Some mathematical facts about optimal cache replacement

Pierre Michaud

Some mathematical facts about optimal cache replacement

Pierre Michaud

Inria
pierre.michaud@inria.fr

December 7, 2016

Abstract

This paper exposes and proves some mathematical facts about optimal cache replacement that were previously unknown or not proved rigorously. An explicit formula is obtained, giving OPT hits and misses as a function of past references. Several mathematical facts are derived from this formula, including a proof that OPT miss curves are always convex, and a new algorithm called *OPT tokens*, for reasoning about optimal replacement.

1 Introduction

Accessing a small and nearby memory is generally faster than accessing a large or distant one. Caching is the idea, ubiquitous in computer architecture, to keep some data in a small and nearby memory called a *cache*. The total data set is often larger than the cache. When the requested data is not in the cache, this is a *miss*. Otherwise, this is a *hit*. Upon a miss, a decision must be made: should the accessed data be kept in the cache, and if so, which data should be evicted from the cache to make room for the missing data ? This decision is taken by an algorithm called a *replacement policy*. Not all replacement policies generate the same number of misses. The performance and energy cost of a miss is greater than that of a hit, and we would like to have as few misses as possible.

Fifty years ago, Belady introduced a replacement policy minimizing the number of misses [3]. However, this optimal policy requires to know future memory accesses, while replacement policies that can be implemented in practice generally base their decisions on past accesses. Still, optimal replacement is a valuable conceptual tool for evaluating and analyzing practical replacement policies. Even though practical replacement policies cannot be optimal in general, they can be optimal or close to optimal on certain classes of program behavior. Moreover, optimal replacement was a direct inspiration for some cache replacement policies, in principle implementable, that were proposed recently [16, 9].

Though introduced five decades ago, optimal replacement is unintuitive and not completely understood. The main goal of this paper is to expose new, unknown mathematical facts about optimal replacement and to prove them rigorously. In pursuing this goal, we also clarify some facts about optimal replacement that we believe are not well known. We consider optimal replacement both with and without cache bypassing.

Although the term *cache* is used throughout this paper, optimal replacement is of interest for any cache-like structure, such as Translation Lookaside Buffer (TLB), Branch Target Buffer (BTB), main memory

(paging), etc. We call *items* the objects stored in the cache. E.g., an item may be an aligned memory block (hardware cache), a virtual page address (TLB), a branch address (BTB), etc. In this study, we consider fully-associative caches. Nevertheless, most of our conclusions apply to set-associative caches, as every cache set behaves like a small independent fully-associative cache.

2 Structure of the paper

The goal of Sections 4 and 6 is to obtain equations (14) and (18). All subsequent mathematical facts are derived from these equations. We provide some corroboration for these equations in Section 5 by proving the (already known) fact that all LRU hits are OPT hits, and in Section 7 by computing the OPT miss probability for random traces and comparing with Monte Carlo simulations. In Section 8, we introduce the *OPT matrix*, an algorithm for reasoning about all cache sizes simultaneously. Using the OPT matrix, we show that the output trace of OPT distances has a special structure, described by theorem 1. Using theorem 1, we prove in Section 9 that the OPT miss curve is always a convex function of the cache size, a fact discovered recently but not proved rigorously [2]. In Section 10, we focus on a single cache size and derive from the OPT matrix an algorithm called the *OPT tokens* for obtaining OPT hits and misses. In Section 11, we derive directly from the OPT tokens an equivalence between a cache of size $j + 1$ without bypassing and a cache of size j with bypassing. Finally, in Section 12, and using the OPT tokens, we explain why accessing a data set in a circular fashion maximizes the number of OPT misses. We cite related work where we think appropriate. Results for which we do not mention explicitly any related work are, to the best of our knowledge, new results.

This paper contains only mathematical facts, independent of empirical properties of applications. We had to introduce a substantial amount of notations permitting to reason with precision. Our most important notations are listed in Table 2.

3 Belady's MIN and Mattson's OPT

Belady's 1966 paper [3] and Mattson et al.'s 1970 paper [13] are important and widely cited. Yet, there seems to be a long-lived confusion between Belady's MIN and Mattson's OPT. Many authors who referred to Belady's MIN actually described Mattson's OPT.[1]

Upon a cache miss, Mattson's OPT replacement policy evicts from the cache the item that will be re-referenced the furthest in time. This is stated very clearly by Mattson et al., who provide a proof that OPT is indeed optimal, in the sense that it minimizes the number of misses. Mattson et al. showed that OPT (like LRU) is a *stack* replacement policy, i.e., the content of a cache of size j is always included in a cache of size $j + 1$. They introduced the *OPT stack*, based on the OPT replacement policy, for computing the minimum number of cache misses for all cache sizes at once.

Belady's MIN is not identical to Mattson's OPT. The MIN algorithm and the MIN replacement policy should be distinguished. The MIN algorithm is for determining hits and misses under optimal replacement [3, 4].

Table 1 shows how the MIN algorithm works on an example, assuming a 3-entry cache. MIN can be explained with a matrix where rows are associated with distinct items and columns represent successive

[1] The origin of this confusion is not clear. Confusion was already there in the early years [1, 7].

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
trace	A	B	C	D	E	E	C	D	F	A	B	D	B	A	D	E	F	B
MIN matrix																		
A		r									r	r	r	r	r			
B			r									r	r	r	r	r	r	r
C				r	r	r	r	r										
D					r	r	r	r	r	r	r	r	r	r	r	r		
E						r	r										r	
F										r								r
hits						h	h	h				h	h	h	h			h
evictions																		
MIN				A	B				E	C	F					A	D	
OPT				B	A				C	F	E					A	D	

Table 1: Belady's MIN algorithm on an example, assuming a 3-entry cache. Each row of the MIN matrix is associated with a distinct item. Columns represent successive times. Yellow cells show when items are referenced. A cell is marked "r" when a cache entry is reserved at that time for that item.

times.[2] When the item referenced at time t and previously referenced at time t' is determined to be in the cache at t (hit), a cache entry is reserved for the item for the time interval $]t', t+1]$, under the constraint that the number of reserved entries at any time cannot exceed the cache capacity. If cache entries cannot be reserved in $]t', t+1]$ (because of the capacity constraint), this is a miss, and an entry is reserved only for the time $t+1$.

The MIN algorithm determines evicted items after a delay. More specifically, when a column of the MIN matrix contains a number of reservations equal to the cache capacity, we know which items are not in the cache and must have been evicted. For instance in Table 1, at time $t = 8$, the MIN algorithm determines that the cache contained items C,D and E at time $t = 7$, hence that items A and B had been evicted earlier. The MIN replacement policy attributes evictions to misses in the following way: among a group of items to evict, the least-recently referenced item is evicted the earliest [3]. As can be seen in Table 1, MIN and OPT, although both optimal, are different replacement policies. Note that, at $t = 8$, when MIN determines the eviction of A and B, it is not known yet which of A or B will be re-referenced first.

In 1966, Belady did not prove that MIN is optimal. The optimality of MIN was proved in 1974 when Belady and Palermo, relying on Mattson's proof of optimality for OPT, showed that MIN and OPT yield the same hits and misses [4].

Remarkably, the MIN algorithm is an *online* algorithm: it provides hit/miss information without delay, determining at time t, without look-ahead, if the item referenced at time t is or is not in the cache. That is, even though optimal replacement depends on future references, whether the current reference is a hit or a miss depends only on past references. This property was not mentioned in the 1966 paper. It was emphasized later in the 1974 paper [4]. This property of MIN has been exploited in a new cache replacement policy published recently [9].

[2] Belady's 1966 algorithm uses a vector, not a matrix. However the matrix captures the essence of MIN and is easier to explain [4, 20, 9].

notation	definition	description	remark
β	bypassing is allowed ($\beta = 1$) or not ($\beta = 0$)		$\beta \in \{0, 1\}$
$R_{[t,t'[}$	set of items referenced in the time interval $[t, t'[$		$R_{[t,t[} = \varnothing$
X_t	item referenced at time t		$\{X_t\} = R_{[t,t+1[}$
s_t	$s_t := \left\lvert R_{[0,t[} \right\rvert$	number of distinct items referenced before time t	
C_t^j	set of items in an OPT cache of size j at time t		$C_t^0 = \varnothing$, $C_t^{s_t} = R_{[0,t[}$
p_t	OPT stack position of item X_t		$\{X_t\} = C_t^{p_t} - C_t^{p_t-1}$
$M^j_{[t',t[}$	$M^j_{[t',t[} := \left\lvert \{\tau \in [t', t[: p_\tau > j\} \right\rvert$	number of misses for the cache of size j in the time interval $[t', t[$	$M^{s_t}_{[t',t[} = 0$
$u_t(i)$	$u_t(i) := \max\{\tau \leq t : \left\lvert R_{[\tau,t[} \right\rvert = i\}$	greatest time τ such that i distinct items are referenced in $[\tau, t[$	$u_t(1) = t - 1$ $u_t(i+1) < u_t(i)$
q_t	$q_t := \min\{i \geq 1 : X_{u_t(i)} = X_t\}$	LRU stack position of item X_t	$X_t \notin R_{[u_t(q_t)+1,t[}$
$D_t^j(i)$	$D_t^j(i) := i - M^j_{[u_t(i)-\beta+1,t[}$		$D_t^{s_t}(i) = i$
I_t^j	$I_t^j := \min\{i \in [1, s_t] : D_t^j(i) = j\}$		$I_t^{s_t} = s_t$

Table 2: Main notations.

4 OPT hits and misses depend only on past references

Belady's MIN algorithm demonstrates that, under optimal replacement, hits and misses depend only on past references. Hence, there must exist an explicit formula giving hits and misses as a function of past references. The goal of this section is to find this formula (equation (8)). Our motivation is that such formula will be useful for understanding OPT hits and misses.

Our starting point is the OPT replacement policy, for which multiple proofs of optimality are known [13, 7, 8, 21, 22, 11].

Figure 1 (left part) illustrates Mattson's OPT-stack update procedure [13]. In this example, the item A referenced at time t in the reference stream is found at position 5 in the stack, meaning that 5 is the smallest cache size giving a cache hit at time t. Figure 2 gives a similar example where bypassing is allowed. In the paper we consider both the classical case (no bypassing) and the case with bypassing. We use parameter β to indicate if bypassing is allowed ($\beta = 1$) or not ($\beta = 0$).

Table 2 lists our main notations. We use the usual notations of set theory and the usual notations for intervals (e.g., $[t, t'[$ is the set of values τ such that $t \leq \tau < t'$). For $t < t'$, $R_{[t,t'[}$ represents the set of items referenced at least once between times t (included) and t' (not included). The stack size s_t is the number of distinct items referenced at least once before time t, i.e., $s_t = |R_{[0,t[}|$. The OPT stack position of item X_t referenced at time t (aka *OPT distance*) is denoted p_t. If item X_t is referenced for the first time, we define $p_t = s_t + 1 = s_{t+1}$. C_t^j is the set of items in a cache of size j (items) at time t under optimal replacement ($C_t^0 = \varnothing$). I.e., C_t^j is the set of items whose position in the OPT stack at t does not exceed position j.

Let us focus on one item, X, referenced at time t_2, and let us assume that X was previously referenced at time $t_1 < t_2 - 1$. That is, $X = X_{t_1} = X_{t_2}$ and $X \notin R_{[t_1+1,t_2[}$.

Let $\pi_t \in [1, s_t]$ denote the position of X in the OPT stack at time $t \in]t_1, t_2]$. First, it should be observed

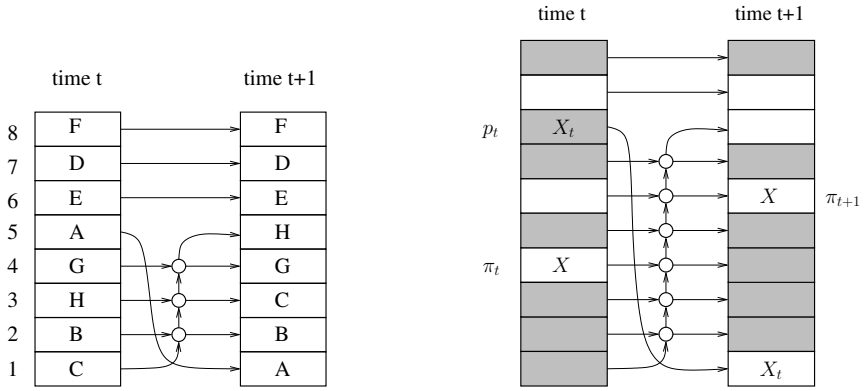

Figure 1: Mattson's OPT stack on an example (no bypassing). **Left:** The item $X_t = A$ referenced at time t is found at position $p_t = 5$ in the stack. Circles represent sorting networks: the item whose time of next reference is furthest is sent to higher stack positions. The alphabetical order indicates how items are ordered at time t by their time of next reference. **Right:** Another example, focusing on a particular item X. Darkened stack entries hold items that will be referenced before X. Item X goes up in the stack when X_t is above X and all the stack positions below X are occupied by items referenced before X.

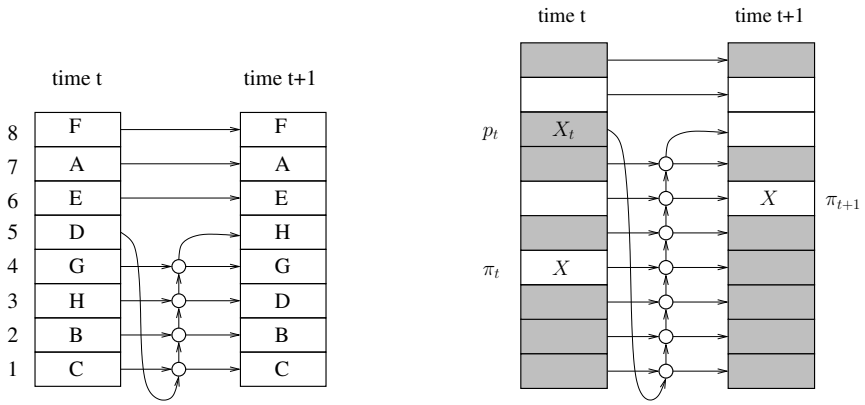

Figure 2: OPT stack with bypassing. **Left:** The alphabetical order indicates how items are ordered at time $t + 1$ by their time of next reference. **Right:** Focus on a particular item X. Item X goes up in the stack when X_t is above X, X_t is re-referenced before X, and all the stack positions below X are occupied by items that will be referenced before X.

that the stack position of an item cannot decrease unless the item is referenced. As X is not referenced during $]t_1, t_2[$, π_t is non-decreasing in $]t_1, t_2]$. A necessary condition for X to go up in the stack (i.e., $\pi_{t+1} > \pi_t$) is that $p_t > \pi_t$ and all the stack positions below π_t are occupied by items that are referenced in $[t+1, t_2[$. A useful quantity to consider is the number N_t^j of items in the cache of size j, at time t, that are referenced in $[t, t_2[$:

$$\text{for } t \in]t_1, t_2] \qquad N_t^j := \left|C_t^j \cap R_{[t,t_2[}\right| \tag{1}$$

Another useful quantity is

$$e_t := \left|R_{[t,t_2[}\right| - \left|R_{[t+1,t_2[}\right|$$

Notice that $e_t \in \{0, 1\}$ and

$$N_{t+1}^j + (1-\beta)e_t \leq j$$

When X goes up in the stack, the new position of X is h_{t+1} defined as

$$h_{t+1} := \min\{j \in [1, s_{t+1}] : N_{t+1}^j < j - (1-\beta)e_t\}$$

For $t \in]t_1, t_2[$, we have $\pi_{t+1} = \max(\pi_t, h_{t+1})$. Note that $\pi_{t_1+1} = 1$ for $\beta = 0$ and $\pi_{t_1+1} = h_{t_1+1}$ for $\beta = 1$. Therefore

$$p_{t_2} = \pi_{t_2} = \max\{h_t : t \in]t_1 + 1 - \beta, t_2]\} \tag{2}$$

The OPT stack update procedure leads to the following recurrence relation:

$$\begin{aligned} j \geq p_t \qquad & N_{t+1}^j = N_t^j - e_t \\ j < p_t \qquad & N_{t+1}^j = \min(j, N_t^j + 1 - \beta e_t) - (1-\beta)e_t \end{aligned} \tag{3}$$

The values $N_{t_1+1}^j$ are not known. However, we know that $N_{t_2}^j = 0$. If recurrence (3) could be reversed, we could obtain N_t^j. Actually, recurrence (3) can be reversed partially. Notice that $j < p_t$ means a miss in the cache of size j at time t. Let us define

$$M_{[t',t[}^j := \left|\{\tau \in [t', t[: p_\tau > j\}\right|$$

$M_{[t',t[}^j$ is the number of misses[3] occurring in $[t', t[$ for an OPT cache of size j. The partially reversed recurrence is:

$$N_{t+1}^j < j - (1-\beta)e_t \qquad \Longrightarrow \qquad N_t^j - N_{t+1}^j = e_t - M_{[t,t+1[}^j \tag{4}$$

Let us denote $rev(t)$ the condition on the left-hand side of (4):

$$rev(t) \quad : \qquad N_{t+1}^j < j - (1-\beta)e_t$$

We have

$$rev(\tau)\ \forall \tau \in [t, t_2[\quad \Longrightarrow \quad N_t^j - N_{t_2}^j \quad = \sum_{\tau \in [t,t_2[} \left(e_\tau - M_{[\tau,\tau+1[}^j\right)$$

Hence

$$rev(\tau)\ \forall \tau \in [t, t_2[\qquad \Longrightarrow \qquad N_t^j = \left|R_{[t,t_2[}\right| - M_{[t,t_2[}^j$$

[3] We recall that when item X_τ is referenced for the first time at $\tau \in [t', t[$, we define $p_\tau = s_\tau + 1$. With our definition of $M_{[t',t[}^j$, the reference at τ is counted as a hit for all $j \geq s_\tau + 1$.

and

$$rev(\tau)\ \forall \tau \in [t+1, t_2[\quad \Longrightarrow \quad N^j_{t+1} + (1-\beta)e_t = \left|R_{[t+\beta,t_2[}\right| - M^j_{[t+1,t_2[} \tag{5}$$

As the equality on the right-hand side of (5) is always true for $t = t_2 - 1$, we can apply a backward induction on condition $rev(\tau)$:

$$rev(\tau)\ \forall \tau \in [t, t_2[\iff \forall \tau \in [t, t_2[\quad \left|R_{[\tau+\beta,t_2[}\right| - M^j_{[\tau+1,t_2[} < j \tag{6}$$

Let us define for $\beta = 0, j > 1$ and for $\beta = 1, j > 0$

$$T^j_{t_2} := \min\Big\{t < t_2 : \forall \tau \in [t, t_2[,\ \left|R_{[\tau+\beta,t_2[}\right| - M^j_{[\tau+1,t_2[} < j\Big\} \tag{7}$$

and for $\beta = 0$, $T^1_{t_2} := t_2$.

Consider the case $T^j_{t_2} > t_1 + 1 - \beta$. Then for $t = T^j_{t_2} - 1 > t_1 - \beta$, by definition (7) and using equivalence (6), condition $rev(t)$ must be false. Hence $h_{t+1} > j$, which implies (from (2)) $p_{t_2} > j$. On the other hand, when $T^j_{t_2} \le t_1 + 1 - \beta$, condition $rev(t)$ is true for all $t \in [t_1 + 1 - \beta, t_2[$, hence $h_{t+1} \le j$ for all $t \in [t_1 + 1 - \beta, t_2[$, and $p_{t_2} \le j$. Therefore

$$p_{t_2} = \min\{j \ge 1 : T^j_{t_2} \le t_1 + 1 - \beta\} \tag{8}$$

So far, we have assumed $t_2 > t_1 + 1$, but equation (8) is valid also for $t_2 = t_1 + 1$ ($p_{t_2} = 1$). It should be noted that $T^j_{t_2}$ does not depend on which item is referenced at t_2. Only the t_1 in equation (8) depends on the item referenced at t_2.

We end this section with two facts that are used later in the paper. First,

$$\forall j \quad T^{j+1}_t \le T^j_t \tag{9}$$

This can be seen as follows. Instead of referencing X_t at $t = t_2$, let us reference the item that was last referenced at $t_1 = T^j_t - 1 + \beta$. That item exists because, by definition of T^j_t, $\left|R_{[\tau+\beta,t[}\right|$ must be incremented when going from $\tau = T^j_t$ to $\tau = T^j_t - 1$. As $T^j_t = t_1 + 1 - \beta$, we must have $p_t \le j$, hence $p_t \le j + 1$, which implies $T^{j+1}_t \le t_1 + 1 - \beta$, that is, $T^{j+1}_t \le T^j_t$.

The second fact is

$$\tau \in [T^j_t, t[\quad \Longrightarrow \quad M^j_{[\tau,t[} - M^{j+1}_{[\tau,t[} \in \{0, 1\} \tag{10}$$

To see this, let us reference at $t = t_2$ the item that was last referenced at $t_1 = T^{s_t}_t - 1 + \beta$, and let us consider $j < s_t$. We must have $T^{j+1}_t \ge T^{s_t}_t = t_1 + 1 - \beta$. Hence for $\tau \in [T^j_t, t]$, quantities N^j_τ and N^{j+1}_τ are defined. From (7), (6) and (5), we have

$$\begin{aligned}
\tau \in [T^j_t - 1, t[\quad &\Longrightarrow \quad rev(t') \quad \forall t' \in [\tau + 1, t[\\
&\Longrightarrow \quad N^j_{\tau+1} + (1-\beta)e_\tau = \left|R_{[\tau+\beta,t[}\right| - M^j_{[\tau+1,t[} \\
&\Longrightarrow \quad M^j_{[\tau+1,t[} - M^{j+1}_{[\tau+1,t[} = N^{j+1}_{\tau+1} - N^j_{\tau+1}
\end{aligned}$$

which, from definition (1), proves (10).

5 All LRU hits are OPT hits

The fact that OPT is optimal means that no replacement policy can generate fewer misses than OPT. However, an individual reference that is a hit under a given policy P is not necessarily a hit under OPT. Still, it is known (though perhaps not widely) that every LRU hit is also an OPT hit [15, 12, 5]. It can be verified that equation (8) is consistent with this fact. Let us assume $p_{t_2} > 1$. From (8), we must have $T_{t_2}^{p_{t_2}-1} > t_1+1-\beta$. Letting $t = T_{t_2}^{p_{t_2}-1} - 1$, and from (7),

$$\begin{aligned} p_{t_2} - 1 &= \left|R_{[t+\beta,t_2[}\right| - M_{[t+1,t_2[}^{p_{t_2}-1} \\ &\leq \left|R_{[t+\beta,t_2[}\right| \\ &< \left|R_{[t_1,t_2[}\right| \end{aligned}$$

Let us define

$$q_{t_2} := \left|R_{[t_1,t_2[}\right|$$

Quantity q_{t_2} is the LRU stack position (aka reuse distance) of the item X referenced at time t_2. We have

$$p_{t_2} \leq q_{t_2} \tag{11}$$

Inequation (11) is valid in the case $p_{t_2} = 1$, and also when item X_{t_2} is referenced for the first time ($p_{t_2} = q_{t_2} = |R_{[0,t_2[}| + 1$). In words, inequation (11) means that every LRU hit is also an OPT hit.

There is something remarkable here. The LRU replacement policy, which evicts from the cache the item referenced the least recently, is an online policy: it makes replacement decisions without knowing future references. LRU is not optimal in general, but it works well in many practical applications. The performance of LRU lies in empirical properties of traces, not in LRU itself. Indeed, if we consider random traces (items have equal probability to be referenced), *all* online replacement policies yield the same average hit ratio, equal to the cache size divided by the number of distinct items. Without considering trace properties, LRU is neither better nor worse than any other online policy. Still, inequation (11) highlights a special relationship between LRU and OPT. This suggests a (rather theoretical) question, which we will not try to answer in this paper: is there an online policy P other than LRU such that, for any trace, all hits under policy P are hits under OPT ?

6 Putting the reuse distance into the equation

Equation (8) is written in terms of t_1 and t_2. In this section, we rewrite it in terms of t and q_t (the reuse distance). Let us define

$$I_t^j := \left|R_{[T_t^j-1+\beta,t[}\right| \tag{12}$$

Note that (9) implies

$$\forall j \quad I_t^{j+1} \geq I_t^j \tag{13}$$

Equation (8) can be written

$$p_t = \min\{j \geq 1 : I_t^j \geq q_t\} \tag{14}$$

except for the special case $p_t = s_t + 1$ for a first reference. As we defined it, I_t^j depends on T_t^j. For the sake of convenience, we would like to find a more direct expression for I_t^j. Let us introduce the quantity

$$D^j_{[\tau,t[} \;:=\; \left|R_{[\tau+\beta,t[}\right| - M^j_{[\tau+1,t[}$$

we can rewrite (7) as

$$T_t^j \;=\; \min\Big\{t' < t : \forall \tau \in [t', t[,\ D^j_{[\tau,t[} < j\Big\} \tag{15}$$

When τ decreases, $D^j_{[\tau,t[}$ increases only when $\left|R_{[\tau+\beta,t[}\right|$ is incremented. Instead of considering all values τ, we can focus on the values τ such that $\left|R_{[\tau,t[}\right|$ is incremented. For $i > 0$, let

$$u_t(i) := \max\{\tau < t : \left|R_{[\tau,t[}\right| = i\} \tag{16}$$

As $\left|R_{[T_t^j+\beta,t[}\right| < I_t^j$ (cf. the argument used in proving (9)), we must have

$$u_t(I_t^j) = T_t^j - 1 + \beta$$

Consequently,

$$\begin{array}{rcl} i < I_t^j & \Longrightarrow & u_t(i) > u_t(I_t^j) \\ & \Longrightarrow & u_t(i) - \beta \geq T_t^j \\ & \Longrightarrow & D^j_{[u_t(i)-\beta,t[} < j \end{array}$$

Moreover,

$$D^j_{[u_t(I_t^j)-\beta,t[} = D^j_{[T_t^j-1,t[} = j$$

Let us define for $i \in [1, s_t]$

$$D_t^j(i) := D^j_{[u_t(i)-\beta,t[} = i - M^j_{[u_t(i)-\beta+1,t[} \tag{17}$$

We find that I_t^j can be obtained as

$$I_t^j \;=\; \min\{i \in [1, s_t] : D_t^j(i) = j\} \tag{18}$$

Notice that $D_t^j(i) \leq i$, hence from (18), $j = D_t^j(I_t^j) \leq I_t^j$. Therefore $I_t^{q_t} \geq q_t$ and, from (14), $p_t \leq q_t$ (already shown in section 5). The rest of the paper is built on equations (14) and (18).

If we focus on a single cache size j, the procedure for determining whether the reference at time t is a hit or a miss is simply:

$$\begin{array}{rcl} \text{OPT hit at time } t & \Longleftrightarrow & p_t \leq j \\ & \Longleftrightarrow & I_t^j \geq q_t \\ & \Longleftrightarrow & \forall i < q_t,\ D_t^j(i) < j \end{array}$$

This is illustrated in Table 3 on an example, assuming bypassing ($\beta = 1$) and cache size $j = 3$. This example shows how the hit/miss outcome at time t is obtained from past hit/miss outcomes. On this example, this is a hit at time t, as all the $D_t^j(i)$ values for $i < q_t$ are strictly less than three.

time	$\cdots$	$t-9$	$t-8$	$t-7$	$t-6$	$t-5$	$t-4$	$t-3$	$t-2$	$t-1$	t
trace	$\cdots$	A	D	F	C	F	E	C	B	B	A
hit/miss	$\cdots$		hit	miss	hit	hit	miss	hit	miss	hit	hit
$u_t(i)$	$\cdots$	$u_t(q_t)$	$u_t(5)$			$u_t(4)$	$u_t(3)$	$u_t(2)$		$u_t(1)$	
$M^3_{[u_t(i),t[}$	$\cdots$		3			2	2	1		0	
$D^3_t(i)$	$\cdots$		2			2	1	1		1	

Table 3: Example: determining hit/miss at time t from past hits/misses ($\beta = 1$ and cache size $j = 3$).

7 Random traces

Equations (14) and (18) can be used to compute the OPT miss probability for random traces, i.e., traces such that the number s of distinct items is fixed and items have equal probability $1/s$ to be referenced. Smith [17] and Knuth [10] have shown that the exact miss ratio of an OPT cache of size j on random traces can be obtained by solving a Markov chain with $\binom{s}{j}$ states. According to Knuth, there does not seem to be any simple formula for the exact OPT miss ratio of random traces for any s and j in general, although Knuth found closed-form expressions for the special cases $j = 2$ and $j = s - 1$. In general, for obtaining the OPT miss ratio on random traces, one must solve a (possibly huge) Markov chain or do Monte-Carlo simulations.

Here we search an approximation for the case $s \to \infty$, $j \to \infty$ but $j/s \in [0, 1]$ fixed. Let us consider an interval $[t, t + \Delta t[$ with Δt large. For a given item x, the probability that x is not in $R_{[t,t+\Delta t[}$ is $(1 - \frac{1}{s})^{\Delta t}$. Hence

$$\begin{aligned} \left|R_{[t,t+\Delta t[}\right| &\approx s \times (1 - (1 - \tfrac{1}{s})^{\Delta t}) \\ &\approx s \times (1 - e^{-\frac{\Delta t}{s}}) \end{aligned}$$

Then, we have

$$t - u_t(i) \approx -s \times \ln(1 - \frac{i}{s})$$

Denoting m the miss probability and taking $\beta = 1$, we have

$$D^j_t(i) \approx i + s \times \ln(1 - \frac{i}{s}) \times m$$

As $D^j_t(I^j_t) = j$, we obtain the following equation:

$$j \approx I^j_t + s \times \ln(1 - \frac{I^j_t}{s}) \times m$$

But m is the probability that $p_t > j$, i.e., the probability that $q_t > I^j_t$. As q_t is uniformly distributed in $[1, s]$, we have

$$m = 1 - \frac{I^j_t}{s}$$

Substituting $\frac{I^j_t}{s}$ with $1 - m$, we obtain

$$\frac{j}{s} \approx 1 - m + m \times \ln(m) \tag{19}$$

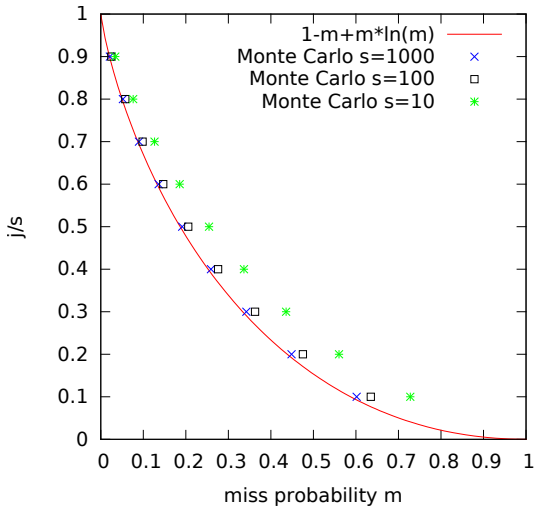

Figure 3: OPT cache of size j (bypassing allowed) and random trace with s distinct items. Relation between j/s and the miss probability m.

Figure 3 shows a plot of j/s vs. m as given by (19), compared with Monte Carlo simulations. Formula (19) models the asymptotic behavior and is more accurate as s gets larger. OPT is very effective on random traces: asymptotically, the number of distinct items must be about ten times the cache size for the OPT miss ratio to exceed sixty percent.

8 The OPT matrix

In Section 6, we considered a fixed t. Let us look at how $D_t^j(i)$ changes from t to $t+1$. First, notice that

$$i \le q_t \qquad u_{t+1}(i) = u_t(i-1)$$

$$i > q_t \qquad u_{t+1}(i) = u_t(i)$$

Then, from (17), we obtain the following update procedure for $p_t \in [1, s_t]$:

	$i > q_t$	$i \in [2, q_t]$	$i = 1$
$j \in [p_t, s_t]$	$D_{t+1}^j(i) = D_t^j(i)$	$D_{t+1}^j(i) = D_t^j(i-1) + 1$	$D_{t+1}^j(1) = 1$
$j < p_t$	$D_{t+1}^j(i) = D_t^j(i) - 1$	$D_{t+1}^j(i) = D_t^j(i-1)$	$D_{t+1}^j(1) = 1 - \beta$

(20)

In the special case $q_t = p_t = s_t + 1 = s_{t+1}$, we have $D_{t+1}^{s_t+1}(i) = i$. Procedure (20) (along with equations (14) and (18)) defines an algorithm for computing p_t. We call this algorithm the *OPT matrix*. The OPT matrix is not intended for analyzing real traces. It is inefficient in computational complexity and memory usage.[4] Nevertheless, the OPT matrix is a useful conceptual tool for reasoning about OPT, considering all

[4]Our goal here is not to obtain a computationally efficient algorithm. Sugumar and Abraham described an efficient OPT stack implementation requiring a single pass (unlike Mattson's original algorithm), but with some look-ahead [19]. They implemented this algorithm in a cache simulator called Cheetah, distributed with the Simplescalar processor simulation infrastructure [6]. We believe that Cheetah is fast enough for most practical purposes.

cache sizes simultaneously. In particular, we are going to show that, whatever the input trace, the sequence of OPT distances has a remarkable structure:

Theorem 1. *Given $t' > t$ such that $p_{t'} = p_t > 2 - \beta$ and $\forall \tau \in]t, t'[, p_\tau \neq p_t$, $\forall j \in [2 - \beta, p_t[$ there exists $\tau \in]t, t'[$ such that $p_\tau = j$.*

In words, theorem 1 states that, in the sequence of OPT distances p_t, a given distance greater than $2 - \beta$ cannot reoccur until all lower distances (except 1 in the no-bypassing case) have occurred at least once. Below is a sequence of OPT distances that we generated from a random input trace with 8 distinct items, without bypassing:

1 2 3 1 2 4 2 1 3 2 4 3 2 5 3 2 4 6 2 2 3 7 1 4 2 3 8 2 5 4 1 1 6 2 1 3 7 2 4 5 2 2 3 8 2 4 6 2 5 1 3 7 2 2 2 4
5 2 3 4 6 5 7 2 2 3 1 4 5 2 8 3 4 2 3 2 2 2 1 5 6 2 1 2 7 2 4 3 5 8 6 2 1 3 4 5 1 7 2 3 4 1 6 2 5 3 2 4 3 6 5 2
8 2 2 1 3 4 5 1 7 6 2 2 3 4 5 6 1 1 2 3 1 2 3 4 5 7 2 3 4 2 6 3 4 2 8 2 2 5 6 3 2 4 7 8 2 2 3 4 2 5 2 3 4 6 5 2
3 4 5 1 7 2 8 2 3 4 5 1 1 2 1 3 4 2 1 6 5 2 3 2 4 3 5 1 2 7 8 3 2 6 3 4 5 2 1 7 3 2 3 2 4 3 2 5 2 1 6 3 2 3 4 1
8 2 1 3 4 5

We have highlighted a few parts of the sequence to illustrate theorem 1. Here is the sequence of OPT distances produced from the same input trace when bypassing is allowed:

1 2 3 1 2 4 1 1 2 1 3 4 2 5 1 2 3 6 1 2 4 7 1 3 2 4 8 1 5 2 1 1 6 1 1 2 7 1 3 4 1 2 5 6 1 3 7 1 4 1 2 8 3 1 2 4
5 1 3 2 6 4 7 1 2 3 1 4 5 2 8 1 3 2 1 3 1 2 1 4 5 2 1 2 6 1 7 3 4 8 2 1 1 3 5 4 1 6 2 3 4 1 7 2 3 1 2 5 3 4 6 1
8 1 2 1 3 4 5 1 7 6 2 1 3 4 5 6 1 1 2 3 1 2 3 4 5 7 1 2 3 1 6 2 4 3 8 1 2 3 5 4 1 6 7 8 2 1 3 4 1 5 2 1 3 6 4 1
5 2 3 1 7 2 8 1 3 2 4 1 1 2 1 3 4 1 1 5 6 1 2 3 4 1 5 1 2 7 8 3 1 6 2 4 5 3 1 7 2 1 3 2 4 1 3 5 1 1 6 2 1 3 4 1
8 2 1 3 4 2

Notice that, without bypassing, distance 2 can occur several times consecutively, while this does not occur with bypassing.

Before proving theorem 1, let us make some preliminary observations. From (17) and (10) we must have

$$\forall i \in [1, I_t^j], \quad D_t^{j+1}(i) - D_t^j(i) \in \{0, 1\}$$

and for any $k \geq 0$

$$\forall i \in [1, I_t^j], \quad D_t^{j+k}(i) \leq D_t^j(i) + k$$

Also, observe (from (17)) that

$$D_t^j(i) \neq D_t^{j-1}(i) \quad \Longrightarrow \quad D_t^j(i+1) \neq D_t^{j-1}(i+1)$$

Let us define

$$Y_t^j := \min\{i \geq 1 : D_t^j(i) \neq D_t^{j-1}(i)\}$$

As $D_t^{j-1}(I_t^{j-1}) = j - 1$,

$$I_t^j > I_t^{j-1} \quad \Longrightarrow \quad D_t^j(I_t^{j-1}) < j \quad \Longrightarrow \quad D_t^j(I_t^{j-1}) = j - 1 \quad \Longrightarrow \quad Y_t^j > I_t^{j-1}$$

On the one hand, from (13), (14), (18), and (20), we have

$$\forall j \quad I_t^j < I_t^{p_t} \quad \Longrightarrow \quad I_{t+1}^j = I_t^j + 1$$

On the other hand, consider the smallest j such that $I_t^j > I_t^{p_t}$. As $I_t^j > I_t^{j-1} = I_t^{p_t}$, we have $Y_t^j > I_t^{j-1} = I_t^{p_t}$, hence $D_t^j(i) = D_t^{j-1}(i)$ for all $i \leq I_t^{p_t}$. Therefore

$$i < I_t^{p_t} \quad \Longrightarrow \quad D_t^{j-1}(i) < j - 1 \quad \Longrightarrow \quad D_t^j(i) < j - 1 \quad \Longrightarrow \quad D_t^{j+k}(i) < j + k - 1$$

for any $k \geq 0$. Consequently, and from (20),

$$\forall j \quad I_t^j > I_t^{p_t} \qquad \Longrightarrow \qquad I_{t+1}^j = I_t^j$$

I_t^j cannot decrease unless $I_t^j = I_t^{p_t}$. From (20), we must have

$$\forall j \quad I_t^j = I_t^{j-1} = I_t^{p_t} \quad \Longrightarrow \quad I_{t+1}^j \geq Y_t^j + 1$$

From (20), Y_t^j is updated as follows:

$$Y_{t+1}^{p_t} = 2 - \beta$$

$$j \neq p_t, \quad Y_t^j \leq q_t \qquad \Longrightarrow \qquad Y_{t+1}^j = Y_t^j + 1$$

$$j \neq p_t, \quad Y_t^j > q_t \qquad \Longrightarrow \qquad Y_{t+1}^j = Y_t^j$$

We are ready to prove theorem 1.

Proof. As $I_{t+1}^j \geq j$ and $Y_{t+1}^{p_t} = 2 - \beta$, we must have $\forall j \geq 2 - \beta$, $I_{t+1}^j \geq Y_{t+1}^{p_t}$. From (14) and the assumption $p_{t'} = p_t$, we must have $I_{t'}^{p_t} > I_{t'}^{p_t - 1}$, hence $Y_{t'}^{p_t} > I_{t'}^{p_t - 1}$ and $\forall j < p_t$, $I_{t'}^j < Y_{t'}^{p_t}$. Let us focus on one $j \in [2 - \beta, p_t[$. There must exist a $\tau \in]t, t'[$ such that $I_\tau^j \geq Y_\tau^{p_t}$ and $I_{\tau+1}^j < Y_{\tau+1}^{p_t}$, an event that we call *crossing*. This τ must be such that $I_\tau^{p_\tau} \leq I_\tau^j$ (otherwise I_τ^j would be incremented and crossing at τ would not be possible). We prove by contradiction that in fact we must have $I_\tau^j = I_\tau^{p_\tau}$. Let us assume $I_\tau^{p_\tau} < I_\tau^j$. Consequently, $I_{\tau+1}^j = I_\tau^j$ and crossing at τ implies $I_\tau^j = Y_\tau^{p_t} \leq q_\tau \leq I_\tau^{p_\tau}$, which contradicts the assumption. Therefore, $I_\tau^j = I_\tau^{p_\tau}$. If $p_\tau = j$, we are done with the proof (we found a $\tau \in]t, t'[$ such that $p_\tau = j$). Otherwise, we have $j > p_\tau$ and $I_\tau^j = I_\tau^{j-1} = I_\tau^{p_\tau}$, hence $Y_\tau^j + 1 \leq I_{\tau+1}^j < Y_{\tau+1}^{p_t} \leq Y_\tau^{p_t} + 1$ (as $p_\tau \neq p_t$). Therefore $Y_\tau^j < Y_\tau^{p_t}$, which implies (as $Y_{t+1}^j \geq Y_{t+1}^{p_t}$) that there is a $\tau' \in]t, \tau[$ such that $p_{\tau'} = j$. □

Belady and Palermo noticed that the sequence of OPT distances is constrained [4]. In particular, they understood that no two consecutive references can have the same OPT distance if that distance is greater than two (they were assuming $\beta = 0$). However, they did not mention theorem 1.

Theorem 1 provides a sanity test for OPT stack implementations or algorithms emulating an OPT stack. If, by a visual inspection of the sequence of OPT distances, we find a case where theorem 1 is violated, this is a sure indication that the algorithm or its implementation is incorrect. Note that there is no such sanity test for an LRU stack.

Corollary 1. *For an OPT cache of size j, a time interval $[t, t'[$ such that $s_t = s_{t'} = s > j$ and containing $s - j + 1$ cache misses must also contain at least $j - 1 + \beta$ cache hits.*

Proof. In $[t, t'[$, only $s - j$ distinct stack positions greater than j can be accessed. As there are $s - j + 1$ misses in $[t, t'[$, by the pigeonhole principle, there must exist a repetition, i.e., two references in $[t, t'[$ at the same stack position greater than j. By theorem 1, all $j - 1 + \beta$ stack positions between $2 - \beta$ and j must be accessed between the repetition. □

Corollary 2. *If the number of distinct items is bounded and equal to s, for any $j < s$*

$$\lim_{t \to \infty} \frac{M_{[0,t[}^j}{t} \leq \frac{s - j}{s - 1 + \beta}$$

I.e., the asymptotic miss ratio of an OPT cache of size j cannot exceed $\frac{s-j}{s-1+\beta}$.

Proof. There is a miss at a finite time t_0 such that $\forall t \geq t_0, s_t = s$. The next miss occurs at $t_1 > t_0$, the next one at $t_2 > t_1$, and so forth. An interval $[t_{k(s-j)}, t_{(k+1)(s-j)}]$ contains exactly $s-j+1$ misses. By corollary 1, there must be at least $j-1+\beta$ hits in $]t_{k(s-j)}, t_{(k+1)(s-j)}[$. Interval $[t_0, t_{k(s-j)}]$ contains at least $k(j-1+\beta)$ hits. That is, $[t_0, t_{k(s-j)}]$ contains exactly $k(s-j)+1$ misses and at least $k(s-j)+1+k(j-1+\beta) = k(s-1+\beta)+1$ references, that is, $t_{k(s-j)} - t_0 \geq k(s-1+\beta)$. As k grows to infinity, the impact of initial misses (before t_0) on the overall miss ratio vanishes, and the asymptotic miss ratio does not exceed $\frac{k(s-j)}{k(s-1+\beta)} = \frac{s-j}{s-1+\beta}$. □

For example, let us consider an 8-way set-associative cache ($j = 8$) managed by LRU, without bypassing ($\beta = 0$), and an application with a data set twice larger than the cache ($s = 16$). If the measured LRU miss ratio is significanly greater than $\frac{s-j}{s-1+\beta} = \frac{16-8}{16-1} \approx 53\%$, we do not need to run simulations with OPT to know that OPT outperforms LRU significantly on that application.

It should be noted that, for online replacement policies (such as LRU), the upper bound for the miss ratio is 100%. Indeed, one can generate an *ad hoc* trace defeating the replacement policy, by referencing at each time t an item X_t not in the cache at time t. Corollary (2) implies that it is *impossible* to defeat OPT when the number of distinct items is bounded. The content of the OPT cache at time t is not fixed, it depends on the item X_t we are going to choose. OPT knows the future and can anticipate our moves.

9 The OPT miss curve is convex

Recently, Beckmann and Sanchez argued that, under optimal replacement, the miss curve must be a convex function of the cache size [2]. They argue that, for any replacement policy with a non-convex miss curve, it is possible, by partitioning the cache in a certain way, to derive a policy yielding fewer misses. They assume that removing some items from a trace entirely is equivalent to enlarging the cache, whatever the replacement policy. This might be *approximately* valid, *empirically*. Whether this is always true is uncertain.

With Theorem 1, we can prove rigorously that the OPT miss curve is *always* convex.

Corollary 3. *For all t and for all $j \geq 3$,*

$$M^{j-1}_{[0,t[} - M^{j}_{[0,t[} \quad \leq \quad M^{j-2}_{[0,t[} - M^{j-1}_{[0,t[} \tag{21}$$

Proof. The terms on the left-hand and right-hand sides of inequality (21) are respectively the number of references in $[0, t[$ accessing position j in the OPT stack and the number of references accessing position $j-1$. The first reference in the trace accessing position $j-1$ (i.e., when the stack size is incremented from $j-2$ to $j-1$) occurs before the first reference at position j (when the stack size is incremented from $j-1$ to j). So (21) is true on the first access at position j. From theorem 1, between two accesses at position j, there must exist at least one access at position $j-1$. Therefore on each new access at position j, inequality (21) remains true. □

As noted by Beckmann and Sanchez, if a replacement policy's miss curve is markedly non convex, it is certainly far from the OPT miss curve [2].

10 The OPT tokens algorithm

The OPT matrix introduced in Section 8 considers all cache sizes simultaneously. When we focus on a single cache size, a simpler algorithm can be derived from the OPT matrix.

Notice that, for a fixed j, the s_t values $D_t^j(i)$ for $i \in [1, s_t]$ are sufficient to determine hits and misses in a cache of size j. In practice, we only need the values $D_t^j(i) \leq j$. Moreover, there are only j such distinct values. Let us generalize quantity I_t^j as follows:

$$\text{for } i \in [1, j], \qquad I_t^j(i) := \min\{k \in [1, s_t] : D_t^j(k) = i\}$$

We have $I_t^j(j) = I_t^j$. Note that $I_t^j(i) < I_t^j(i+1)$ (cf. definition (17)). From the update procedure (20), we obtain what we call the *OPT tokens* algorithm:

case	**action**	
start	$I_0^j(i) = i$	
$q_t > I_t^j(j)$	$\beta = 0$	$I_{t+1}^j(1) = 1$
miss	$i \geq 2 - \beta$	$I_{t+1}^j(i) = I_t^j(i) + 1$
$q_t \leq I_t^j(j)$		$I_{t+1}^j(1) = 1$
hit	$I_t^j(i) < q_t$	$I_{t+1}^j(i+1) = I_t^j(i) + 1$
	$I_t^j(i) \geq q_t$	$I_{t+1}^j(i+1) = I_t^j(i+1)$

(22)

The OPT tokens algorithm takes as input the LRU stack positions (q_t) generated from the input trace (X_t). Consistent with our choice to define $q_t = s_t + 1$ upon referencing an item for the first time, the first reference to an item occurring at a time t such that $s_t < j$ is counted as a hit. Misses can occur only when $s_t \geq j$. Table 4 shows how the OPT tokens algorithm works on an example with 6 distinct items referenced in a circular fashion, assuming cache size $j = 3$.

Figure 4 shows a visual for the OPT tokens. The number of tokens is equal to the cache size j. $I_t^j(i)$ is the height of the i^{th} token. An OPT miss occurs if q_t is above the highest token, i.e., $I_t^j(j)$. Otherwise this is an OPT hit. Then, the tokens are moved according to the rules (22).

11 Cache bypassing

Historical papers about optimal replacement studied paging in main memory [3, 13, 4]. They considered the caching problem without bypassing the cache (which is main memory here). To the best of our knowledge, Scott McFarling was the first to consider optimal replacement with bypassing [14]. McFarling modified the OPT policy as shown in Figure 2. Let us denote OPTb and OPTn the OPT policy with and without bypassing respectively.

We are not aware of any published proof that OPTb is optimal.[5] It is relatively straightforward, however, to take one of the several published optimality proofs for OPTn, e.g., [11], and to adapt it to prove the optimality of OPTb. We will not do this here.

In his thesis, McFarling proved the following:

[5] McFarling did not provide an actual proof that OPTb is optimal.

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
trace	A	B	C	D	E	F	A	B	C	D	E	F	A	B	C	D	E	F
q_t	1	2	3	4	5	6	6	6	6	6	6	6	6	6	6	6	6	6
without bypassing ($\beta = 0$)																		
$I_t^3(3)$	3	3	3	3	4	5	6	6	3	4	5	6	6	3	4	5	6	6
$I_t^3(2)$	2	2	2	2	3	4	5	2	2	3	4	5	2	2	3	4	5	2
$I_t^3(1)$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
miss				m	m	m			m	m	m			m	m	m		
with bypassing ($\beta = 1$)																		
$I_t^3(3)$	3	3	3	3	4	5	6	6	6	3	4	5	6	6	6	3	4	5
$I_t^3(2)$	2	2	2	2	3	4	5	5	2	2	3	4	5	5	2	2	3	4
$I_t^3(1)$	1	1	1	1	2	3	4	1	1	1	2	3	4	1	1	1	2	3
miss				m	m	m				m	m	m				m	m	m

Table 4: OPT tokens algorithm on an example with cache size $j = 3$.

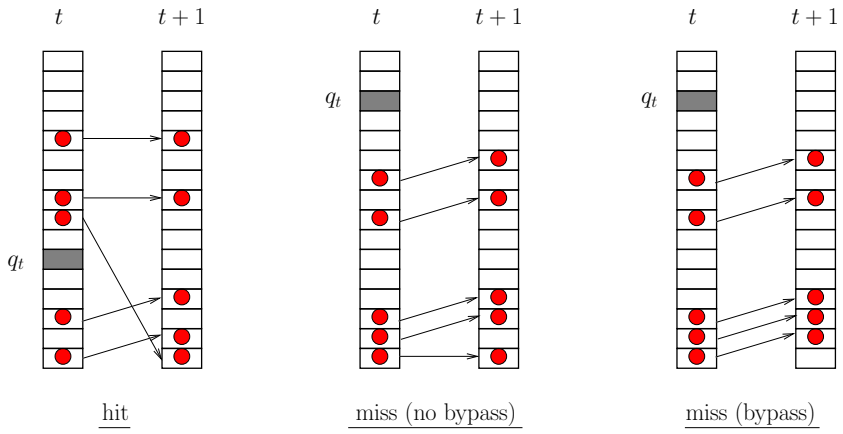

Figure 4: Visual for the OPT tokens. The number of tokens is equal to the cache size (here 5). If q_t (the LRU stack position of item X_t) is above the highest token, this is an OPT miss, otherwise this is an OPT hit.

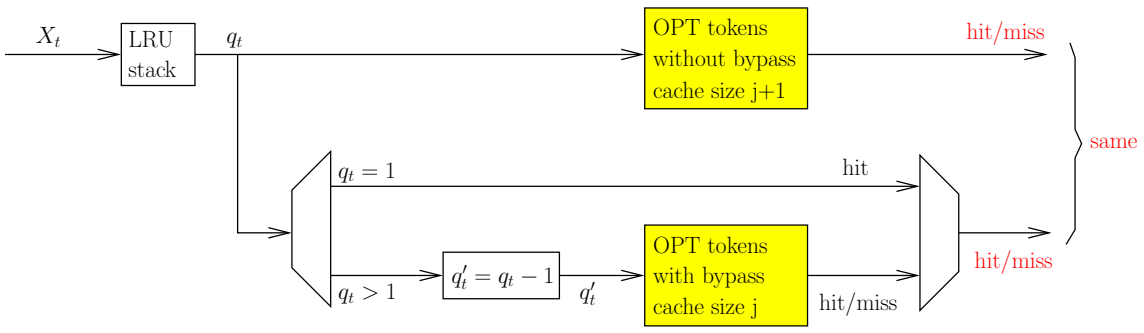

Figure 5: Equivalence between an OPT cache of size $j+1$ without bypassing and an OPT cache of size j with bypassing.

McFarling's inequality. *An OPTb cache of size j generates no more misses than an OPTn cache of size j and no more hits than an OPTn cache of size $j+1$.*

McFarling's argument is very simple. An OPTn cache cannot generate fewer misses than an equal-sized OPTb cache, as OPTb is optimal and outperforms all policies, including those like OPTn that never bypass the cache. For the other part of the inequality, McFarling gives the following argument. If one takes an OPTb cache of size j and stores in a victim cache of size 1 the item evicted from the OPTb cache or bypassing the OPTb cache, the OPTb cache and the victim cache together can be viewed as a cache of size $j+1$ that is not bypassed. Hence the OPTb+victim cache (and the OPTb cache alone) does not generate more hits than an OPTn cache of size $j+1$.

The OPT tokens algorithm provides another way to relate OPTn and OPTb. Upon an OPTn miss (see Figure 4), all the tokens but the lowest one move upwards. The lowest token never leaves the bottom position. Apart from that, OPTn and OPTb behave identically. More precisely, if we ignore hits at $q_t = 1$, an OPTn cache of size $j+1$ behaves like an OPTb cache of size j seeing reuse distances $q'_t = q_t - 1$. That is, the two caches yield exactly the same hits and misses. This equivalence is summarized in Figure 5.

For example, consider a random trace with s distinct items, i.e., such that all LRU stack positions in $[1, s]$ have equal probability $1/s$ to be accessed. Let us denote $m_n(j, s)$ and $m_b(j, s)$ the OPT miss ratios for an OPT cache of size j on random traces, without and with bypassing respectively. We mentioned in Section 7 that there is probably no simple mathematical formula for computing exactly m_n and m_b. Yet, from the equivalence of Figure 5, we can derive a simple equation relating m_n and m_b. A fraction $1/s$ of references are such that $q_t = 1$ (hit). After applying the transformation shown in Figure 5, we obtain a random trace such that all LRU stack positions in $[1, s-1]$ have equal probability to be accessed. This yields the following equality:

$$m_n(j+1, s) = (1 - \frac{1}{s}) \times m_b(j, s-1)$$

12 Circular traces

We already mentioned that, if the number s of distinct items is bounded, OPT cannot be defeated, i.e., its miss ratio is always below 100% (corollary 2). On the other hand, all online replacement policies can be defeated. In particular, it is well known that, when the cache size j is less than s, accessing s distinct items in a circular fashion (as in the example of Table 4) defeats LRU.

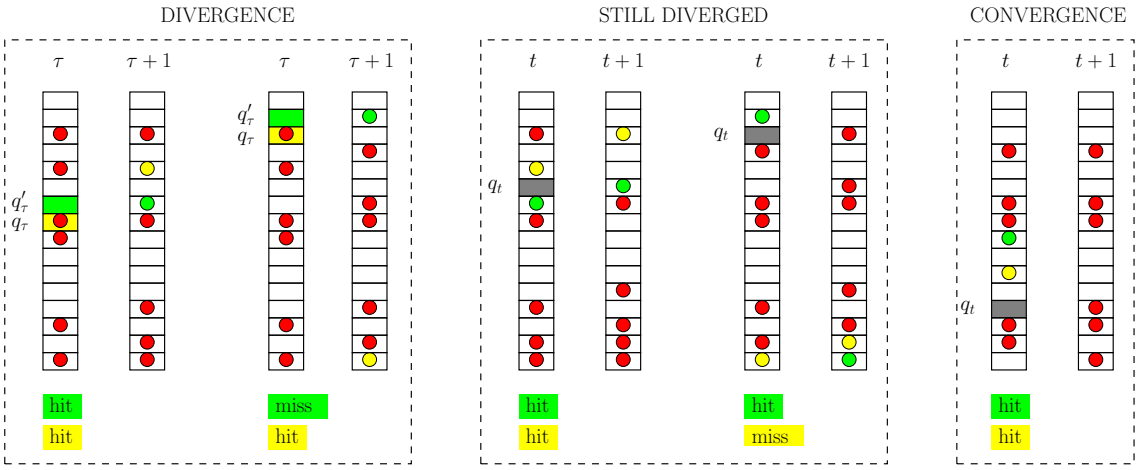

Figure 6: The different cases for proving proposition 1. At time τ, trace $\mathcal{T}$ accesses LRU stack position q_τ, and trace $\mathcal{T}'$ accesses LRU stack position $q'_\tau = q_\tau + 1$. For $t > \tau$, the two traces access the same LRU stack positions. The diverging token for trace $\mathcal{T}$ is colored in yellow, the diverging token for $\mathcal{T}'$ in green (the diverging token for a trace is not a token for the other trace; they are shown simultaneously for the sake of succinctness). The "leader" trace is the one whose diverging token is the lowest. When a miss for the leader trace is a hit for the other trace, the leader changes.

The OPT miss ratio on a circular trace can be calculated easily (see for instance [18] for an OPT cache without bypassing). One can check that the OPT tokens algorithm yields the correct result (see Table 4). For OPT with bypassing, the steady-state miss ratio on a circular trace is

$$\text{OPTb miss ratio on a circular trace} = \frac{s-j}{s}$$

For OPT without bypassing, we replace j by $j-1$ and s by $s-1$ in the expression above (cf. Figure 5) and we obtain $\frac{(s-1)-(j-1)}{s-1}$, that is,

$$\text{OPTn miss ratio on a circular trace} = \frac{s-j}{s-1}$$

The steady-state OPT miss ratio is equal to the upper bound stated by corollary 2. In other words, accessing a data set in a circular fashion maximizes the OPT miss ratio.

It is not coincidental that the trace with the worst OPT miss ratio is the one that maximizes q_t. We are going to show the following:

Proposition 1. *If two traces $\mathcal{T} = (X_t)$ and $\mathcal{T}' = (X'_t)$, generating reuse distances (q_t) and (q'_t) respectively, have the same length and are such that $q_t \leq q'_t$ for all t, then $\mathcal{T}$ does not generate more OPT misses than $\mathcal{T}'$.*

Proof. To show proposition 1, it is sufficient to start from a trace (q_t), increment one q_τ ($q'_\tau = q_\tau + 1$, $q'_t = q_t$ for $t \neq \tau$), and show that the number of misses does not decrease. Figure 6 provides examples illustrating the different cases. Before time τ, the two traces have exactly the same behavior. If q_τ does not coincide with a token, no divergence occurs, and the two traces continue to have the same behavior. Otherwise, if q_τ coincides with a token, the two traces diverge. Once diverged, the two sets of tokens (one set for each trace)

coincide except for one token, the *diverging token*. In Figure 6, the two diverging tokens (one for each trace) are shown simultaneously, the one for trace $\mathcal{T}$ colored in yellow, and the one for trace $\mathcal{T}'$ colored in green. We distinguish two situations. We say that trace $\mathcal{T}$ is the "leader" if its diverging token is lower than the diverging token of trace $\mathcal{T}'$. Otherwise, trace $\mathcal{T}'$ is the leader.

If there is divergence at τ, two situations can occur (left part of Figure 6):

1. both $\mathcal{T}$ and $\mathcal{T}'$ hit at τ: $\mathcal{T}'$ is the leader at $\tau + 1$
2. $\mathcal{T}$ hits and $\mathcal{T}'$ misses: $\mathcal{T}$ is the leader at $\tau + 1$

For $t > \tau$ and before convergence, two situations can occur (middle part of Figure 6):

1. $\mathcal{T}$ and $\mathcal{T}'$ both hit or both miss at t: the leader at t is the leader at $t + 1$
2. the leader trace misses and the other trace hits: the leader changes

Finally, convergence can occur only when both traces hit (right part of Figure 6). After convergence, the two sets of tokens coincide, and the two traces have exactly the same behavior.

Between divergence and convergence, for having a miss for $\mathcal{T}$ and a hit for $\mathcal{T}'$ at t, the leader at t must be $\mathcal{T}$. But if $\mathcal{T}$ is the leader at t, it means that there was previously a hit for $\mathcal{T}$ and a miss for $\mathcal{T}'$. Hence $\mathcal{T}$ does not generate more misses than $\mathcal{T}'$. □

From proposition 1, it is obvious why circular accesses are worst for OPT. Sometimes, proposition 1 can also provide a lower bound for the number of OPT misses:

Corollary 4. *For an OPT cache of size j, on an infinitely long trace, if there exists a finite time τ such that for all $t \geq \tau$, $q_t \geq q_{min} > j$, the asymptotic OPT miss ratio is greater than or equal to $\frac{q_{min}-j}{q_{min}-1+\beta}$.*

Proof. We transform trace (q_t) into (q'_t) such that $q'_t = q_t$ for $t < \tau$ and $q'_t = q_{min}$ for $t \geq \tau$. From proposition 1, the OPT miss ratio of (q_t) is greater than or equal to that of (q'_t). For $t \geq \tau$, trace (q'_t) has a circular behavior (i.e., q'_t is constant). Whatever the positions of tokens at τ, the asymptotic miss ratio of (q'_t) is $\frac{q_{min}-j}{q_{min}-1+\beta}$. □

If a trace is such that q_t is not constant but remains in an interval $[q_{min}, q_{max}]$, the asymptotic OPT miss ratio is in the interval $[\frac{q_{min}-j}{q_{min}-1+\beta}, \frac{q_{max}-j}{q_{max}-1+\beta}]$. If q_{min} and q_{max} are close, we know approximately the OPT miss ratio.

13 Conclusion

Belady's MIN algorithm and Mattson's OPT replacement policy provide two different ways to think about optimal replacement. This paper brings three new equivalent ways to reason about OPT hits and misses: equation (14), the OPT matrix, and the OPT tokens. With these conceptual tools, we have discovered and proved several facts that were previously unknown, and we have proved rigorously the convexity of OPT miss curves. Theorem 1 highlights a remarkable structure in traces of OPT distances, and offers a very simple way to detect bugs in OPT stack implementations, just by a visual inspection of the output. Also, we found that the notion of reuse distance, which is naturally related to the LRU policy, is also useful for understanding optimal replacement. Probably, many mathematical facts about optimal replacement remain to be discovered.

Though the goal of this paper is to improve our understanding of optimal replacement, some of our results might lead to practical findings. For instance, the Hawkeye cache replacement policy is directly inspired from Belady's MIN algorithm [9]. The OPT tokens algorithm might inspire alternate implementations of the Hawkeye policy.

References

[1] A. V. Aho, P. J. Denning, and J. D. Ullman. Principles of optimal page replacement. *Journal of the ACM*, 18(1), January 1971.

[2] N. Beckmann and D. Sanchez. Talus: A simple way to remove cliffs in cache performance. In *International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[3] L. A. Belady. A study of replacement algorithms for virtual-storage computers. *IBM Systems Journal*, 5(2), 1966.

[4] L. A. Belady and F. P. Palermo. On-line measurement of paging behavior by the multivalued MIN algorithm. *IBM Journal of Research and Development*, 18(1), January 1974.

[5] J. Brock, X. Gu, B. Bao, and C. Ding. Pacman: program-assisted cache management. In *International Symposium on memory Management (ISMM)*, 2013.

[6] D. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. *ACM SIGARCH Computer Architecture News*, 25(3), June 1997.

[7] E. G. Coffman and P. J. Denning. *Operating Systems Theory*. Prentice-Hall, 1973.

[8] A. Cohen and W. A. Burkhard. A proof of the optimality of the MIN paging algorithm using linear programming duality. *Operations Research Letters*, 18(1), August 1995.

[9] A. Jain and C. Lin. Back to the future: leveraging Belady's algorithm for improved cache replacement. In *International Symposium on Computer Architecture (ISCA)*, 2016.

[10] D. E. Knuth. An analysis of optimum caching. *Journal of Algorithms*, 6(2), June 1985.

[11] M.-K. Lee, P. Michaud, J. S. Sim, and D. Nyang. A simple prooof of optimality for the min cache replacement policy. *Information Processing Letters*, 116(2), February 2016.

[12] W.-F. Lin and S. Reinhardt. Predicting last-touch references under optimal replacement. Technical Report CSE-TR-447-02, University of Michigan, 2002.

[13] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2), 1970.

[14] S. McFarling. *Program analysis and optimization for machines with instruction cache*. PhD thesis, Stanford University, 1991.

[15] T. R. Puzak. *Analysis of cache replacement algorithms*. PhD thesis, University of Massachusetts - Amherst, 1985.

[16] K. Rajan and R. Govindarajan. Emulating optimal replacement with a Shepherd cache. In *International Symposium on Microarchitecture (MICRO)*, 2007.

[17] A. J. Smith. Analysis of the optimal, look-ahead demand paging algorithms. *SIAM Journal on Computing*, 5(4), December 1976.

[18] J. E. Smith and J. R. Goodman. Instruction cache replacement policies and organizations. *IEEE Transactions on Computers*, C-34(3), March 1985.

[19] R. A. Sugumar and S. G. Abraham. Efficient simulation of caches under optimal replacement with applications to miss characterization. In *ACM SIGMETRICS*, 1993.

[20] O. Temam. An algorithm for optimally exploiting spatial and temporal locality in upper memory levels. *IEEE Transactions on Computers*, 48(2), February 1999.

[21] B. Van Roy. A short proof of optimality for the MIN cache replacement algorithm. *Information Processing Letters*, 102(2), April 2007.

[22] W. Vogler. Another short proof of optimality for the MIN cache replacement algorithm. *Information Processing Letters*, 106(5), May 2008.