A Universal Online Caching Algorithm Based on Pattern Matching

Gopal Pandurangan Wojciech Szpankowski
Department of Computer Science
Purdue University
West Lafayette, IN 47907, USA
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).**

I. Introduction

We present a universal algorithm for the classical online problem of *caching* or *(demand) paging* [3]. In our setting the page request sequence is drawn from an *unknown* probability distribution (i.e., mixing source) and our 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. For example, Karlin, Phillips, and Raghavan [10] present efficient algorithms (that run in time polynomial in the cache size) when the request sequence is generated by a Markov source under the assumption that the online algorithm has complete knowledge of the Markov chain. The authors of [10] assume that the Markov chain can be "learned" from looking at a very long input. (Similarly, Franaszek and Wagner [6] 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 don't have knowledge of the class (cf. [12], [13]).

This paper is motivated by the work of Lund et al. [11] on caching strategy and universal prediction based on pattern matching due to Jacquet et al. [9]. The authors of [11] propose an efficient randomized 4-competitive online caching algorithm that works for *any* distribution D but it needs to know for (each pair of) pages p and q the probability that p will next be requested before q. 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). In this paper, we present a *universal* and simple algorithm based on pattern matching for mixing sources (including 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). We should point out that our novel approach can be also used for other caching strategies.

Caching is related to *prefetching* problem, and the latter is essentially the same as the *prediction problem*, studied extensively in information theory. In both problems, we have a collection $\mathcal{H}$ of pages in memory and a cache of size k (typically $k \ll |\mathcal{H}|$). Given a page request (from $\mathcal{H}$), 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 and pages are fetched only *on demand*.

Both these problems can be formulated as online decision problems (e.g., [2], [12], [18]) 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, $1 \le t \le n$, 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 [13]).

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 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* (e.g.,[8], [12]) (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* (e.g., [4]).

In the probabilistic setting, universal algorithms have been well-studied for prefetching and prediction problems. For example, Vitter and Krishnan [17] considered a model where the sequence of page requests is assumed to be generated by a *Markov source*. They show that the fault rate of a Ziv-Lempel based ([19]) 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. [4] 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., [12] 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. [13] on sequential strategies for loss function with memory that we discuss in some detail below.

In [13] 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. Per Weinberger [private correspondence, April 2004] this limited memory loss function approach can be set up in terms of the caching problem we discuss here as follows: Before each observation x_i, one takes an action b_i, consisting of adding a page to the cache and evicting either page $1, 2, \ldots, k$ or not to evict any page (call it: evict page 0). The cost at time i is infinity if the added page is not x_{i-1} or if x_{i-1} is not in cache and nothing is evicted. It is 1 if x_{i-1} is not in cache and it is added evicting someone else, and is 0 if no eviction was needed. Notice that it is allowed not to evict even when there is a page-fault, but then the infinite loss takes care of that. Thus, a loss is a function of all previous requests and all previous actions. This is defined as a loss function with memory in [13] where it is proved that there exists a universal algorithm for individual sequence with respect to an expert set (e.g., FSMs).

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 [14]. For online caching (or demand paging) the well known LRU (Least Recently Used) has a competitive ratio of k [14], where k is the cache size, while the randomized MARKER algorithm is $O(\log k)$ competitive [5]. 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 following the work of Lund et al. [11], 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], [6]) and for Markov sources (l-order Markov) [3]. 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) [10]. (In particular, many "natural" online strategies[1] such as LAST[2], MAX-REACH-TIME[3] perform poorly even on Markov chains [10].) 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 ([10], [11]); however, these results, as mentioned earlier, assume full knowledge of the Markov source, and hence not universal. In this paper, we propose a universal caching algorithm for mixing sources (this includes Markov sources) 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*,

[1]On the other hand, for prefetching, the optimal universal strategies (e.g., see Algoet [2]) are somewhat more "natural" and intuitive.

[2]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.

[3]On a fault for page r, evict that page whose expected time to be reached from r is maximum.

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.

II. Algorithm

To motivate our algorithm we first briefly describe the DOMinating-distribution (DOM) algorithm of Lund et al. ([11], [3]). DOM *assumes* that for a given distribution D, one can compute (efficiently) for all distinct pages q and r the probability $p(q, r)$ that r will be requested before q (such a distribution is called *pairwise-predictive*). A pairwise-predictive distribution 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 $p : S \times S \rightarrow [0, 1]$ satisfying the property that $p(q, r) + p(r, q) = 1$ for all $r \neq q$ in S and $p(r, r) = 0$ for all $r \in S$. Given a pairwise predictive distribution D and paging algorithm ALG, the weight function p is determined by D (e.g., 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 $\tilde{p} : S \rightarrow [0, 1]$ such that for every $q \in S$, if $u \in S$ is chosen with probability $\tilde{p}(u)$ then $E[p(q, u)] \leq 1/2$ (this expectation is taken with respect to both D and $\tilde{p}$). Lund et al. show the following key lemma on dominating distributions:

Lemma 1 ([11]): Every weighted tournament $T(S, w)$ has a weighted distribution and such a distribution can be found by solving a linear program consisting of S variables.

Now we can state the DOM algorithm: Let D be a pairwise-predictive input distribution and let $x = x_1, x_2, \ldots$ be a request sequence. On the tth request x_t, if x_t is a page fault (otherwise do nothing), then (as determined by D) construct a weighted tournament $T_t(S, p)$ on the k pages presently in the cache. Evict page q with probability $\tilde{p_t}(q)$ where $\tilde{p_t}$ is the dominating distribution for the tournament $T_t(S, p)$.

The following theorem gives the performance of DOM.

Theorem 1 ([11]): For all request sequences x from D, the following holds $E[DOM(x)] \leq 4 \cdot ON(x)$, and the complexity per page fault is bounded by k *assuming* that for all distinct pairs of pages q and r, we have precomputed $p(q, r)$.

We now propose a new universal caching algorithm that uses the idea of Sampled Pattern Matching (SPM) [9] to obtain a good estimate of the probability that page p occurs before page q (Steps 1-3 below). We then apply the caching strategy of [11] 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.
If x_n is not in the cache C and C is full do:

1. Find the largest suffix of x_1^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 $L_n \geq 2$ 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 a will occur before b *after* the marker position as follows: Let $Y_j(a, b)$ be the indicator r.v. for the event that a occurs before b in the substring that starts after the jth marker, $1 \leq j \leq L_n$. Then the estimator is

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

4. Compute a distribution $p(x)$ (call this a *dominating distribution* parameterized by x) by solving a LP as in Equation 2 (Section III) such that for each page a in C, if b is chosen according to p, then $E[\tilde{P}(a, b)] \leq x$, for some $x \in [0, 1]$ (whose value will be determined in the proof). Choose a page to evict from C according to the distribution $p(x)$.

The algorithm can be naturally implemented by maintaining a suffix tree [7]. 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 $\mathcal{H}$. 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. [7]), but on average will take only $O(n^{1-\alpha})$ (for some $\alpha > 1/2$) since there are only so many markers whp ([9]) and the delay is $O(\log^2 n)$ whp (cf. Section III). 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 $O(n^{1-\alpha})$ [9]. Step 4 can be implemented by solving a LP (cf. Theorem 2) and hence is polynomial in the size of the cache.

From the algorithmic complexity point of view, the above algorithm has the drawback of constructing a suffix tree after each eviction. In order to rectify it, one can propose the *fixed database caching algorithm* [15] in which we are given a database (training sequence) that is utilized to compute the estimator $\tilde{P}(a,b)$. 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. This approach will be elaborated in the full version of the paper.

III. Main Results 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. [15]).

Definition 1 (MX - (Strongly) ϕ-Mixing Source): Let $\mathcal{F}_m^n$ be a σ-field generated by $X_n^m = X_m X_{m+1} \ldots X_n$ for $m \leq n$. The source is called *mixing*, if there exists a bounded function $\phi(g)$ such that for all $m, g \geq 1$ and any two events $A \in \mathcal{F}_1^m$ and $B \in \mathcal{F}_{m+g}^\infty$ the following holds: $(1-\phi(g))\Pr(A)\Pr(B) \leq \Pr(AB) \leq (1+\phi(g))\Pr(A)\Pr(B)$. 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 $\gamma < 1$) [15]. For our analysis below, we assume that the ϕ mixing coefficient satisfies $\lim_{n\to\infty} n^{1-\alpha}\phi(n^\epsilon) = 0$ for any arbitrary small $\epsilon > 0$. Our main result is formulated next.

Theorem 2: Let A_n and OPT_n denote the number of page faults incurred by our algorithm and the optimal online algorithm respectively after n requests from a strongly mixing source. Then

$$E[A_n] \leq (4 + o(1))E[OPT_n]$$

as $n \to \infty$.

We now prove Theorem 2. We need the following results from [9]:
1. *Marker separation property*: There exists $\epsilon > 0$ such that for $\alpha > 1/2$ with high probability (whp)[4] as $n \to \infty$ two consecutive markers 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 is $n^{1-\alpha}$ whp.
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., $L = max_{1\leq j\leq L_n} L_j$ where L_j the delay before we all symbols after the jth marker.

Lemma 2: $L = O(\log^2 n)$ whp.

Proof. Let X_k be the first symbol after the end of a marker and consider the next $c\log^2 n$ symbols starting from X_k, i.e., the subsequence $X_k^{k+c\log^2 n}$ 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 $(1+\phi(\log n)))^{c\log n}(p_a)^{\log n} \leq 1/n^2$, for a suitably large constant c, where p_a is the (unconditional) probability of occurrence 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, we have that all symbols in C occur in $X_k^{k+c\log^2 n}$ with probability $1 - 1/n$ for some suitably large constant c. Thus the delay for this marker is $O(\log^2 n)$ with probability at least $1 - 1/n$. We appeal to the marker separation property and the fact that are at most $n^{1-\alpha}$ markers whp to conclude that whp that the delay is $O(\log^2 n)$ after every marker. ■

[4]i.e., with probability at least $1 - 1/n^\nu$ for some constant $\nu > 0$.

In the next lemma we prove that our estimator $\tilde{P}(a,b)$ is consistent.

Lemma 3: 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.

Proof sketch: The main idea of the proof is to show that that the estimator random variables $Y_j(a,b)$, $1 \leq j \leq L_n$ (computed in Step 3) are almost independent. We need the concept of a *favorite* string. Fix $\epsilon > 0$. Let i_j be the position after the last symbol of marker j, $1 \leq j \leq L_n$. Define a favorite string as 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 $O(\log^2 n)$. The marker separation properties and Lemma 2 imply that whp any string is a favorite. Define the set of favorite strings: $F_n = \{X_1^n : X_1^n \text{ is a favorite string}\}$. Consider the *delay subsequence*: $X_{i_j}^{i_j+c\log^2 n}$, $1 \leq j \leq L_n$, (c is a suitably large constant), i.e., the subsequence consisting of $O(\log^2 n)$ symbols after every marker. Using Lemma 2 and the two marker properties (which guarantees that whp that the markers are stable and the delays after are separated by n^ϵ for some $\epsilon > 0$) we show that the delay subsequence is mixing if $X_1^n \in F_n$ (cf. Lemma 7 in [9]). This implies that for favorite strings, the probability distribution of the estimator Y_j's are within factor of $(1 \pm O(\phi(n^\epsilon)))^{L_n}$ from an i.i.d. sequence. Let $P(a,b)$ be the true estimate. Thus, using a Chernoff bound for some $\nu \in (0,1)$

$$\Pr\left(|\tilde{P}(a,b) - P(a,b)| \leq n^{-\theta}\right)$$

$$\leq (1+O(\phi(n^\epsilon)))e^{-\frac{1}{3}L_n P(a,b) n^{-2\theta}} + \Pr(X_1^n \notin F_n) = O(1/n^\nu).$$

■

Finally, we are ready to prove our main result.

Proof of Theorem 2. We first show that the dominating distribution $p(x)$ can be chosen in Step 4 such that $x \leq 1/2 + 1/n^\theta$ whp. We use the approach in [11]. Consider the

following LP:
minimize x subject to

$$\begin{aligned} \sum_{a\in C} \tilde{P}(b,a)p(u) &\leq x \quad (\forall b \in C), \qquad (2)\\ \sum_{a\in C} p(a) &= 1, \quad p(a) \geq 0, \quad (\forall a \in C) \end{aligned}$$

For the purpose of analysis, appealing to Lemma 3, we rewrite the first constraint as:

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

where $P(b,a)$ is the true estimate and θ is a suitably small positive constant. By considering the dual LP, we can show that the solution of the LP is at most $1/2 + O(1/n^{\theta})$. By Lemma 3, 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 p be a random variable denoting the page that is evicted. Now the following property holds: for every page q in C, the probability that q is next requested no later than p is at least $1/2 - O(1/n^{\theta}) - O(1/n^{\nu})$. By Lemma 2.5 in [11], we conclude that the expected number of page faults is at most $4 + o(1)$ times the optimal online algorithm as $n \to \infty$. ■

IV. Acknowledgment

We thank Dr. M. Weinberger for a useful discussion regarding sequential strategies with memory loss function.

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

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

[5] 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.

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

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

[8] 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.

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

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

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

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

[13] 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.

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

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

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

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

[18] 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.

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