A Universal Online Caching Algorithm Based on Pattern Matching[*]

Gopal Pandurangan and Wojciech Szpankowski[†]
Department of Computer Science
Purdue University
W. Lafayette, IN 47907 U.S.A.
Email: {gopal,spa}@cs.purdue.edu

Abstract

We present a universal algorithm for the classical online problem of *caching* or *demand paging*. We consider the caching problem when the page request sequence is drawn from an *unknown* probability distribution and the goal is to devise an efficient algorithm whose performance is close to the *optimal online* algorithm which has *full knowledge* of the underlying distribution. Most previous works have devised such algorithms for specific classes of distributions with the assumption that the algorithm has full knowledge of the source. In this paper, we present a *universal* and simple algorithm based on pattern matching for mixing sources (includes Markov sources). The expected performance of our algorithm is within $4 + o(1)$ times the optimal online algorithm (which has full knowledge of the input model and can use unbounded resources).

Keywords: Online Computation; Caching; Universal Algorithm; Stochastic Model.

1 Introduction

A fundamental algorithmic goal in online computation is to design *universal* algorithms, i.e., algorithms that work well *without knowledge* of the input (source) model. In many applications, no *a priori* knowledge of the source characteristics is available and statistical tests are either impossible, unreliable, or too costly [24]. Universal algorithms can overcome these difficulties and indeed have been shown to be very useful in important practical applications, e.g., prefetching [4] and data compression [24, 25]. For example, the universal data compression scheme of Ziv and Lempel [25], achieves an asymptotic optimal compression (equal to the entropy rate of the source) without any knowledge of the input source model. This algorithm is universal over a large model class, namely *stationary ergodic sources* [3].

The focus of this paper is the classical online problem of *caching* or *(demand) paging* [5]. In this problem, we have a finite collection $\mathcal{A} = \{a_1, a_2, \ldots, a_N\}$ of pages in memory and a cache of size k (typically $k \ll |\mathcal{A}| = N$). The cache can hold a subset of $\mathcal{A}$ of size up to k pages. Given a page request, say a_i, if it is not in the cache we incur a *page fault*. Further, if there is a page fault, a_i has to be placed in the cache. The issue is, if the cache is full, some page must be evicted from the cache to make room for the current page a_i. In order to minimize the number of page faults (associated with future page requests), the choice of which page to evict is crucial. The page requests come one after the other in an online fashion and each page request has to be handled immediately without knowledge of future requests.

In this paper, we present a universal algorithm for online caching. In our setting the page request sequence is drawn from an *unknown* probability distribution (e.g., a mixing source) and our goal is to devise an efficient online algorithm whose performance is close to that of an *optimal online* algorithm which has *full knowledge* of the underlying distribution. The performance is measured by a *competitive ratio* parameter which is defined as ratio of the number of page faults incurred by our algorithm to that of the optimal online algorithm. Our goal is to design an online algorithm that competes well (has a low competitive ratio) with the optimal online algorithm, even though our algorithm has no knowledge of the underlying distribution.

Most previous works have devised online caching algorithms for specific classes of distributions with the assumption that the algorithm has full knowledge of the source. For example, Karlin, Phillips, and Raghavan [12] present efficient algorithms (that run in time polynomial in the cache size) when the request sequence is generated by a Markov chain under the assumption that the online algorithm has complete knowledge of the Markov chain. The authors of [12] assume that the Markov chain can be "learned" from looking at a very long input. (Similarly, Franaszek and Wagner [8] give an optimal algorithm for memoryless sources.) Although this is possible in principle when you know the class of the model (say memoryless, Markov Chain), it is not clear how in general the errors in learning will affect the performance of the online algorithm. The problem becomes more complicated when we do not have knowledge of the class or the order of the underlying Markov process (cf. [14, 15]).

This paper is motivated by the work of Lund et al. [13] on caching strategy and universal prediction based on pattern matching due to Jacquet et al. [11]. The authors of [13] propose an efficient randomized 4-competitive online caching algorithm that works for *any* distribution $\mathcal{D}$ but it needs to know for (each pair of) pages a and b the probability that a will next be requested before b. It is also remarked that even if the algorithm can determine the probabilities approximately,

it would be ensured competitiveness. However, it is not clear how to compute these probabilities efficiently and sufficiently accurately when we have very little knowledge of the distribution or the class it belongs to (e.g., how to know the order of Markov process).

We present a simple caching algorithm based on pattern matching that is universal for the class of mixing sources (includes Markov sources). We show that our universal algorithm gives an expected performance that is within $4 + o(1)$ times the optimal online algorithm (which has full knowledge of the input model and can use unbounded resources). Our universal algorithm uses the DOMinating-distribution (DOM) algorithm of Lund et al. [13]. This algorithm assumes that, for each pair of pages a and b, the probability that b will next be requested before a is known at every time step t. We address the problem of estimating these probabilities at each time t, assuming that the request sequence is generated by a mixing source. The main contribution is to show how to estimate the above probabilities using a pattern matching algorithm of Jacquet et al. [11].

The rest of the paper is organized as follows. In Section 2 we give more background on the caching problem and discuss prior research. In Section 3 we give our universal online caching algorithm. In Section 4 we analyze our algorithm and state our main theorem. We conclude in Section 5 with issues left for further work.

2   Background and Related Work

Caching is related to *prefetching* problem, and the latter is essentially the same as the *prediction problem*, studied extensively in information theory [15]. In both problems, we have a collection $\mathcal{A}$ of pages in memory and a cache of size k (typically $k \ll |\mathcal{A}|$). Given a page request (from $\mathcal{A}$), if the page is not in the cache we incur a page fault, otherwise we don't, and in both problems, we are interested in minimizing the number of page faults. However, in prefetching, we are allowed to *prefetch k* items to the cache prior to each page request, while in caching, we are not allowed to prefetch pages which are fetched only *on demand*.

Both these problems can be formulated as online decision problems [2, 14, 23] as follows. We are given a temporal sequence of observations (in other words, a request sequence) $x_1^n = x_1, x_2, \ldots, x_n$, for which corresponding actions $b_1, b_2, \ldots, b_n$ result in instantaneous losses $l(b_t, x_t)$, for each time instant t, $1 \leq t \leq n$, where $l(.,.)$ denotes a non-negative loss function. The action b_t, for all t, is a function of the previous observations x^{t-1} only; hence the sequence of actions can be considered as an online algorithm or strategy. A normalized loss

$$L = \frac{1}{n} \sum_{t=1}^{n} l(b_t, x_t) \tag{1}$$

accumulates instantaneous loss contributions from each action-observation pair and the objective of the online strategy is to minimize this loss function. Prediction and prefetching can be thought of as sequential decision problems with memoryless loss functions i.e., the loss does not depend on previous action-request pairs. On the other hand, in caching, the loss function is not memoryless and this is one reason why designing optimal online strategies for caching is more complicated in general than prefetching or prediction (discussed more below; see also [15]).

One can study such online decision problems in two settings: *a probabilistic framework*, in which the sequence of requests is viewed as a sample of a random process; or using an *individual*

3

sequence approach i.e., *comparing* the performance of the online strategy for an *arbitrary* sequence with certain classes of competing *offline* strategies — such as in the *sequential decision approach* [10, 14] (where the online strategy is compared with the best *constant offline* algorithm which has full knowledge of the given sequence of observations), or with *finite state machines* [6]).

In the probabilistic setting, universal algorithms have been well-studied for prefetching and prediction problems. For example, Vitter and Krishnan [22] considered a model where the sequence of page requests is assumed to be generated by a *Markov source*. They show the fault rate of a Ziv-Lempel based prefetching algorithm approaches the fault rate of the best prefetcher (which has full knowledge of the Markov source) for the given Markov source as the page request sequence length $n \to \infty$. In fact, a general result in a probabilistic setting was shown by Algoet [2]: if the request sequence is generated by a stationary ergodic process then it is shown that the optimum strategy is to select an action that minimizes the conditional expected loss given the currently available information at each step and this strategy is shown to be asymptotically optimal in the sense of the strong law of large numbers. In the individual sequence approach, we refer to the work of Feder et al. [6] on predicting binary sequences (corresponding to prefetching in a universe of two pages with cache of size 1).

Thus while there has been a lot of work on universal algorithms for prefetching (and prediction) — both in the probabilistic setting and in the individual sequences approach (see e.g., [14] for a survey), there has not been much work for the more difficult problem of online caching except perhaps the recent work of Merhav et al. [15] on sequential strategies. In [15] the authors assume that the loss function depends also on the past action-observation pairs. In particular, at time t the loss function $l(b_{t-1}, b_t, x_t)$ depends on the current and previous decisions.

The difference between prefetching and caching is also discussed in [18] where it is shown that, in a probabilistic setting, entropy can "characterize" the performance of the best prefetching algorithm, but, in general, entropy alone is not sufficient to characterize the performance of the best caching algorithm. However, if the request sequence is generated by a memoryless source then the best caching algorithm is essentially the same as the best prefetching algorithm and one can give explicit bounds on the fault-rate of the best algorithm in terms of the entropy of the source [18].

In the theoretical computer science literature, however, the online caching problem has received a lot of attention and, in fact, was one of the first problems to be analyzed under the framework of *competitive analysis* where the performance of the online algorithm is compared with the *best offline* algorithm [19]. For online caching (or demand paging) the well known LRU (Least Recently Used) has a competitive ratio of k [19], where k is the cache size, while the randomized MARKER algorithm is $O(\log k)$ competitive [7]. In fact, it is known that any deterministic algorithm for caching has a competitive ratio of at least k and any randomized algorithm has a competitive ratio of $\Omega(\log k)$. We note, that for the problem of prefetching, competitive analysis is meaningless as the optimal offline algorithm will always prefetch the correct item and hence incurs no cost.

In this paper, we take the probabilistic approach, and assume that the loss function represents the page fault. Following the work of Lund et al. [13], we compare the performance of our algorithm to the optimal *online algorithm* (henceforth called as *ON*) which has full knowledge of the input distribution. The optimal online caching strategy (assuming full knowledge of the underlying distribution) is known for memoryless sources ([1, 8, 18]) and for Markov sources (*l*-order Markov)

4

[5]. While the best online strategy is easy for memoryless sources (simply keep the $k-1$ pages with the highest probabilities in the cache), the best strategy for higher order sources (in particular, even when the request sequence is generated by a Markov chain) is nontrivial, and involves computing the optimal policy in a Markov decision process (MDP) [12, 18]. In particular, many "natural" online strategies such as LAST (i.e., on a fault, evict the page that has the highest probability of being the last of the k pages in the cache to be requested), MAX-REACH-TIME (i.e., on a fault for page r, evict that page whose expected time to be reached from r is maximum) perform poorly even on Markov chains [12]. On the other hand, for prefetching, the optimal universal strategies (e.g., see Algoet [2]) are somewhat more "natural" and intuitive. However, known methods for computing this optimal online strategy takes time exponential in k and this has motivated work on computing *near-optimal* online strategies (which closely approximate the performance of *ON*) which take time polynomial in k ([12, 13]); however, these results, as mentioned earlier, assume full knowledge of the Markov source, and hence not universal. The universal caching algorithm of this paper works for mixing sources (this includes Markov sources) and has a performance that is within a constant factor of the optimal online algorithm.

In the probabilistic setting, one can also compare the performance of universal algorithms with *offline strategies*, e.g., with the optimal offline algorithm (that has access to the request string output by the source, and serves it optimally). We remark that the performance of *ON* will typically have a higher expected cost than the optimal offline algorithm, and in the worst case, *ON* has a cost which is a factor of at most $\Theta(\log k)$ of the optimal offline algorithm, and this immediately implies that our universal algorithm gives a performance that is $O(\log k)$ times the optimal offline algorithm.

3   The Caching Algorithm

To motivate our algorithm we first briefly describe the DOMinating-distribution (DOM) algorithm of Lund et al. ([13, 5]). DOM *assumes* that for a given distribution $\mathcal{D}$, one can compute for all distinct pages a and b the probability $p(a,b)$ that b will be requested before a (such a distribution is called *pairwise-predictive*). A pairwise-predictive distribution $\mathcal{D}$ and an online paging algorithm *ALG* naturally induce a *weighted tournament* as follows. A weighted tournament $T(S,p)$ is a set of states S and a (probability) weight function $p : S \times S \to [0,1]$ satisfying the property that $p(a,b) + p(b,a) = 1$ for all $a \neq b$ in S and $p(a,a) = 0$ for all $a \in S$. Given a pairwise predictive distribution $\mathcal{D}$ and paging algorithm *ALG*, the weight function p is determined by $\mathcal{D}$ (just before each new request), and S will be the set of *ALG*'s pages in the cache. A *dominating distribution* $\tilde{p}$ for a tournament $T(S,p)$ is a probability function $\tilde{p} : S \to [0,1]$ such that for every $a \in S$, if $b \in S$ is chosen with probability $\tilde{p}(b)$, then $E_{\mathcal{D},\tilde{p}}[p(a,b)] \leq 1/2$ (i.e., the expectation is taken with respect to both $\mathcal{D}$ and $\tilde{p}$). It follows that, for every a in the cache, if b in the cache is chosen with probability $\tilde{p}(b)$, then with probability $\geq 1/2$, a's next request will occur no later than b's next request. Lund et al. show the following key lemma on dominating distributions:

Lemma 3.1 ([13]) *Every weighted tournament $T(S,p)$ has a dominating distribution and such a*

distribution can be found by solving the following linear program consisting of $|S|$ variables:

$$\min \ z$$

$$\text{subject to:} \quad \sum_{b \in S} p(a,b)\tilde{p}(b) \ \leq \ z \quad (\forall a \in S),$$

$$\sum_{b \in S} \tilde{p}(b) \ = \ 1, \quad \tilde{p}(b) \geq 0, \quad (\forall b \in S)$$

The solution to the above linear program is at most 1/2.

To summarize DOM, let $x = x_1, x_2, \ldots$ be a request sequence. The DOM algorithm is as follows.

1. On the tth request x_t, if x_t is a page fault (otherwise do nothing), then (as determined by $\mathcal{D}$) construct a weighted tournament $T_t(S,p)$ on the k pages presently in the cache.

2. Evict page a with probability $\tilde{p}_t(a)$ where $\tilde{p}_t$ is the dominating distribution for the tournament $T_t(S,p)$.

The performance of DOM can be analyzed using the following key lemma.

Lemma 3.2 ([13]) *Let A denote a caching algorithm. Assume that each time A evicts a page a chosen from some distribution following property holds: for every page b in cache, the probability that b is next requested no later than a is at least $1/c$, where $c \geq 1$ is some fixed constant. Let A_n and ON_n denote the number of page faults incurred by A and the optimal online algorithm (ON) respectively after n requests. Then $E[A_n] \leq 2cE[ON_n]$.*

Applying the above lemma to the DOM algorithm yields:

Theorem 3.1 ([13]) *For all request sequences x from $\mathcal{D}$, the following holds: $E[DOM(x)] \leq 4 \cdot ON(x)$. Furthermore, the complexity per page fault is bounded by a polynomial in k, the cache size, assuming that for all distinct pairs of pages a and b, we have precomputed $p(a,b)$.*

We now propose a new universal caching algorithm that uses the idea of the Sampled Pattern Matching (SPM) [11] to obtain a good estimate of the probability that page b occurs before page a (Steps 1-3 below). We then apply the caching strategy of [13] to evict a page upon a fault (Step 4). We will show that the expected page fault rate of our algorithm will be at most $4 + o(1)$ times ON, the optimal online algorithm. First we state our algorithm below.

Universal Caching Algorithm:
Let $x_1, x_2 \ldots$ be the request sequence. Let $1/2 < \alpha < 1$ be a fixed constant, and $k = |C|$, be the cache size.
If x_n is not in the cache C and C is full do:

1. Find the largest suffix of $x_1^n = x_1, \ldots, x_n$ whose copy appears somewhere in the string x_1^n. Call this the *maximal suffix* and let its length be D_n.
2. Take an α fraction of the maximal suffix of length $k_n = \lceil \alpha D_n \rceil$, i.e., the suffix $x_{n-k_n+1} \ldots x_n$.

Each occurrence of this suffix in the string x_1^n is called a *marker*. Let $K_n \geq 2$ be the number of occurrences of the marker in x_1^n.

3. For every pair of elements a, b in C, estimate the probability $P(a, b)$ that b will occur before a after the marker position as follows: Let $Y_j(a, b)$ be the indicator r.v. for the event that b occurs before a in the substring that starts after the jth marker, $1 \leq j \leq K_n$. Then the estimator is

$$\tilde{P}(a, b) = \frac{\sum_{j=1}^{K_n} Y_j(a, b)}{K_n}.$$

4. Compute a distribution p by solving the following Linear Program (LP) in which we minimize z subject to

$$\sum_{b \in C} \tilde{P}(a, b) p(b) \leq z \quad (\forall a \in C),$$

$$\sum_{b \in C} p(b) = 1, \quad p(b) \geq 0, \quad (\forall b \in C)$$

(Note that the above LP is the same as the one mentioned in Lemma 3.1.) We will show that the above LP has a feasible solution for some $z \in [0, 1]$ (whose value will be proved to be $\leq 1/2 + 1/n^\theta$ for some $\theta > 0$). Thus, for each page a in C, if b is chosen according to p, then $E[\tilde{P}(a, b)] \leq 1/2 + 1/n^\theta$. Choose a page to evict from C according to the distribution p.

The above algorithm can be naturally implemented by maintaining a suffix tree [9]. The longest suffix, markers, the delay sequences and the estimates (Steps 1-3), can be computed efficiently from a suffix tree. The suffix tree of $x_1, \ldots, x_n$ is a *trie* (i.e., a digital tree) built from all suffixes of $x_1, \ldots, x_n\$$ where $\$$ is a special symbol that does not belong to the alphabet $\mathcal{A}$. External nodes of such a suffix tree contain information about the the suffix positions in the original string and the substring itself that leads to this node. In addition, we keep pointers to those external nodes that contain suffixes ending with the special symbol $\$$ (since one of them will be the longest suffix that we are looking for; in fact, the one with the longest path). It is very easy to find all markers once the suffix tree is built. Indeed, they are located in the subtree that can be reached following the last $\lceil \alpha D_n \rceil$ symbols of the longest suffix.

Given a suffix tree on n nodes, the worst case time to do these operations is $O(n)$ (cf. [9]), but on average will take only $O(n^{1-\alpha})$ (for some $\alpha >> 1/2$) since there are only so many markers with high probability (whp)[1]([11]) and the delay is $O(\log 2n)$ whp (cf. Section 4). Moreover, it is easy to update the suffix tree when the new symbol x_{n+1} is added. The only nodes that we must look at are the ones with $\$$ to which we keep pointers. In the worst case, we need to inspect $O(n)$ nodes, but on average only $O(n^{1-\alpha})$ [11]. Step 4 can be implemented by solving a LP in k variables and hence the running time is polynomial in the size of the cache.

4 Main Result and Analysis

Throughout, we assume that the request sequence $X_1, X_2, \ldots, X_n$ is generated by a stationary (strongly) mixing source over a finite alphabet $\mathcal{A}$ (the cache size is $k < |\mathcal{A}|$) (cf. [20]).

---

[1]Throughout this paper, this means with probability at least $1 - 1/n^\nu$ for some constant $\nu >> 0$.

Definition 4.1 (MX - (Strongly) ϕ-Mixing Source) *Let $\mathcal{F}_m^n$ be a σ-field generated by $X_m^n = X_m X_{m+1} \ldots X_n$ for $m \le n$. The source is called* mixing, *if there exists a bounded function $\phi(g)$ such that for all $m, g \ge 1$ and any two events $A \in \mathcal{F}_1^m$ and $B \in \mathcal{F}_{m+g}^{\infty}$ the following holds:*

$$(1 - \phi(g))\mathsf{Pr}(A)\mathsf{Pr}(B) \le \mathsf{Pr}(AB) \le (1 + \phi(g))\mathsf{Pr}(A)\mathsf{Pr}(B).$$

If, in addition, $\lim_{g \to \infty} \phi(g) = 0$, then the source is called strongly mixing.

Strongly mixing sources include *memoryless* sources (mixing with $\phi(g) = 0$) and *Markov sources* over a finite alphabet (mixing with $\phi(g) = O(\gamma^g)$ for some $\gamma < 1$) [20]. For our analysis below we assume that our ϕ mixing coefficient decays faster than the reciprocal of every polynomial function, i.e., $\phi(g) = o(1/g^\gamma)$ for every $\gamma > 0$. Our main result is formulated next.

Theorem 4.1 *Let A_n and ON_n denote the number of page faults incurred by our algorithm and the optimal online algorithm (ON) respectively after n requests from a strongly mixing source. Then, for a positive constant δ, we have*

$$E[A_n] \le (4 + O(\frac{1}{n^\delta}))E[ON_n]$$

as $n \to \infty$.

The rest of this section is devoted to the proof of Theorem 4.1. We start with reviewing some results of [11] needed for the proof of our result.

We need the following results from [11]:

1. *Marker separation property*: There exists $\epsilon >> 0$ such that for some constant $\alpha \in (1/2, 1)$ whp as $n \to \infty$ two consecutive markers (i.e., copies of α portion of the longest suffix) in the string X_1^n cannot be closer than n^ϵ positions. A consequence of the separation property is that the number of markers (i.e., K_n) is n^β whp for some constant $\beta \in (0, 1)$.

2. *Marker stability property*: There exists $\epsilon > 0$ such that whp no modification of any of the $\lceil n^\epsilon \rceil$ symbols following a marker will transform the string X_1^n into another string $\tilde{X}_1^n$ with a new set of markers.

Let L denote the maximum *delay* before we see all the symbols in the (current) cache C after any marker, i.e., $L = max_{1 \le j \le K_n} L_j$ where L_j the delay before we see all symbols after the jth marker.

Lemma 4.1 $L = O(\log 2n)$ *whp.*

Proof. Let X_i be the first symbol after the end of a marker and consider the next $c \log 2n$ symbols starting from X_i, i.e., the subsequence $X_i^{i+c \log^2 n - 1}$ where $c > 0$ is a suitably large fixed constant. We first bound the probability that a particular symbol (say a) in C will *not* occur in the above subsequence. Partitioning this subsequence into blocks of size $\log n$ and using the mixing property the above probability is bounded by

$$(1 + \phi(\log n)))^{c \log n}(1 - p_a)^{c \log n} \le 1/n2,$$

8

for a suitably large constant c, where p_a is the (unconditional) probability of occurrence of symbol a in the sequence. (Since we consider a finite alphabet and the source is stationary and ergodic, p_a is some positive constant independent of n.) Applying the union bound ([16][Lemma 1.2]), we have that all symbols in C occur in $X_i^{i+c\log^2 n}$ (for some suitably large constant c) with probability $1 - 1/n$. Thus the delay for this marker is $O(\log 2n)$ with probability at least $1 - 1/n$. We appeal to the marker separation property and the fact that are at most n^β markers whp to conclude that whp that the delay is $O(\log 2n)$ after every marker. ∎

In the next lemma we prove that our estimator $\tilde{P}(a, b)$ is consistent. We need one more definition from [11], namely the concept of *favorite strings*.

Definition 4.2 (Favorite String) *Fix a constant $\epsilon > 0$. Let i_j be the position after the last symbol of marker j, $1 \leq j \leq K_n$. A favorite string is one for which any modification of any $\lceil n^\epsilon \rceil$ symbols following a marker does not change the position of any marker and the delay L_j after any marker is $O(\log 2n)$.*

Lemma 4.2 *Let $\theta \in (0,1)$ be a suitably small positive constant. The estimators $\tilde{P}(a,b)$ for every pair of symbols a and b in cache are within $1/n^\theta$ of the true estimates whp for sufficiently large n.*

Proof: The main idea of the proof is to show that that the random variables $Y_j(a, b)$, $1 \leq j \leq K_n$ (computed in Step 3) are almost independent. Let F_n be the set of favorite strings as defined above: $F_n = \{X_1^n : X_1^n \text{ is a favorite string}\}$. The marker separation and stability properties and Lemma 4.1 imply that whp any string is a favorite. Consider the *delay subsequence*: $X_{i_j}^{i_j + c \log 2n - 1}$, $1 \leq j \leq K_n$, (c is a suitably large constant), i.e., the subsequence consisting of $O(\log 2n)$ symbols after every marker. Lemma 4.1 and the two marker properties guarantee that whp the markers are stable and the delays after are separated by n^ϵ for some $\epsilon > 0$. Using this and arguments similar to proof of Lemma 7 in [11], it follows that the delay subsequence is mixing if $X_1^n \in F_n$.

The mixing property of the delay subsequence implies that, for favorite strings, the probability distribution of the Y_js are within factor of $(1 \pm O(\phi(n^\epsilon)))^{K_n}$ from an i.i.d. sequence (this follows from the definition of a mixing source). Let $P(a, b)$ be the true estimate. We bound the probability of error as follows:

$$\Pr\left(|\tilde{P}(a, b) - P(a, b)| \leq n^{-\theta}\right) \leq \Pr\left(|\tilde{P}(a, b) - P(a, b)| \leq n^{-\theta}; \ X_1^n \in F_n\right) + \Pr(X_1^n \notin F_n)$$

Since for a favorite string the Y_js are within factor of $(1 \pm O(\phi(n^\epsilon)))^{K_n}$ from an i.i.d sequence, we bound the first probability on the right hand side by using a Chernoff bound ([17, Chapter 4]) and noting that whp (say, $1 - 1/n^\rho$) K_n is n^β for some $\beta \in (0, 1)$:

$$\Pr\left(|\tilde{P}(a, b) - P(a, b)| \leq n^{-\theta}\right) \leq (1 + O(\phi(n^\epsilon)))^n e^{-\frac{1}{3} n^\beta P(a,b) n^{-2\theta}} + \Pr(X_1^n \notin F_n) + O(1/n^\rho) = O(1/n^\nu)$$

for large n and for some constant $\nu > 0$. In the above we used the fact that $K_n \leq n$ and $\phi(g) = o(1/g^\gamma)$ for every $\gamma > 0$. ∎

We are now ready to prove our main result.

Proof of Theorem 4.1. We first show that the LP stated in Step 4 has a feasible solution for $z \leq 1/2 + 1/n^\theta$ whp. We use the approach in [13]. Consider the LP (cf. Step 4 of the Universal Caching Algorithm):

$$\min\ z$$
$$\text{subject to :} \quad \sum_{b \in C} \tilde{P}(a,b)p(b) \quad \leq \quad z \quad (\forall a \in C),$$
$$\sum_{b \in C} p(b) \quad = \quad 1, \quad p(b) \geq 0, \quad (\forall b \in C)$$

For the purpose of analysis, appealing to Lemma 4.2, we can rewrite (whp) the first constraint as:

$$\sum_{b \in C} P(a,b)p(b) \leq z - O(1/n^\theta) \quad (\forall a \in C)$$

where $P(a,b)$ is the true estimate of the probability that b will be requested before a and θ is a suitably small positive constant. By considering the dual LP, we can show that the solution of the above LP is at most $1/2 + O(1/n^\theta)$ ([13]). By Lemma 4.2, this holds with probability at least $1 - 1/n^\nu$ for some $\nu > 0$. When our algorithm has a page fault and must evict a page, let a be a random variable denoting the page that is evicted. Now the following property holds, by definition of the algorithm: for every page b in C, the probability that b is next requested no later than a is at least $1/2 - O(1/n^\theta) - O(1/n^\nu)$. By Lemma 3.2, we conclude that the expected number of page faults is at most $4 + O(1/n^\delta)$ times the optimal online algorithm, for a positive constant δ, as $n \to \infty$. ∎

5 Concluding Remarks and Further Work

Our pattern matching approach appears to be quite general and can be a useful technique for designing universal caching algorithms. The technique gives a way to "convert" a caching algorithm that is not universal to begin with into one which is universal. We illustrated by showing how to design a universal caching algorithm based on the DOM algorithm.

There are a few open questions for future work. From the algorithmic complexity point of view, our universal algorithm is not efficient, since it has the drawback of constructing a suffix tree after each eviction. Moreover, it has to keep and deal with a large sequence of requests in memory in order to use the sampling algorithm. One approach that can be used to get faster and space-efficient algorithms is to use a *fixed database approach* [20] to "store" a fixed long sequence and then bound the errors that can accumulate. In this approach, we are given a database (offline training sequence) that is utilized to pre-compute the estimator $\tilde{P}(a,b)$ for all $a, b \in \mathcal{A}$. It is assumed that the request sequence and the database sequence are independent and identically distributed. Clearly, now we need only to construct a suffix tree of a given database sequence. Another approach would be to use a "sliding window", where only the most recent m symbols are taken into consideration, where m is some fixed function of n. It would be interesting to analyze the performance guarantee of these approaches.

Another question is to design universal algorithms with better performance guarantees. In particular, a key question is whether we can design a universal caching algorithm that converges asymptotically to the optimal online algorithm.

References

[1] A. Aho, P. Denning, and J.D. Ullman. Principles of Optimal Page Replacement, *JACM*, **18**(1), 1971, 80-93.

[2] P. Algoet, Universal Schemes for Prediction, Gambling and Portfolio Selection, *Annals of Probability*, **20**(2), 1992, 901-941.

[3] T.M. Cover and J.A. Thomas, *Elements of Information Theory*, Wiley, New York, 1991.

[4] K. Curewitz, P. Krishnan and J.S. Vitter. Practical Prefetching Via Data Compression, In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1993, 257-266.

[5] R. El Yaniv and A. Borodin. *Online Computation and Competitive Analysis*, Cambridge University Press, 1998.

[6] M. Feder, N. Merhav and M. Gutman, Universal Prediction of Individual Sequences, *IEEE Transactions on Information Theory*, **38**, 1992, 1258-1270.

[7] A. Fiat, R.M. Karp, M. Luby, L. A. McGeoch, D.D. Sleator and N.E. Young, On Competitive Algorithms for Paging Problems, *Journal of Algorithms*, **12**, 1991, 685-699.

[8] P.A. Franaszek and T.J. Wagner. Some Distribution-free Aspects of Paging Performance, *Journal of the ACM*, **21**, 1974, 31-39.

[9] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, 1997.

[10] J.F. Hannan. Approximation to Bayes risk in repeated plays, in *Contributions to the Theory of Games, Vol. 3, Annals of Mathematics Studies*, Princeton, NJ, 1957, 97-139.

[11] P. Jacquet, W. Szpankowski, and I. Apostol. A Universal Predictor Based on Pattern Matching, *IEEE Transaction on Information Theory*, **48**(6), 2002, 1462-1472.

[12] A.R. Karlin, S.J. Phillips and P. Raghavan. Markov Paging, *SIAM Journal on Computing*, **30**(3), 906-922, 2000.

[13] C. Lund, S. Phillips, and N. Reingold. Paging against a Distribution and IP Networking, *Journal of Computer and System Sciences*, **58**, 1999, 222-231.

[14] N. Merhav and M. Feder. Universal Prediction, *IEEE Trans. Information Theory*, 44, 2124-2147, 1998.

[15] N. Merhav, E. Ordentlich, G. Seroussi, and M. J. Weinberger. On Sequential Strategies for Loss Functions With Memory, *IEEE Transactions on Information Theory*, **48**(7), 1947-1958, 2002.

[16] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*, Cambridge University Press, 2005.

[17] R. Motwani and P. Raghavan. *Randomized Algorithms*, Cambridge University Press, 1995.

[18] G. Pandurangan and E. Upfal. Entropy-Based Bounds for Online Algorithms, *ACM Transactions on Algorithms*, **3**(1), 2007.

[19] D.D. Sleator and R.E. Tarjan. Amortized Efficiency of List Update and Paging Rules, *Communications of the ACM*, **28**(2), 1985, 202-208.

[20] W. Szpankowski. *Average Case Analysis of Algorithms on Sequences*, John Wiley, 2001.

[21] W. Szpankowski. A Generalized Suffix Tree and Its (Un)Expected Asymptotic Behaviors, *SIAM J. Computing*, 22, 1176–1198, 1993.

[22] J.S. Vitter and P. Krishnan. Optimal Prefetching Via Data Compression, *Journal of the ACM*, **43**(5), 1996, 771-793.

[23] M. Weinberger and E. Ordentlich. On-line decision making for a class of loss functions via Lempel-Ziv Parsing, In *Proc. of the IEEE Data Compression Conference*, 2000, 163-172.

[24] J. Ziv and A. Lempel. A Universal Algorithms for Sequential Data Compression, *IEEE Transactions on Information Theory*, **23**(3), 1977, 337-343.

[25] J. Ziv and A. Lempel, Compression of Individual Sequences via Variable Rate Coding, *IEEE Transactions on Information Theory*, **24**(5), 1978, 530-536.