \Delta \Phi \leq 4N \cdot C_O(\sigma) \leq 0$.

If **O**PT faults on request σ , then $C_O = 1$. The contents of the cache maintained by **O**PT does change, which will affect the potential function. **O**PT will bring in page x into its cache. Assume that it evicts page y from its cache. The entry of page x into **O**PT's cache can only decrease the potential function. The exit of page y from **O**PT's cache can increase the potential function by at most 4N. The reason is as follows. Since the sum of b'_1, b'_2, t'_1, t'_2 cannot exceed the size of **O**PT's cache, we have $0 \le b'_1 + t'_1 + b'_2 + t'_2 \le N$. Since $b'_1 + 2t'_1 + 3b'_2 + 4t'_2 \le 4(b'_1 + t'_1 + b'_2 + t'_2)$, the left hand side cannot decrease by more than 4N. Thus, $C_A(\sigma) + \Delta \Phi_1 \le 4N$, proving inequality (11).

Step 2: Arc services request σ

There are four possible cases, which correspond to the four cases in ARC's replacement algorithm. Case 1 deals with the case when ARC finds the page in its cache. The other three cases assume that ARC faults on this request because the item is not in $T_1 \cup T_2$. Cases 2 and 3 assume that the missing page is found recorded in the history in lists B_1 and B_2 , respectively. Case 4 assumes that the missing page is not recorded in history.

Case I: ARC has a page hit.

Clearly, $C_A = 0$. We consider several subcases. In each case, the requested page will be moved to $MRU(T_2)$ while shifting other pages in T_2 down.

- Case I.1 If the requested page is in T_1' , the move of this page from T_1' to T_2' implies $\Delta t_1' = -1$; $\Delta t_2' = +1$ and $\Delta \Phi = -(2 \cdot \Delta t_1' + 4 \cdot \Delta t_2') = -2$.
- Case I.2 If the requested page is in T_2' , the move of this page to $MRU(T_2)$ does not change the set of items in T_2' . Thus, $\Delta t_1' = \Delta t_2' = 0$ and $\Delta \Phi = 0$.
- Case I.3 If the requested page is in $T_1 T'_1$, then $\Delta t'_1 = 0$; $\Delta t'_2 = +1$ and $\Delta \Phi = -4$. One subtle point to note is that moving x from $T_1 T'_1$ could potentially increase t'_1 if the following conditions are met: x is located just below T'_1 in T_1 , it is not in **O**PT's cache, and the items in T_1 immediately below it are in **O**PT. However, x is already in **O**PT's cache and there must be some item above it in T_1 that is not in **O**PT.
- Case I.4 If the requested page is in $T_2 T_2'$, then $\Delta t_2' = +1$ and $\Delta \Phi = -4$. The subtle point mentioned in Case I.3 also applies here.

Next we will analyze the three cases when the requested page is not in ARC's cache. Since $C_A = 1$, the change in potential must be at most -1 in order for inequality (11) to be satisfied. We make the following useful observations in the form of lemmas.

Lemma 6. If **A**RC has a miss and if the page is not in **A**RC's history, we have $\ell' = t'_1 + t'_2 + b'_1 + b'_2 < N$. Consequently, we also have $\ell'_1 < N$ and $\ell'_2 < N$.

Proof. Since **OPT** has just finished serving the request, the page is present in the cache maintained by **OPT** just before **A**RC starts to service the request. If **A**RC has a miss, there is at least one page in the cache maintained by **OPT** that is not present in the cache maintained by **A**RC, implying that l' < N. By definition, $\ell' = \ell'_1 + \ell'_2 = t'_1 + t'_2 + b'_1 + b'_2$. Thus, the lemma holds.

Lemma 7. A call to procedure REPLACE either causes an element to be moved from T_1 to B_1 or from T_2 to B_2 . In either case, the change in potential due to REPLACE, denoted by $\Delta\Phi_R$, has an upper bound of 1.

Proof. Procedure REPLACE is only called when ARC has a page miss. Clearly, it causes an item to be moved from T_1 to B_1 or from T_2 to B_2 . If that item is in T_1' (or T_2'), then $T_1 = T_1'$ ($T_2 = T_2'$, resp.) and the moved item becomes part of B_1' (B_2' , resp.). Because the coefficients of b_1' and b_2' and b_2' resp.) differ by 1, we have $\Delta \Phi_R = +1$. On the other hand, if that element is in $T_1 - T_1'$ ($T_2 - T_2'$, resp.), then B_1' (B_2' , resp.) was empty before the move and remains empty after the move, and thus, $\Delta \Phi_R = 0$.

Lemma 8. On an **A**RC miss after phase P(0), if $T_1 = T'_1$ then the REPLACE step will not move a page from T'_2 to B_2 . On the other hand, if $T_2 = T'_2$ then REPLACE will not move a page from T'_1 to B_1 .

Proof. In an attempt to prove by contradiction, let us assume that $T_1 = T_1'$ and $T_2 = T_2'$ are simultaneously true and **A**RC has a miss. By Lemma 5, we know that after phase, we have $t = t_1 + t_2 = N$, which by our assumption means that $t_1' + t_2' = N$; this is impossible by Lemma 6. Thus, if $T_1 = T_1'$, then $T_2 \neq T_2'$. Consequently, if $LRU(T_2)$ is moved to B_2 , this item cannot be from T_2' . By a symmetric argument, if $T_2' = T_2$, then $T_1 \neq T_1'$, and $LRU(T_1)$ is not in T_1' .

Case II: ARC has a miss and the missing page is in B_1

Note that in this case the value of p will change by +1, unless its value equals N, in which case it has no change. Thus $\Delta p \leq 1$.

If the missing item is in B'_1 , then $\Delta b'_1 = -1$ and $\Delta t'_2 = +1$. Adding the change due to REPLACE, we get

$$\begin{array}{rcl} \Delta\Phi & \leq & 1 - (\Delta b_1' + 4 \cdot \Delta t_2') + \Delta\Phi_R \\ & < & -1 \end{array}$$

If the missing item is in $B_1 - B_1$, then we have $\Delta t_2 = 1$ and $\Delta b_1 = 0$. Thus, we have

$$\Delta \Phi \leq 1 - (\Delta b_1' + 4 \cdot \Delta t_2') + \Delta \Phi_R$$

$$\leq -2$$

Case III: ARC has a miss and the missing page is in B_2 .

Note that in this case the value of p will change by -1, if its value was positive, otherwise it has no change. Thus $\Delta p \leq 0$.

If the requested item is in B_2' , then $\Delta t_2' = 1$, and $\Delta b_2' = -1$. Thus, we have

$$\begin{array}{rcl} \Delta\Phi & = & \Delta p - (3\cdot\Delta b_2' + 4\cdot\Delta t_2') + \Delta\Phi_R \\ & \leq & 0 \end{array}$$

But this is not good enough since we need the potential change to be at most -1. When $\Delta p=-1$, then we get the required inequality $\Delta\Phi \leq -1$. Clearly, the difficulty is when $\Delta p=0$, which happens when p=0. Since the missing item is from b_2' , it implies that B_2' is non-empty and $T_2'=T_2$. By Lemma 8 above, there must be at least one item in T_1-T_1' , which means that means that $t_1>0$. As per the algorithm, since T_1 is non-empty and p=0, we are guaranteed to replace $LRU(T_1)$, and not an element from T_1' . Therefore, REPLACE will leave t_1' and b_1' unchanged, implying that $\Delta\Phi_R=0$. Thus, we have

$$\Delta\Phi = \Delta p - (3 \cdot \Delta b_2' + 4 \cdot \Delta t_2') + \Delta\Phi_R$$

< -1

If the requested item is from $B_2 - B_2'$, then $\Delta t_2' = 1$, and $\Delta b_2' = 0$. Thus, we have

$$\begin{array}{rcl} \Delta\Phi & \leq & \Delta p - (4 \cdot \Delta t_2') + \Delta\Phi_R \\ & \leq & -3 \end{array}$$

Case IV: ARC has a miss and the missing page is not in $B_1 \cup B_2$

We consider two cases. First, when $\ell_1 = N$, ARC will evict the $LRU(L_1)$. Since by Lemma 6, $\ell'_1 < N$, we know that for this case, b'_1 remains unchanged at 0 and $\Delta t'_1 = +1$. Thus,

$$\Delta \Phi \leq -(2 \cdot \Delta t_1') + \Delta \Phi_R$$

$$< -1$$

On the other hand, if $\ell_1 < N$, then ARC will evict the $LRU(L_2)$. Again, if the cache is full (i.e., $t_1 + t_2 = N$ and $\ell_1 + \ell_2 = 2N$), then we know that $\ell_2 > N$, which means that $L'_2 \neq L_2$ and $LRU(L_2)$ is not in L'_2 . Thus, deletion of $LRU(L_2) = LRU(B_2)$ will not affect b'_2 or any of the other quantities in the potential function. Then comes the REPLACE step, for which a bound has been proved earlier. Finally, a new item is brought in and placed in $MRU(T_1)$. Thus $\Delta t'_1 \leq 1$. Putting it all together, we have

$$\Delta \Phi \leq -(2 \cdot \Delta t_1') + \Delta \Phi_R
\leq -1$$

Wrapping up the proof of Theorem 8 Tying it all up, we have shown that inequality (11) holds for every request made after the cache is full, i.e.,

$$C_A(\sigma) + \Delta \Phi \le 4N \cdot C_O(\sigma).$$

If we assume that the caches started empty, then the initial potential is 0, while the final potential can be at most 4N. Thus, we have

$$C_A(\sigma) \le 4N \cdot C_O(\sigma) + 4N,$$

thus proving Theorem 8.

4.6 Analyzing the Competitiveness of CAR

Next, we analyze the competitiveness of CAR. The main result of this section is the following:

Theorem 9. Algorithm CAR is 18N-competitive.

Proof. Let $P_X[q]$ be the position of page q in an arbitrary ordered sequence of pages X. When the set is obvious, we will drop the subscript and denote $P_X[q]$ simply by P[q]. The set of history pages B_1 and B_2 will be treated as an ordered sequence of pages ordered from its LRU position to its MRU position. The set of main pages T_1^0 (resp., T_2^0 , T_1^1 , and T_2^1) will be treated as an ordered sequence of unmarked (resp., unmarked, marked, and marked) pages in T_1 (resp, T_2 , T_1 , and T_2) ordered from head to tail. Let $\mathbf{O}pt$ and $\mathbf{C}ar$ be the set of (main and history) pages stored in the caches for algorithms \mathbf{OPT} and $\mathbf{C}AR$ respectively. Let $D = (T_1 \cup T_2 \cup B_1 \cup B_2) \setminus \mathbf{O}pt$. Thus D consists of pages in $\mathbf{C}ar$ but not in $\mathbf{O}pt$.

We associate each page with a rank value R[q], which is defined as follows:

$$R[q] = \begin{cases} P_{B_1}[q] & \text{if } q \in B_1 \\ P_{B_2}[q] & \text{if } q \in B_2 \\ 2P_{T_1^0}[q] + b_1 & \text{if } q \in T_1^0 \\ 2P_{T_2^0}[q] + b_2 & \text{if } q \in T_2^0 \\ 3N + 2P_{T_1^1}[q] + b_1 & \text{if } q \in T_1^1 \\ 3N + 2P_{T_2^1}[q] + b_2 & \text{if } q \in T_2^1 \end{cases}$$

$$(12)$$

Finally, we define the potential function as follows:

$$\Phi = \left(\frac{1}{N - N_O + 1}\right) \left(p + 2(b_1 + t_1) + 3\sum_{q \in D} R[q]\right)$$
(13)

The initial value of Φ is 0. If the following inequality (14) is true for any request σ , where $\Delta\Phi$ is the change in potential caused by serving the request, then when summed over all requests, it proves Theorem 9.

$$C_{\mathbf{C}ar}(\sigma) + \Delta\Phi \le \left(\frac{18N}{N - N_O + 1}\right)C_{\mathbf{O}pt}(\sigma). \tag{14}$$

As before, we assume that request σ is processed in two distinct steps: first when **O**PT serves and, next when **C**AR serves. We will show that inequality (14) is satisfied for each of the two steps.

Step 1: Opt serves request σ

Since only **O**PT acts in this step, $C_{\mathbf{C}ar} = 0$, and $T_1 \cup T_2$ does not change. There are two possible cases: either **O**PT faults on σ or it does not. If **O**PT does not fault on this request, then it is easy to see that $C_{\mathbf{O}pt} = 0$ and $\Delta \Phi = 0$, thus satisfying inequality (14).

If **O**PT faults on request σ , then $C_{\mathbf{O}pt}=1$ and some page, q, is evicted from the cache maintained by **O**PT. If q is maintained by **C**AR then it follows that q will belong to D after this step and thus its rank will contribute to the potential function, which will increase by three times the rank of q. The maximal positive change in potential will occur when q is the marked head page in T_2 . In this case the rank of q is given by: $R[q] = 3N + 2P[q] + b_2$. The maximal possible values for each of the terms P[q] and b_2 will be N, hence the maximum possible rank of q will be 3N + 2N + N = 6N. Therefore resulting potential change is at most 3(6N) = 18N.

Step 2: Car serves request σ

We break down the analysis into four cases. Case 2.1 deals with the case when CAR finds the page in its cache. The other three cases assume that CAR faults on this request because the item is not in $T_1 \cup T_2$. Cases 2.2 and 2.3 assume that the missing page is found recorded in the history in lists B_1 and B_2 , respectively. Case 2.4 assumes that the missing page is not recorded in history.

Case 2.1: CAR has a page hit Clearly, the page was found in $T_1 \cup T_2$, and $C_{\mathbf{C}ar} = 0$. We consider the change of each of terms in the potential function individually.

- 1. As per the algorithm, p can only change when the page is found in history. (See lines 14 through 20 of CAR(x).) Since the page is not found in CAR's history, $\Delta p = 0$.
- 2. Neither the cache nor the history lists maintained by CAR will change. Thus, the contribution to the second term in Φ , i.e., $2(b_1 + t_1)$ does not change.
- 3. Since **O**PT has already served the page, the page is in **O**PT's cache. Therefore, even if the page gets marked during this hit, its rank value does not change. Thus, the contribution to the last term in Φ , also remains unchanged.

We, therefore, conclude that $\Delta \Phi = 0$, satisfying inequality (14).

Next we will analyze the three cases when the requested page is not in CAR's cache. Since $C_{\mathbf{C}ar} = 1$, the change in potential must be at most -1 in each case in order for inequality (14) to be satisfied. Before tackling the three cases, the following lemmas (9 and 10) are useful for understanding the potential change caused by the last term in the potential function, i.e., $\sum_{q \in D} R[q]$. It is worth pointing out that a call to REPLACE moves either an item from T_1 to T_2 to T_2 to T_3 , which is exactly the premise of Lemma 9 below

Lemma 9. When a page is moved from T_1 to B_1 (or from T_2 to B_2) its rank decreases by at least 1.

Proof. Let q be any page in T_1 . In order for q to be moved from T_1 to B_1 it must have been unmarked and located at the head of T_1 . Since $P_{T_1}[q] = 1$, the rank of q prior to the move must have been $R[q] = 2P_{T_1}[q] + b_1 = b_1 + 2$, where b_1 is the size of B_1 prior to moving q.

After q is moved to the MRU position of B_1 , $R[q] = P_{B_1}[q] = b_1 + 1$. Thus its rank decreased by 1. The arguments for the move from T_2 to B_2 are identical with the appropriate changes in subscripts.

Lemma 10. When CAR has a page miss, the term $\sum_{q \in D} R[q]$ in the potential function Φ cannot increase.

Proof. We examine the rank change based on the original location of the page(s) whose ranks changed and in each case show that the rank change is never positive. Wherever appropriate we have provided references to line numbers in Pseudocode CAR(x) from Appendix.

Case A: $q \in B_1 \cup B_2$

The rank of $q \in B_1$, which is simply its position in B_1 , can change in one of three different ways.

- 1. Some page x less recently used than q (i.e., $P_{B_1}[x] < P_{B_1}[q]$) was evicted (Line 7). In this case, it is clear that $P_{B_1}[q]$ decreases by at least 1.
- 2. The page q is the requested page and is moved to T_2 (Line 16). In this case, $q \in \mathbf{O}pt$ and hence its rank cannot affect the potential function.
- 3. Some page x is added to MRU of B_1 (Line 27). Since pages are ordered from LRU to MRU, the added page cannot affect the rank of q.

Using identical arguments for $q \in B_2$, we conclude that a miss will not increase the rank of any page in $B_1 \cup B_2$.

Case B: $q \in T_1^0 \cup T_2^0$

The rank of page $q \in T_1^0$, defined as $R[q] = 2P_{T_1^0}[q] + b_1$, may be affected in four different ways.

- 1. If page q is the head of T_1 and gets moved to B_1 (Line 27), by lemma 9, the change in rank of q is at most -1.
- 2. If an unmarked page x is added to the tail of T_1 (Line 13), then since the ordering is from head to tail, it does not affect the position of page q. Since there was no change in b_1 , it is clear that the change in R[q] is 0.
- 3. If the unmarked page $x \neq q$ at the head of T_1 is marked and moved to tail of T_2 (Line 29), then P[q] decreases by at least 1. Since the content of B_1 is unchanged, the change in $R[q] = 2P[q] + b_1$ is at most -2.
- 4. If the unmarked page $x \neq q$ at the head of T_1 is moved to B_1 (Line 29), then P[q] decreases by at least 1, and b_1 increases by 1. Hence the change in $R[q] = 2P[q] + b_1$ is at most -1.

The arguments are identical for $q \in T_2^0$. In each case, we have shown that a miss will not increase the rank of any page in $T_1^0 \cup T_2^0$.

Case C: $q \in T_1^1$

The rank of page $q \in T_1^1$, defined as $R[q] = 3N + 2P_{T_1^1}[q] + b_1$, may be affected in four different ways.

- 1. If an unmarked page x is added to the tail of T_1 (Line 13), then since the ordering is from head to tail, it does not affect the position of page q. Since there was no change in b_1 , it is clear that the change in R[q] is 0.
- 2. If the unmarked page $x \neq q$ at the head of T_1 is marked and moved to tail of T_2 (Line 29), then P[q] decreases by at least 1. Since B_1 is unchanged, the change in $R[q] = 3N + 2P[q] + b_1$ is at most z^2
- 3. If the unmarked page $x \neq q$ at the head of T_1 is moved to B_1 (Line 29), then P[q] decreases by at least 1, and b_1 increases by 1. Hence the change in $R[q] = 3N + 2P[q] + b_1$ is at most -1.

4. Next, we consider the case when the marked page q is the head of T_1 and gets unmarked and moved to T_2 (Line 29). Prior to the move, the rank of q is given by $R[q] = 3N + 2P_{T_1^1}[q] + b_1$. Since B_1 could be empty, we know that $R[q] \geq 3N + 2$. After page q is unmarked and moved to T_2 , its rank is given by $R[q] = 2P_{T_2^0}[q] + b_2$. Since $P[q] \leq N$ and $b_2 \leq N$, we know that the new $R[q] \leq 3N$. Thus, the rank of page q does not increase.

In each case, we have shown that a miss will not increase the rank of any page in T_1^1 .

Case D: $q \in T_2^1$

The rank of page $q \in T_2^1$, defined as $R[q] = 3N + 2P_{T_2^1}[q] + b_2$, may be affected in four different ways.

- 1. If an unmarked page x is added to the tail of T_2 (Lines 16, 19, or 29), and if b_2 does not change, it is once again clear that the change in R[q] is 0.
- 2. If a marked page $x \neq q$ at the head of T_2 gets unmarked and moved to the tail of T_2 (Line 36), the position of q will decrease by 1 and there is no change in b_2 . Thus R[q] changes by at most -2.
- 3. If an unmarked page x at the head of T_2 is moved to B_2 (Line 34), P[q] decreases by 1 and b_2 increases by 1. Thus R[q] changes by at most -1.
- 4. Finally, we consider the case when the marked page q is the head of T_2 and gets unmarked and moved to the tail of T_2 (Line 36). Prior to the move, the rank of q is given by $R[q] = 3N + 2P_{T_2^1}[q] + b_2$. Even if B_2 is empty, we know that $R[q] \ge 3N + 2$. After page q is unmarked and moved to T_2 , its rank is given by $R[q] = 2P_{T_2^0}[q] + b_2$. Since $P[q] \le N$ and $b_2 \le N$, we know that the new $R[q] \le 3N$. Thus, the rank of page q does not increase.

In each case, we have shown that a miss will not increase the rank of any page in T_2^1 .

The four cases (A through D) together complete the proof of Lemma 10.

We continue with the remaining cases for the proof of Theorem 9.

Case 2.2: CAR has a page miss and the missing page is in B_1 We consider the change in the potential function (defined in Eq. 13) by analyzing each of its three terms.

- 1. Value of p increases by 1, except when it is equal to N, in which case it remains unchanged. (See Line 15.) Thus, the first term increases by at most 1.
- 2. The call to REPLACE has no effect on the value of $(t_1 + b_1)$ because an item is moved either from T_1 to B_1 or from T_2 to B_2 . Since the requested page in B_1 is moved to T_2 , $(t_1 + b_1)$ decreases by 1.
- 3. By Lemma 10, we already know that the last term increases by at most 0.

Since p increases by at most 1 and the term $2(t_1+b_1)$ decreases by at least 2, the total change in the potential function, is at most -1.

Case 2.3: CAR has a page miss and the missing page is in B_2 When the missing page is in B_2 , CAR makes a call to REPLACE (Line 5) and then executes Lines 18-19. Thus, p is decremented except if it is already equal to 0. We consider two subcases: $\Delta p < 0$ and $\Delta p = 0$.

 $\Delta p < 0$: As in Case 2.2, the call to REPLACE has no effect on $(t_1 + b_1)$. Since, Lines 18-19 do not affect $T_1 \cup B_1$, the second term does not change. By Lemma 10, we know that the last term increases by at most 0. Since $\Delta p \leq -1$, the total change in the potential function, $\Delta p + \Delta 2(t_1 + b_1)$ is at most -1.

 $\Delta p = 0$: Unlike the subcase above when p decreases by 1, the change in p cannot guarantee the required reduction in the potential. We therefore need a tighter argument. We know that there is a call to REPLACE. Three cases arise and are discussed below.

- If T_1 is empty, then T_2 must have N pages, at least one of which must be in D. Also, REPLACE must act on T_2 , eventually evicting an unmarked page from head of T_2 , causing the rank of any page from $T_2 \setminus \mathbf{Opt}$ to decrease by 1.
- If T_1 is not empty and has at least one page from D, then the condition in Line 24 passes and REPLACE must act on T_1 , eventually evicting an unmarked page from head of T_1 , causing the rank of at least one page from $T_1 \setminus \mathbf{O}pt$ to decrease by 1.
- Finally, if T_1 is not empty and all its pages are in **O**PT, then T_2 must have a page $q \in D$. Since the requested page x was found in B_2 and is moved to the tail of T_2 , even though the position of q in T_2 does not change, b_2 decreased by 1 and consequently the rank of q decreases by 1.

Thus, in each case, even though neither p nor the quantity $(t_1 + b_1)$ changed, the third term involving ranks, and consequently, the potential function decreased by at least 3.

The following two lemmas are useful for Case 2.4, when the missing page is not in $T_1 \cup T_2 \cup B_1 \cup B_2$.

Lemma 11. We make two claims:

- 1. If $t_1 + b_1 = N$ and the LRU page of B_1 is evicted from the cache on Line 7, then $\sum_{q \in D} R[q]$ will decrease by at least one.
- 2. If $t_2 + b_2 > N$, and the LRU page of B_2 , is evicted from the cache on Line 9, then $\sum_{q \in D} R[q]$ will decrease by at least one.

Proof. We tacke the first claim. Assume that y is the LRU page of B_1 that is being evicted on Line 7. Then CAR must have had a page miss on $x \notin B_1 \cup B_2$, and the requested page x is added to the tail of T_1 . Since $t_1 + b_1 = N$, there is at least one page $q \in T_1 \cup B_1$ that is not in **OPT**'s cache and whose rank contributes to the potential function. First, we assume that $q \in T_1 \setminus \mathbf{Opt}$, whose rank is given by: $R[q] = 2 * P[q] + b_1$. For each of the three cases, we show that the potential function does decrease by at least 1.

- If REPLACE acts on T_1 and the unmarked head of T_1 , different from q, is moved to B_1 then the size of B_1 remains the same (because a page gets added to B_1 while another page is evicted) but the position of q in T_1 decreases by one. Therefore R[q] decreases by 2.
- If Replace acts on T_1 and q itself is moved to B_1 then by Lemma 9, R[q] decreases by at least 1.
- If REPLACE acts on T_2 , then we use the fact that a page is evicted from B_1 , and the b_1 term in R[q] must decrease by 1.

Next, we assume that $q \in B_1 \setminus \mathbf{O}pt$. Since $LRU(B_1)$ is evicted, the position of the page q will decrease by one. Thus $R[q] = P_{B_1}[q]$ must decrease by at least 1, completing the proof of the first claim in the lemma. The proof of the second claim is very similar and only requires appropriate changes to the subscripts. \square

Next we tackle the last case in the proof of Theorem 9.

Case 2.4: CAR has a page miss and the missing page is not in $B_1 \cup B_2$ We assume that CAR's cache is full (i.e., $l_1 + l_2 = 2N$). We consider two cases below – first, if $l_1 = N$ and the next when $l_1 < N$. If $l_1 = t_1 + b_1 = N$, CAR will call Replace, evict $LRU(B_1)$ and then add the requested page to the tail of T_1 . Below, we analyze the changes to the three terms in the potential function.

- Since p is not affected, the first term does not change.
- Since a page is added to T_1 and a page is evicted from B_1 , the net change in the second term is 0.
- Since the conditions of Lemma 11 apply, the total rank will decrease by at least 1.

Adding up all the changes, we conclude that the potential function decreases by at least 3.

If $l_1 < N$, CAR will call REPLACE, evict $LRU(B_2)$ and then add a page to the tail of T_1 . As above, we analyze the changes to the three terms in the potential function.

- Since p is not affected, the first term does not change.
- A page is added to T_1 and a page is evicted from B_2 hence $(t_1 + b_1)$ increases by 1.
- Since $l_2 > N$, the conditions of Lemma 11 apply, the total rank will decrease by at least 1.

Adding up all the changes, we conclude that the potential function decreases by at least 1, thus completing Case 2.4.

Wrapping up the proof of Theorem 9: Combining the four cases (2.1 through 2.4) proves that inequality (14) is satisfied when CAR serves request σ . This completes the proof of Theorem 9, establishing that the upper bound on the competitiveness of CAR is 18N.

5 Conclusions and Future Work

Adaptive algorithms are tremendously important in situations where inputs are infinite online sequences and no single optimal algorithm exists for all inputs. Thus, different portions of the input sequence require different algorithms to provide optimal responses. Consequently, it is incumbent upon the algorithm to sense changes in the nature of the input sequence and adapt to these changes. Unfortunately, these algorithms are harder to analyze. We present the analysis of two important adaptive algorithms called **A**RC and **C**AR and show that they are competitive along with proving good lower bounds on the competitiveness ratios.

Two important open questions remain unanswered. Given that there is a gap between the lower and upper bounds on the competitiveness ratios of the two adaptive algorithms, ARC and CAR, what is the true ratio? More importantly, is there an "expected" competitiveness ratio for request sequences that come from real applications? The second question would help explain why ARC and CAR perform better in practice than LRU and CLOCK, respectively.

Acknowledgments This work was partly supported by two NSF Grants (CNS-1018262 and CNS-1563883) and the NSF Graduate Research Fellowship (DGE-1038321). We are grateful to Kirk Pruhs for suggesting enhancing our results with the assumption of unequal cache sizes.

References

- [Alb96] S. Albers. Competitive online algorithms. Technical report, BRICS Lecture Series, Computer Science Department, University of Aarhus, 1996.
- [BM04] S. Bansal and D. S. Modha. CAR: CLOCK with adaptive replacement. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, FAST '04, pages 187–200, Berkeley, CA, USA, 2004. USENIX Association.
- [Cor68] F. J. Corbato. A paging experiment with the MULTICS system. Technical report, DTIC Document, 1968.
- [Fri99] M. B. Friedman. Windows NT page replacement policies. In Proceedings of the Intl. CMG Conference, pages 234–244, 1999.
- [Hoc97] D. S. Hochbaum, editor. Approximation algorithms for NP-hard problems. PWS Publishing Co., Boston, MA, USA, 1997.
- [JCZ05] S. Jiang, F. Chen, and X. Zhang. CLOCK-Pro: An effective improvement of the CLOCK replacement. In *USENIX Annual Technical Conference, General Track*, pages 323–336, 2005.
- [JIPP10] A. Janapsatya, A. Ignjatovic, J. Peddersen, and S. Parameswaran. Dueling CLOCK: adaptive cache replacement policy based on the CLOCK algorithm. In *Design*, *Automation & Test in Europe Conference & Exhibition (DATE)*, 2010, pages 920–925. IEEE, 2010.
- [JS94] T. Johnson and D. Shasha. 2Q: A low overhead high performance buffer management replacement algorithm. In *Proc. of VLDB*, pages 297–306, 1994.
- [JZ02] S. Jiang and X. Zhang. LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. In *Proc. ACM Sigmetrics Conf.*, pages 297–306. ACM Press, 2002.
- [LCK⁺01] D. Lee, J. Choi, J. H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Trans. Comput.*, 50(12):1352–1361, December 2001.
- [MM03] N. Megiddo and D. S. Modha. ARC: A self-tuning, low overhead replacement cache. In Proceedings of the 2nd USENIX Conference on File and Storage Technologies, FAST '03, pages 115–130, Berkeley, CA, USA, 2003. USENIX Association.
- [MM04] N. Megiddo and D. S. Modha. Outperforming LRU with an adaptive replacement cache algorithm. $IEEE\ Computer,\ 37(4):58-65,\ 2004.$
- [OOW93] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. SIGMOD Rec., 22(2):297–306, June 1993.
- [ST85] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. Commun. ACM, 28(2):202-208, February 1985.

6 Appendix

We reproduce the pseudocode for **A**RC and **C**AR below.

```
\overline{\mathbf{Pseudo}}\mathbf{code}: \mathbf{A}\mathbf{RC}(x)
INPUT: The requested page x
INITIALIZATION: Set p = 0 and set lists T_1, B_1, T_2, and B_2 to empty
 1: if (x \text{ is in } T_1 \cup T_2) then
                                                                                                                ▷ cache hit
        Move x to the top of T_2
 3: else if (x \text{ is in } B_1) then
                                                                                                       ▷ cache history hit
        Adaptation: Update p = \min\{p+1, N\}
                                                                                                       \triangleright learning rate = 1
 4:
        Replace()
                                                                                                \triangleright make space in T_1 or T_2
 5:
        Fetch x and move to the top of T_2
 6:
 7: else if (x \text{ is in } B_2) then
                                                                                                       ▷ cache history hit
 8:
        Adaptation: Update: p = \max\{p - 1, 0\}
                                                                                                      \triangleright learning rate = 1
        Replace()
                                                                                                \triangleright make space in T_1 or T_2
 9:
10:
        Fetch x and move to the top of T_2
                                                                                                11: else
        if (t_1 + b_1 = N) then
12:
13:
            if (t_1 < N) then
                Discard LRU item in B_1
14:
                Replace()
                                                                                                \triangleright make space in T_1 or T_2
15:
            else
16:
                Discard LRU page in T_1 and remove from cache
17:
18:
            end if
        else if ((t_1 + b_1 < N) \text{ and } (t_1 + t_2 + b_1 + b_2 \ge N)) then
19:
            if (t_1 + t_2 + b_1 + b_2 = 2N) then
20:
                Discard LRU item in B_2
21:
            end if
22:
23:
            Replace()
                                                                                                \triangleright make space in T_1 or T_2
24:
        end if
        Fetch x and move to the top of T_1
25:
26: end if
Replace()
26: if ((t_1 \ge 1) \text{ and } ((x \in B_2 \text{ and } t_1 = p) \text{ or } (t_1 > p))) then
27:
        Discard LRU page in T_1 and insert as MRU history item in B_1
28:
    else
        Discard LRU page in T_2 and insert as MRU history item in B_2
29:
30: end if
```

```
Pseudocode: CAR(x)
INPUT: The requested page x
INITIALIZATION: Set p = 0 and set lists T_1, B_1, T_2, and B_2 to empty
 1: if (x \text{ is in } T_1 \cup T_2) then
                                                                                                         ⊳ cache hit
 2:
        Mark page x
                                                                                                        ▷ cache miss
 3: else
       if (t_1 + t_2 = N) then
                                                                           ⊳ cache full, replace a page from cache
 4:
 5:
           Replace()
                                                                                          \triangleright make space in T_1 or T_2
           if ((x \notin B_1 \cup B_2) \text{ and } (t_1 + b_1 = N)) then
 6:
               Discard LRU page in B_1
 7:
           else if ((x \notin B_1 \cup B_2) \text{ and } (t_1 + t_2 + b_1 + b_2 = 2N)) then
 8:
               Discard LRU page in B_2.
 9:
10:
           end if
        end if
11:
       if (x \notin B_1 \cup B_2) then
                                                                                                        ▷ cache miss
12:
           Insert x at the tail of T_1; Unmark page x
13:
        else if (x \in B_1) then
                                                                                                 ▷ cache history hit
14:
           ADAPTATION: Update p = \min\{p+1, N\}
                                                                                                \triangleright learning rate = 1
15:
16:
           Move x to the tail of T_2; Unmark page x
                                                                                                 ▷ cache history hit
17:
           Adaptation: Update: p = \max\{p-1, 0\}
                                                                                                \triangleright learning rate = 1
18:
           Move x to the tail of T_2; Unmark page x
19:
20:
        end if
21: end if
Replace()
22: found = false
23: repeat
24:
        if (t_1 \ge \max\{1, p\}) then
           if (head page in T_1 is unmarked) then
25:
               found = true
26:
               Discard head page in T_1 and insert as MRU history item in B_1
27:
28:
               Unmark head page in T_1, move page as tail page in T_2, and move head of T_1 clockwise
29:
           end if
30:
        else
31:
           if (head page in T_2 is unmarked) then
32:
33:
               found = true
               Discard head page in T_2 and insert as MRU history item in B_2
34:
35:
               Unmark head page in T_2, and move head of T_2 clockwise
36:
37:
           end if
        end if
38:
39: until (found)
```