Indonesian Journal of Electrical Engineering and Computer Science
Vol. 38, No. 1, April 2025, pp. 417∼427
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v38.i1.pp417-427

An enhanced least recently used page replacement algorithm

Afaf Tareef[1], Khawla Al-Tarawneh[2], Omar Alhuniti[2]
[1]Faculty of Information Technology, Mutah University, Karak, Jordan
[2]King Abdullah II School for Information Technology, University of Jordan, Amman, Jordan

Article Info

Article history:

Received Mar 5, 2024
Revised Sep 23, 2024
Accepted Oct 7, 2024

Keywords:

Enhanced least recently used
First in first out
Least frequently used
Least recently used
Most recently used
Page fault
Page replacement

ABSTRACT

Page replacement algorithms play a crucial role in enhancing the performance issue brought on by variations in processor speeds and memory by effectively removing pages from computer memory to improve overall efficiency. The majority of these algorithms can address the page replacement problems, but their implementation is challenging. This paper introduces a new efficient page replacement algorithm, i.e., enhanced least-replacement (E-LRU) based on two introduced features used to select the victim page. By incorporating elements of traditional algorithms such as first in first out (FIFO) and least recently used (LRU), E-LRU presents itself as a new approach with potential benefits for memory management. This study evaluates the effectiveness of E-LRU in reducing power consumption by reducing cache faults and compares its performance to existing algorithms in various settings. The results provide insight into the advantages and disadvantages of E-LRU and essential perspectives on its potential benefits for contemporary memory management algorithms. Furthermore, the study puts E-LRU into the perspective of evolving algorithms and provides directions for future investigation and improvement in the ever-changing field of memory management. The study proved that E-LRU works better than FIFO and LRU algorithms.

Corresponding Author:

Afaf Tareef
Faculty of Information Technology, Mutah University
Karak (61710), Jordan
Email: a.tareef@mutah.edu.jo

1. INTRODUCTION

Memory management stands out as a critical component of the operating system, involving the division of main memory into fixed-size frames [1]. Simultaneously, the virtual address space undergoes division into fixed-size blocks known as pages. When a page fault occurs in the main memory, signifying the need for a new page, the operating system must determine which existing page to replace, making space for the incoming page. This decision arises when an executing process references a specific page, leading to a search in the main memory. If the required page is absent, a page fault occurs, potentially necessitating the removal of pages due to limited storage. Herein lies the role of page replacement algorithms in minimizing the fault rate by strategically selecting the optimal victim page for removal from main memory [2].

The page replacement algorithm determines which memory page to replace or remove, a process known as swap out. This replacement occurs when a page fault transpires, indicating the need to load a new page into the main memory when no available space (frame) exists. The approach of page replacement algorithms is direct: when a new page requires loading into main memory, and no free frame is available, the algorithm selects which pages to replace, aiming to decrease the overall number of fault pages. The algorithm selects a page not currently in use, freeing it to accommodate the required page [3].

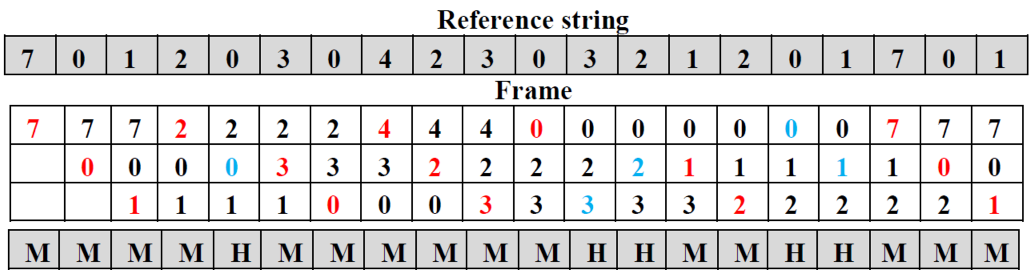
There are many page replacement algorithms introduced in the literature, e.g., [4]–[9]. The evaluation of such algorithms requires executing them on a particular sequence of memory references and calculating the number of page faults. A lower number of page faults indicates a more efficient algorithm for that specific scenario [10], [11]. The majority of these algorithms are inefficient and challenging to execute. To this context, our study proposes an efficient least recently used (LRU)-based page replacement strategy to cluster cache pages. More fairness in the victim page selection is achieved by introducing a new characteristic, i.e., the total number of references (TNR), which improves cache memory efficiency.

2. BACKGROUND

There are four types of replacement algorithms. Four subsections below described them with simple examples:

2.1. First in first out

The most straightforward page replacement strategy is the first in first out (FIFO) algorithm. To keep track of all the pages that are currently in main memory, the operating system (OS) typically implements queue management. This queue is set up to accommodate several pages, with the most recent page accessed at the back and the oldest page at the front. The replacement of the page that has been in the main memory for the longest is the basic idea behind FIFO [4]. FIFO algorithm focuses on the length of time a page has been in the main memory rather than how much the page has been used in the main memory. For more details, Figure 1 illustrates an example of the FIFO algorithm. Consider the page reference string of size 20: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 with frame size 3 (i.e., maximum three pages in a frame) as shown in Figure 1. According to the figure, the total page faults (M) equal to 15 and total page hits (H) is five.



Reference string

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Frame

7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
		1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
M	M	M	M	H	M	M	M	M	M	M	H	H	M	M	H	H	M	M	M

Figure 1. FIFO representation for three frames, M is fault (fault), and H is hit

In the initial phase, the first three slots of the memory table are vacant, leading to three-page faults as references 7, 0, and 1 fill these empty slots based on their sequential arrival. Subsequently, when reference number 2 is introduced, it is not existing in the memory, resulting in a page fault. The algorithm replaces the oldest page in main memory, reference number 7, as evidenced by its presence in the first frame and being the oldest reference. As reference 0 arrives, it finds a place in the main memory, occupying the second frame, leading to a page hit with no replacement. This pattern continues for the subsequent string references; when page 3 arrives and is not existing in the memory, a page fault occurs, resulting in the replacement of reference 0 and reference 1 by reference 4. Then, the same process will be repeated until all pages complete their traversing.

2.2. Least frequently used

The least frequently used (LFU) page replacement algorithm is one of the prevalent replacement algorithms; the main idea of the LFU algorithm is that the page with the minor visits in a given period is removed. When a page must be allocated or replaced in the main memory, and all the frames are fully used, the LFU will choose the least frequently used page in LFU. If two or more reference pages have the same frequency, perform the FIFO method on these reference pages that have the same frequency and remove the oldest page among them [5].

For more details, Figure 2 illustrates an example of LFU algorithm. Consider the page reference string of size 20: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 with frame of size 3. The total page faults (M) is 13, and total page hits (H) is 7. In the beginning, the first three slots are empty, so directly, three-page faults will appear to fill the first three slots in the table. Specifically, the references 7, 0, and 1 will be allocated in the empty frames based on their arrival, and the number of frequencies will be incremented.

When page 2 comes which is not available in the main memory, a page fault occurs. Since all the frequencies are equal in this case, the FIFO method will replace the oldest page in memory (i.e., page 7) by the new page (i.e., page 2). The frequencies for each page will be then decreased by 1 for page 7, and increased by 1 for page 2. Next, when page 0 comes, it will be found in the main memory in the second frame, so, a page hit occurs and no replacement occurs. Thus, the frequency for page 0 is increased by 1.

When page 3 comes, it is not existing in the main memory, so a page fault occurs, then page 3 replaces page 1. Because two pages shared the same frequencies, pages 1 and 2, the FIFO method is utilized and change the frequencies for each page. The same steps are repeated until all pages complete their traversing. Moreover, in the end, each page's frequencies will be as follows in Figure 2, which shows the previous steps.

Reference string

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Frames:

7	7	7	2	2	2	2	4	4	3	3	3	3	3	3	3	1	7	7	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	1	2	2	2	2	2	2

M	M	M	M	H	M	H	M	M	M	H	H	H	M	M	H	M	M	H	M

Figure 2. LFU representation for three frames, M is fault (fault), and H is hit

2.3. Least recently used

LRU replacement algorithm associates with each page the time of that page's last use. That means this algorithm maintains record of reference page usage across a short period, unlike the FIFO replacement algorithm. The main idea of LRU is assuming that the pages that have been most heavily used in the past are most likely to be used heavily in the future, too [6]. So, this page is replaced first. When a page must be allocated or replaced in the main memory, and all the frames are complete, the LRU will choose the page that has not been used for the most extended period. Figure 3 illustrates an example of LRU, considering the page reference string of size 20: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 with frame size 3. Here, the total page faults (M) is 12, and the total page hits (H) is 8.

Reference string

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Frames:

7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7

M	M	M	M	H	M	H	M	M	M	M	H	H	M	H	M	H	M	H	H

Figure 3. LRU representation for three frames, M is fault (fault) and H is hit

At the outset, with the first three slots unoccupied, three-page faults manifest directly, filling the initial frames in the table as references 7, 0, and 1 are assigned to the empty frames based on their arrival. Subsequently, when page 2 is introduced, its absence in the main memory triggers a page fault. In the context of LRU, the selection criteria involve identifying the least recently used page that is farthest away from the page intended for allocation in the main memory, thus, page 7 is replaced by page 2. As page 0 enters, finding a place in the main memory and occupying the second frame results in a page hit, signifying no replacement.

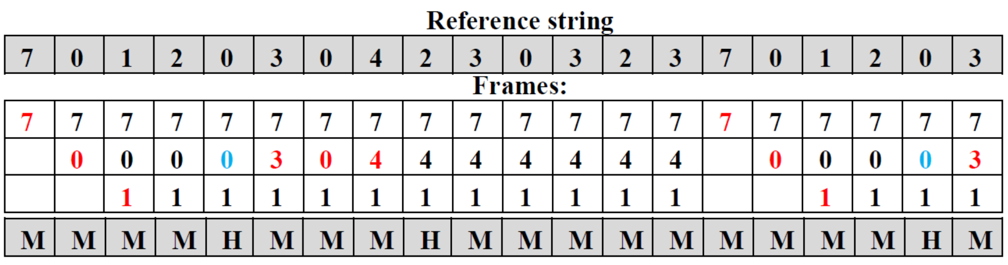
The arrival of page 3, which is not exist in memory, initiates another page fault, replacing page 1, identified as the least recently used page among the existing references. Likewise, when page 4 arrives and is not in main memory, a page fault ensues, leading to the replacement of page 2 in this instance. Then, repeat the same steps until all pages complete their traversing.

2.4. Most recently used

It is a replacement algorithm that assigns a timestamp to each page, indicating the time of its most recent use. This algorithm employs a mechanism to monitor the most recently used page within a brief time-frame, as opposed to the FIFO replacement algorithm. The fundamental principle of the most recently used (MRU) algorithm is to give priority to the pages that have been accessed most recently, under the assumption that they will continue to be heavily used in the immediate future. As a result, the page that was MRU is replaced first [7].

The MRU replacement method exhibits comparable behavior to the optimal replacement strategy. When the main memory is full and a page needs to be allocated or replaced, MRU selects the page that has been accessed most recently. Figure 4 explains this algorithm, given a page reference string consisting of 20 elements: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3, and a frame size of 4. Based on Figure 4, the total number of page faults (M) is 12, while the total number of page hits (H) is 2. At first, when the first four slots are empty, four-page errors happen when references 7, 0, 1, and 2 are allocated to the vacant frames according to their arrival. And also, no need to replacement for 0 and hit it, when page 3 is inserted, its absence in the main memory causes a page fault, resulting in the replacement of the most recently used page (i.e., page 0).

When page 0 is placed in the second frame, it causes a page hit, indicating no replacement is needed. Upon the arrival of page 3, which is not stored in memory, a page fault occurs. This results in the replacement of page 1, which is determined to be the most recently accessed page among the existing references. Likewise, if page 4 is not currently stored in memory, a page fault occurs, resulting in the substitution of page 2. Continue executing these instructions repeatedly until all pages have finished traversing.



Reference string

7	0	1	2	0	3	0	4	2	3	0	3	2	3	7	0	1	2	0	3

Frames:

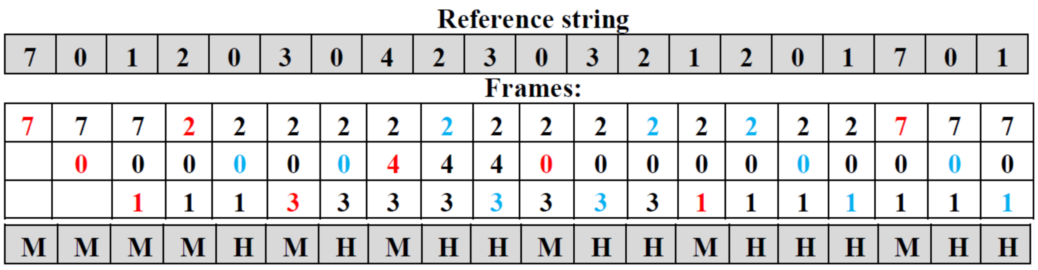
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
	0	0	0	0	3	0	4	4	4	4	4	4	4		0	0	0	0	3
		1	1	1	1	1	1	1	1	1	1	1	1			1	1	1	1
M	M	M	M	H	M	M	M	H	M	M	M	M	M	M	M	M	M	H	M

Figure 4. MRU representation for four frames, M is fault (fault), and H is hit

2.5. Optimal page replacement algorithm

The optimal page replacement (OPR) algorithm is the most efficient page replacement algorithm, resulting in the fewest page faults [8], [9]. The clairvoyant replacement algorithm, often known as Belady's optimal page replacement policy, is a method used for page replacement. This algorithm replaces pages in the memory that will be referenced farthest in the future, i.e., the pages that will not be used for the longest period of time. Figure 5 provides a visual representation of the OPR algorithm, offering additional information. Let's examine a page reference string of length 20: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1, and the frame size is set to 3. According to Figure 5, the total page faults (M) is 9, and the total page hits (H) is 11.

Initially, with the first three slots unoccupied, three-page faults occur directly, filling the initial three frames in the table as references 7, 0, and 1 are assigned to the empty frames based on their arrival order. Subsequently, as page 2 enters and is absent from the memory, a page fault ensues, replacing page 7. The OPR algorithm, employed in this context, seeks the least likely used page for the longest time in the future; thus, page 7, identified as least valuable, is replaced by page 2. Upon the arrival of page 0, no replacement required. However, when page 3 arrives and is not exist in main memory, another page fault occurs, leading to replacing page 1, which is recognized as the least useful for future references. This cycle repeats the same steps until all pages conclude their traversal.



Reference string

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Frames:

7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
		1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1

M	M	M	M	H	M	H	M	H	H	M	H	H	M	H	H	H	M	H	H

Figure 5. OPR representation for three frames, M is fault (fault), and H is hit

2.6. Other page replacement algorithms

Recently, some researches propose new page replacement method [12]–[18]. For instance, Arya *et al.* [12] propose modified page replacement algorithm based on block reading of pages. The model retrieves a set of pages equal to the number of frames allotted for a process. The lazy evaluation method (LEM) was introduced in [13] as a novel cache management replacement algorithm capable of effectively managing long-term re-reference patterns. LEM defers the assessment of the reference count of a page until it is removed from the cache, at which point it determines whether to retain it or not based on a predetermined threshold. The researchers assess the performance of LEM through simulation using traces from the SPEC CPU2006 benchmark. The results demonstrate that LEM surpasses previous methods like LRU and adaptive replacement cache (ARC).

Another page replacement algorithm called fuzzy clustering based page replacement algorithm (FCPRA) is proposed in [14], [19], where fuzzy C-means (FCM) algorithm is utilized to select a page of the lowest priority cluster and the largest login order based on three features: novelty, frequency, and reference rate. Muthusundari *et al.* [15], novel page replacement mechanism known as buffer-based page replacement (BBPR) is discussed. This mechanism effectively minimizes the occurrence of page faults in memory management. The BBPR algorithm utilizes a buffer to store two additional inputs from the reference string prior to changing the pages in the frames. This enables BBPR to enhance the success rate and prevent superfluous page substitutions. The paper assesses the performance of BBPR through simulation using various reference strings and compares it to other algorithms such as FIFO, LRU, and optimal page replacement (OPT). The findings indicate that BBPR effectively decreases the occurrence of page faults by 5%.

Banerjee *et al.* [17] introduces a novel hybrid page replacement algorithm (HPRA) for real-time systems, integrating FIFO, LRU, and optimal algorithms with an anti-variant for monitoring page duplication. Performance evaluation, comparing HPRA to an industry-standard page replacement algorithm, reveals its effectiveness in minimizing page faults in real-time systems. Lee *et al.* [18] introduced an adaptive page replacement (APR), which effectively manages looping access patterns in scientific applications. The APR algorithm identifies loops in real time by leveraging the data stored in the virtual memory subsystem of the operating system and adjusts the page replacement policy accordingly. The study assesses the effectiveness of APR through trace-driven simulation using traces derived from the SPLASH-2x benchmark. The results demonstrate that APR surpasses previous methods, such as CLOCK, in terms of performance.

Several studies conduct a performance analysis of existing page replacement methods, including FIFO, LRU, MRU, and OPR [20], [21]. The performance is evaluated based on various sequences of page references and quantifying the occurrence of page faults. It is concluded that OPR performs the best. However, this approach is costly, limited to a few specific operating systems, and difficult to implement because the operating system cannot predict future reference chains [9].

3. RESEARCH METHOD

The algorithm being presented is derived from the traditional LRU algorithm, and it is named enhanced LRU (E-LRU). Nevertheless, this approach differs from the conventional LRU algorithm due to the inclusion of a useful characteristic, which is the TNR. Furthermore, this algorithm incorporates the notion of modified bit, i.e., M, which is initially assigned to zero for all reference string, and modified to one in case of page hit occurrence. The proposed algorithms is described in details in the following steps:

– Step 1: the proposed technique begins by determining the TNR for each page reference and assigns zeros to all M bits. Subsequently, in case of a page fault, our technique examines the TNR values for every page, and the page with the minimal TNR value will be chosen for eviction from the main memory, and it might be referred to as a victim page.
– Step 2: each page table entry now contains a modified reference. The proposed approach will initialize the modified reference/bit of each page to zero, denoted as M=0. Whenever the contents of a page in the main memory are modified, the value of M will be set to one, indicating that M=1 for that particular page.
– Step 3: in the event that two or more reference pages in the same slot have the same minimum TNR value and M=0 for each page, handle it in a manner consistent with the LRU method.
– Step 4: in the event that there are multiple reference pages in the same slot with the same minimum TNR value and any of these pages are modified, always replace the modified page with a page that has M = 1.
– Step 5: if the minimum TNR value is shared among specific fractions of pages and many or all of them have been amended, replace the most recently modified page with a replacement. Additionally, reset the changed bit for every page and assign it a value of zero (i.e., M=0).

It is important to observe that whenever a modified page is replaced, the modified bits for each table entry will be reset to 0, denoted as M=0. Continuously perform these procedures until all pages have finished being traversed. To further explain the proposed E-LRU algorithm, Figure 6 shows an example of the proposed E-LRU algorithm using a page reference string of length 20: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1. First, the total number of references, denoted as TNR, is computed for each page reference. The modified bit for each page is also set to zero, represented as M = 0.

Reference string

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Frame

7	7	7	2	2	3	3	4	2	2	2	2	2	2	2	2	2	7	7	7
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	1	1	3	3	3	3	1	1	1	1	1	1	1
M	M	M	M	H	M	H	M	M	M	H	H	H	M	H	H	H	M	H	H

Figure 6. E-LRU representation for three frames, M is fault, and H is hit

Initially, the first three slots in the table are empty, leading to the occurrence of three-page faults. These page faults are responsible for filling the first three slots (frames). Consequently, the references 7, 0, and 1 are assigned to the empty frames based on their arrival. The changed bit for each page will be reset to zero, denoted as M=0. When page 2 is requested and not found in the memory, a page fault occurs. As a result, page 2 replaces page 7, which has the lowest TNR value and is only referenced twice. The modified bit for page 2 is also set to 1, as mentioned earlier in the previous steps. Page 0 is loaded into the main memory and allocated in the second frame. In this scenario, a page hit takes happen without any replacement. When page 3 is requested but not exist in main memory, a page fault occurs, resulting in the replacement of page 2 with page 3. Since the minimum TNR value is expected on pages 1 and 2, they possess identical TNR values. In this scenario, it is necessary to examine the parameter M, which represents the modified bit for page 2 and is equal to 1 (M=1). Consequently, page 3 should be promptly replaced with page 2, as stated in step 4. Subsequently, assign a zero value to the changed bit for every page (M=0). When page 4 is requested but not exist in the main memory, a page fault occurs, and page 4 replaces page 3. Because page 3 has the lowest TNR value, it is necessary to assign a value of M=1 to the modified bit of page 4. The same action will be taken when page 2 is encountered. When the second occurrence of page 3 in the reference string is encountered, it is not existing in main memory, resulting in a page fault. Consequently, page 3 replaces page 1. Since the minimum TNR value is expected on pages 1 and 2, they possess identical TNR values. Furthermore, the modified bit for page 2 equals zero (M=0), which applies to page 1. Therefore, it is necessary to execute the LRU algorithm and substitute the page with the lowest usage, specifically page 1 as indicated in step 3. As shown in Figure 6, there have been ten-page faults and an equal number of page hits.

4. RESULTS AND DISCUSSION

The proposed E-LRU algorithm was constructed using Python [22] on the well-known Google Colab environment in order to evaluate the suggested algorithm empirically. The proposed algorithm was executed with different sizes of randomly generated strings with a length (1000, 1500, 2000), each time tested on three different sizes of frames (3, 5, 7, and 9). The current algorithms, such as LRU, MRU, FIFO, and optimal page replacement algorithms, have handled the same set of strings with equal frames [23], [24]. Various aspects, such as the choice of replacement algorithm, the size of the frame, and the degree of locality of reference for cache queries, influence the occurrence of page faults. The hit ratio percentage (HITR) can be defined as (1) [25].

$$HITR = \frac{H}{H+M} \times 100 \tag{1}$$

To clarify further, let's consider a scenario where the total number of hits for a specific replacement procedure is 8, and the total number of faults is 12. In this case, HITR can be calculated as $8/(8 + 12) \times 100 = 40\%$. The hit ratio was determined by calculating the average count of page faults and hits for each replacement for one hundred runs. The results have been reported in Tables 1 to 4.

Table 1 displays a comparison of total hit for several algorithms while Table 2 displays the hit ratios for different string lengths and frame sizes. In general term, the overall number of hits for all methods increases when the frame size is increased from 3 to 9. This is expected since a larger frame size means more pages can be stored in memory, which decreases the demand for page replacements and thus increases the number of hits. As shown in the tables, the proposed method generally performs better than FIFO, LFU, LRU, and MRU, but it's not as efficient as OPR. For instance, with a string length of 1000, the total hits for FIFO, LFU, LRU, and MRU was improved with increasing the frame size to 866, 900, 893, and 893, respectively, whereas the total hits was improved by our proposed E-LRU to 904. Table 3 shows a comparison of the total fault while Table 4 shows the faults ratio across different string lengths and frame sizes. In almost every case, the proposed E-LRU performs better than FIFO, LFU, and MRU, while being slightly behind OPR.

Table 1. Comparison of total hits for several algorithms

String length	Frame size	FIFO	LFU	LRU	MRU	OPR	ELRU
1000	3	311	312	287	287	523	314
1000	5	460	489	496	493	718	500
1000	7	690	701	710	704	859	712
1000	9	866	900	893	893	958	904
1500	3	438	434	438	438	770	436
1500	5	743	755	762	762	1091	734
1500	7	1038	1061	1054	1054	1300	1083
1500	9	1337	1377	1337	1337	1436	1347
2000	3	600	603	625	625	1046	668
2000	5	1005	1001	982	982	1453	1006
2000	7	1417	1393	1387	1387	1742	1432
2000	9	1789	1782	1785	1785	1924	1793

Table 2. Comparison of hit ratio for several algorithms

String length	Frame size	FIFO	LFU	LRU	MRU	OPR	ELRU
1000	3	31.10	31.20	28.70	28.70	52.30	31.40
1000	5	46.00	48.90	49.60	49.30	71.80	50.00
1000	7	69.00	70.10	71.00	70.40	85.90	71.20
1000	9	86.60	90.00	89.30	89.30	95.80	90.40
1500	3	29.20	28.93	29.20	29.20	51.33	29.07
1500	5	49.53	50.33	50.80	50.80	72.73	48.93
1500	7	69.20	70.73	70.27	70.27	86.67	72.20
1500	9	89.13	91.80	89.13	89.13	95.73	89.80
2000	3	30.00	30.15	31.25	31.25	52.30	33.40
2000	5	50.25	50.05	49.10	49.10	72.65	50.30
2000	7	70.85	69.65	69.35	69.35	87.10	71.60
2000	9	89.45	89.10	89.25	89.25	96.20	89.65

Table 3. Comparison of total faults for several algorithms

String length	Frame size	FIFO	LFU	LRU	MRU	OPR	ELRU
1000	3	689	688	713	713	477	686
1000	5	540	511	504	507	282	500
1000	7	310	299	290	296	141	288
1000	9	134	100	107	107	42	96
1500	3	1062	1066	1062	1062	730	1064
1500	5	757	745	738	738	409	766
1500	7	462	439	446	446	200	417
1500	9	163	123	163	163	64	153
2000	3	1400	1397	1375	1375	954	1332
2000	5	995	999	1018	1018	547	994
2000	7	583	607	613	613	258	568
2000	9	211	218	215	215	76	207

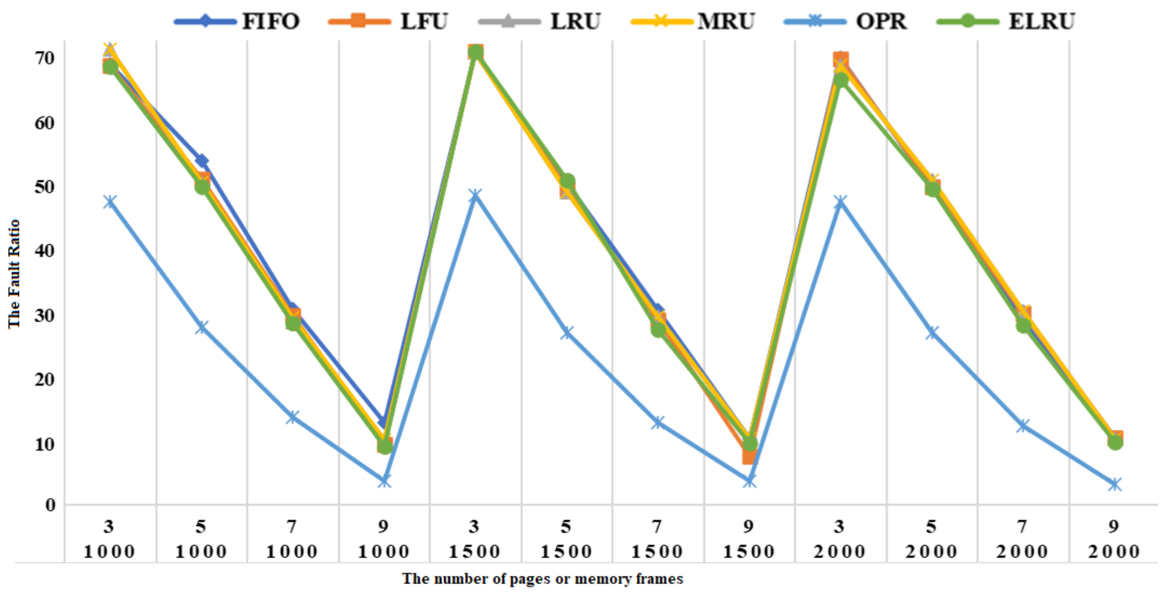
Table 4. Comparison of faults ratio for several algorithms

String length	Frame size	FIFO	LFU	LRU	MRU	OPR	ELRU
1000	3	68.90	68.80	71.30	71.30	47.70	68.60
1000	5	54.00	51.10	50.40	50.70	28.20	50.00
1000	7	31.00	29.90	29.00	29.60	14.10	28.80
1000	9	13.40	10.00	10.70	10.70	4.20	9.60
1500	3	70.80	71.07	70.80	70.80	48.67	70.93
1500	5	50.47	49.67	49.20	49.20	27.27	51.07
1500	7	30.80	29.27	29.73	29.73	13.33	27.80
1500	9	10.87	8.20	10.87	10.87	4.27	10.20
2000	3	70.00	69.85	68.75	68.75	47.70	66.60
2000	5	49.75	49.95	50.90	50.90	27.35	49.70
2000	7	29.15	30.35	30.65	30.65	12.90	28.40
2000	9	10.55	10.90	10.75	10.75	3.80	10.35

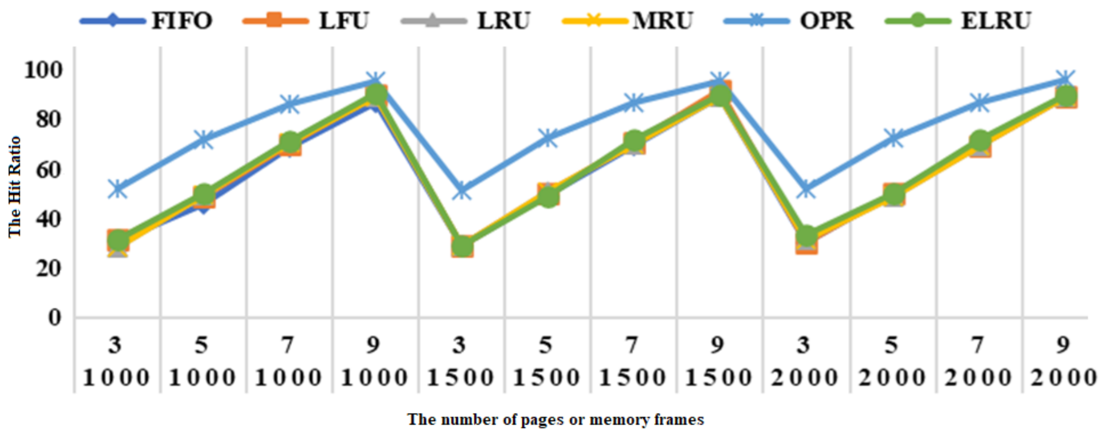
Overall, according to the obtained results in the tables, FIFO tends to perform worse than most other algorithms, especially for larger frame sizes. Its faults decrease as frame size increases, but it still trails behind more sophisticated algorithms like OPR and E-LRU. It is also proved that the suggested algorithm E-LRU is highly efficient, surpassing both the FIFO and LRU algorithms in terms of performance. The algorithm's influence increases proportionally with the number of frames. In addition, the E-LRU algorithm yields the lowest page fault rate compared to the FIFO and LRU algorithms, resulting in a higher hit ratio percentage and total number of hits.

For further evaluation, Figure 7 shows the computed fault ratio for each replacement strategy in Figure 7(a), and the computed hit ratio for each replacement strategy in Figure 7(b). As shown, the proposed E-LRU page replacement algorithm is generally efficient as it minimizes page faults and raises the hit percentage. This results in enhanced system efficiency and performance. Furthermore, the E-LRU algorithm is economically efficient as it builds upon the classic LRU algorithm, which is widely used in page replacement algorithms and has the lowest associated costs. Furthermore, identifying flaws in this algorithm is quite effortless. Previous examples demonstrate that the number of page faults experienced in an n-page memory system utilizing an (E-LRU) algorithm is lower than in a traditional LRU strategy. This indicates that our proposed approach outperforms the LRU algorithm.

Although OPR outperformed other practical algorithms, including E-LRU, however, its implementation in practice is hindered by the fact that the operating system lacks knowledge of future requests. In addition, identifying faults in OPR requires effort, making error handling particularly challenging. Occasionally, the recently accessed page gets substituted, consuming a significant amount of time. Although the proposed E-LRU algorithm may not yield superior outcomes to the OPR algorithm, it is advantageous due to its practical implementation ability. The designed E-LRU successes in producing results nearly identical to OPR, as well as improving system performance by reducing the page fault rate. Furthermore, it is easy to implement without extra cost, unlike OPR. In addition, the error detection in the proposed algorithm is relatively straightforward, and its cost is comparably lower than that of the OPR algorithm. It also yields superior outcomes compared to the currently available page replacement algorithms. Occasionally, mainly when augmenting the frame count, it attains comparable outcomes to OPR, making it an excellent alternative to the OPR algorithm.



(a)



(b)

Figure 7. The computed (a) faults ratio and (b) hit ratio for each replacement strategy

5. CONCLUSION

Several page replacement strategies and algorithms have garnered significant attention from researchers globally to enhance system performance and reduces the occurrence of page faults. In this paper, a new efficient page replacement algorithm, i.e., enhanced LRU, is proposed based on two new features used to select the victim pages, which are the total number of references and the modified bit. The proposed algorithm is implemented and compared with other page replacement algorithms, including FIFO, LRU, and OPR to assess their efficiency and performance. According to the experimental results, our proposed E-LRU demonstrated its efficiency over other replacement algorithms. As a future work, the suggested algorithm can be improved by taking into account the extra parameters and criteria that characterize the features of the pages in the cache, such as the size and cost of each object in the cache.

REFERENCES

[1] J. Kim, W. Choe, and J. Ahn, “Exploring the design space of page management for (Multi-Tiered) memory systems,” in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 715–728.

[2] G. Lia and S. Roman, "Information system for teaching virtual memory management," in *Proceedings of the VII International Scientific and Practical Conference*, 2022, p. 531, doi: 10.46299/ISG.2022.1.17.

[3] W. W. Chu and H. Opderbeck, "The page fault frequency replacement algorithm'," in *AFIPS Conference Proceedings - 1972 Fall Joint Computer Conference, AFIPS 1972*, 1972, vol. 41, pp. 597–609, doi: 10.1145/1479992.1480077.

[4] D. Meint and S. Liebald, "From FIFO to predictive cache replacement," *Network*, vol. 25, 2019.

[5] J. Oh and M. K. Sang-su Kim, "On the study of block replacement policy using LFR," in *Proceedings of the Korean Institute of Electrical Engineers Conference*, 1998, pp. 499–502.

[6] C.kavar and S. Parmar, "Performance analysis of LRU page replacement algorithm with reference to different data structure," *International Journal of Engineering Research and Application*, vol. 3, no. 1, pp. 2070–2076, 2013.

[7] Y. Smaragdakis, S. Kaplan, and P. Wilson, "EELRU: simple and effective adaptive page replacement," *ACM Sigmetrics Performance Evaluation Review*, vol. 27, no. 1, pp. 122–133, 1999, doi: 10.1145/301464.301486.

[8] M. M. Kumar and B. R. Rajendra, "An input enhancement technique to maximize the performance of page replacement algorithms," *International Journal of Research in Engineering and Technology*, vol. 04, no. 06, pp. 302–307, 2015, doi: 10.15623/ijret.2015.0406051.

[9] A. V. Aho, P. J. Denning, and J. D. Ullman, "Principles of optimal page replacement," *Journal of the ACM (JACM)*, vol. 18, no. 1, pp. 80–93, 1971, doi: 10.1145/321623.321632.

[10] A. Saxena, "A study of page replacement algorithms," *International Journal of Engineering Research and General Science*, vol. 2, no. 4, pp. 385–388, 2014.

[11] A. S. Chavan, K. R. Nayak, K. D. Vora, M. D. Purohit, and P. M. Chawan, "A comparison of page replacement algorithms," *International Journal of Engineering and Technology*, vol. 3, no. 2, p. 171, 2011.

[12] G. P. Arya, D. Prasad, and S. S. Rana, "An improved page replacement algorithm using block retrieval of pages," *International Journal of Engineering and Technology(UAE)*, vol. 7, no. 4, pp. 32–35, 2018, doi: 10.14419/ijet.v7i4.5.20004.

[13] H. Nomura, "Experimental investigation of lazy evaluation method in replacement algorithm for long-term re-reference cache management," *Bulletin of Networking, Computing, Systems, and Software*, vol. 9, no. 1, pp. 83–90, 2020.

[14] D. Akbari-Bengar, A. Ebrahimnejad, H. Motameni, and M. Golsorkhtabaramiri, "Improving of cache memory performance based on a fuzzy clustering based page replacement algorithm by using four features," *Journal of Intelligent and Fuzzy Systems*, vol. 39, no. 5, pp. 7899–7908, 2020, doi: 10.3233/JIFS-201360.

[15] S. Muthusundari, M. A. Berlin, J. G. Priya, and K. Balasaranya, "A buffer based page replacement algorithm to reduce page fault," *Materials Today: Proceedings*, vol. 33, pp. 4557–4560, 2020, doi: 10.1016/j.matpr.2020.08.182.

[16] S. Das, N. R. Das, S. K. Basu, H. Mondal, and A. Bose, "Page replacement technique on the basis of frequency of occurrence of pages," in *Proceedings of International Conference on Frontiers in Computing and Systems: COMSYS*, 2021, pp. 823–831, doi: 10.1007/978-981-15-7834-2_77.

[17] P. Banerjee, V. Raj, K. Thakur, B. Kumar, and M. K. Dehury, "A new proposed hybrid page replacement algorithm (HPRA) in real time systems.," in *Proceedings - 5th International Conference on Smart Systems and Inventive Technology, ICSSIT*, 2023, pp. 1620–1625, doi: 10.1109/ICSSIT55814.2023.10060934.

[18] Y. Lee, H. Y. Yeom, and H. Han, "APR: adaptive page replacement scheme for scientific applications," *Cluster Computing*, vol. 26, no. 5, pp. 2551–2562, 2023, doi: 10.1007/s10586-021-03296-2.

[19] D. A. Bengar, A. Ebrahimnejad, H. Motameni, and M. Golsorkhtabaramiri, "A page replacement algorithm based on a fuzzy approach to improve cache memory performance," *Soft Computing*, vol. 24, no. 2, pp. 955–963, 2020, doi: 10.1007/s00500-019-04624-w.

[20] G. Rexha, E. Elmazi, and I. Tafa, "A comparison of three page replacement algorithms: FIFO, LRU and optimal," *Academic Journal of Interdisciplinary Studies*, vol. 4, no. 2, 2015, doi: 10.5901/ajis.2015.v4n2s2p56.

[21] S. H. Abbas, W. A. K. Naser, and L. M. Kadhim, "Study and comparison of replacement algorithms," *International Journal of Engineering Research and Advanced Technology*, vol. 08, no. 08, pp. 01–06, 2022, doi: 10.31695/ijerat.2022.8.8.1.

[22] P. G. Naik, G. R. Naik, and M. B. Patil, *Conceptualizing Python in Google Colab*. Shashwat Publication, 2022.

[23] Aman, "Optimal page replacement algorithm." *Scaler*, 2023, Accessed: Mar. 1, 2024. [Online]. Available: https://www.scaler.com/topics/optimal-page-replacement-algorithm.

[24] "Page replacement algorithms in operating systems," *Geeksforgeeks.org*, 2024. Accessed: Mar. 1, 2024. [Online]. Available: https://www.geeksforgeeks.org/page-replacement-algorithms-in-operating-systems/.

[25] K. R. Baskaran and C. Kalaiarasan, "Improving hit ratio and byte hit ratio using combined pre-fetching and web caching," *International Review on Computers and Software*, vol. 9, no. 8, pp. 1426–1433, 2014, doi: 10.15866/irecos.v9i8.2594.

BIOGRAPHIES OF AUTHORS

Afaf Tareef received a B.Sc. degree in computer science from Mutah University, Jordan in 2008, an M.Phil. degree from the University of Jordan in 2010, and a Ph.D. degree from the University of Sydney, Australia in 2017. She is currently an associate professor in the Faculty of Information Technology at Mutah University, Jordan. She has many publications in several international conferences and journals. Her research interests include image processing and medical image analysis. She can be contacted at email: a.tareef@mutah.edu.jo.

Khawla Al-Tarawneh received a B.Sc. degree in computer science from Mutah University, Jordan in 2015, a master degree from the Mutah University in 2018. and student of Ph.D. in University of Jordan from 2022 to present, lab supervisor and teacher assistant in Mutah University, Jordan from 2015 to present. She can be contacted at email: Kol9220471@ju.edu.jo.

Omar Alhuniti is a Ph.D. candidate in computer science at the University of Jordan, with a strong academic and professional background. He holds an M.Sc. in Computer Science from Princess Sumaya University for Technology (PSUT), and a B.Sc. from AL-Balqa'a Applied University. He currently serves as IT Manager at the Ministry of Tourism and Antiquities in Jordan, where he leverages his skills in programming, systems development, and IT management. He has previously held roles as a teacher assistant at PSUT and as a programmer/systems developer at both the Ministry of Tourism and Antiquities and Next Generation Technologies Co. Ltd. Now pursuing his Ph.D., he is focused on advancing his knowledge in artificial intelligence. He can be contacted at email: AMR9220474@ju.edu.jo.