OPTIMAL PREDICTION FOR PREFETCHING
IN THE WORST CASE[*]

P. KRISHNAN[†] AND JEFFREY SCOTT VITTER[‡]

Abstract. Response time delays caused by I/O are a major problem in many systems and database applications. Prefetching and cache replacement methods are attracting renewed attention because of their success in avoiding costly I/Os. Prefetching can be looked upon as a type of online sequential prediction, where the predictions must be accurate as well as made in a computationally efficient way. Unlike other online problems, prefetching cannot admit a competitive analysis, since the optimal offline prefetcher incurs no cost when it knows the future page requests. Previous analytical work on prefetching [*J. Assoc. Comput. Mach.*, 143 (1996), pp. 771–793] consisted of modeling the user as a probabilistic Markov source.

In this paper, we look at the much stronger form of worst-case analysis and derive a randomized algorithm for pure prefetching. We compare our algorithm for every page request sequence with the important class of finite state prefetchers, making no assumptions as to how the sequence of page requests is generated. We prove analytically that the fault rate of our online prefetching algorithm converges almost surely for every page request sequence to the fault rate of the optimal finite state prefetcher for the sequence. This analysis model can be looked upon as a generalization of the competitive framework, in that it compares an online algorithm in a worst-case manner over all sequences with a powerful yet nonclairvoyant opponent. We simultaneously achieve the computational goal of implementing our prefetcher in optimal constant expected time per prefetched page using the optimal dynamic discrete random variate generator of Matias, Vitter, and Ni [*Proc. 4th Annual SIAM/ACM Symposium on Discrete Algorithms*, Austin, TX, January 1993].

Key words. caching, prefetching, competitive analysis, finite state prefetchers, response time, fault rate, hypertext, operating systems, databases, prediction, machine learning

AMS subject classifications. 68Q25, 68T05, 68P20, 68N25, 60J20

PII. S0097539794261817

1. Introduction. Most computer memories are organized hierarchically. A typical two-level memory consists of a relatively small but fast *cache* (such as internal memory) and a relatively large but slow memory (such as disk storage). Two-level memories can also model on-chip versus off-chip memory in VLSI systems. The pages requested by an application must be in cache before computation can proceed. In the event that a requested page is not in cache, a *page fault* occurs and the application has to wait while the page is fetched from slow memory to cache. The method of fetching pages into cache only when a fault occurs is called *demand fetching*. The problem of *cache replacement* is to decide which pages to remove from cache to accommodate the incoming pages.

In many systems and database applications, users spend a significant amount of

time processing a page, and the computer and I/O system are typically idle during that period. If the computer can predict which page the user will request next, it can fetch that page into cache (if it is not already in cache) *before* the user asks for it. When the user requests the page, it is available in cache, and the user perceives a faster response time. This method of getting pages into cache in the background before they are requested is called *prefetching*.

In many hypertext and interactive database systems, there is often sufficient time between user requests to prefetch as many pages as wanted, limited only by the cache size k. We refer to prefetching under this assumption as *pure prefetching*, and we restrict our analysis to pure prefetching in this paper. Pure prefetching is an important theoretical and practical model that helps in analyzing the benefits of fetching pages in the background.

In general applications, other issues come into play. For example, prefetches are often done well in advance of when the page is expected to be needed, to take into account latency [29, 31]. User requests can also preempt prefetch requests, resulting in fewer than k prefetches being done at a time. In such situations, which we call *nonpure prefetching*, issues of cache replacement come into play; the algorithm must determine not only which page(s) to prefetch but also which page(s) to evict from cache to make room. Pure prefetchers can be converted into efficient and practical nonpure prefetchers by melding them with good cache replacement strategies. In [10], pure prefetchers are used with the popular least recently used (LRU) cache replacement strategy, and significant reductions in page fault rate (number of page faults divided by the number of page requests) are demonstrated. We expect that better pure prefetchers (e.g., the one developed in this paper) melded with better cache replacement strategies (e.g., [5, 13, 19, 20]) may yield even more impressive performance improvements.

An algorithm is *online* if it must make its decisions based only on the past history. An *offline* algorithm can use knowledge of the future. If the program generating page requests is known a priori, prefetching decisions could be made offline, as is done in compiler-directed prefetching [6, 29, 31] where prefetch instructions are explicitly inserted into the code. Without a priori knowledge or statistics of the user request pattern, as is the case in many hypertext and interactive applications (e.g., the world wide web), an algorithm for cache replacement or prefetching must be online. An important computational requirement of online prefetching (and online demand fetching) algorithms is that the time spent deciding which pages to fetch into (or evict from) cache must be minimal. In this paper, we study online pure prefetching.

1.1. Analysis technique. The notion of *competitiveness* introduced by Sleator and Tarjan [34] evaluates an online algorithm by comparing its performance with that of offline algorithms. Competitive algorithms for cache replacement are well examined in the literature [5, 13, 27, 34]. It is unreasonable to expect prefetching algorithms to be competitive in this sense. The trivial optimal offline algorithm for prefetching never faults, if it can prefetch at least one page every time. In order to be competitive, an online algorithm would have to be an almost perfect predictor for any sequence, which seems intuitively impossible. Some restrictions on the power of the offline algorithm are therefore needed for a meaningful analysis.

Vitter and Krishnan [36] analyzed pure prefetching using a form of the competitive philosophy; they assumed that the sequence of page requests was generated by a probabilistic Markov source [14]. They showed that the prediction techniques inherent in data compression methods (such as the Lempel–Ziv algorithm [37]) can be

used to get optimal pure prefetchers. Cache replacement has been studied by Karlin, Phillips, and Raghavan [20] under a different stochastic version of the competitive framework; the sequence of page requests was assumed to be generated by a Markov chain (a subset of Markov sources). In [20, 36], the online prefetching or cache replacement algorithm is compared with the optimal online algorithm that has full prior knowledge of the source. A PAC learning framework incorporating Markov sources of examples was developed in [1]. Recent empirical work on prefetching includes a pattern matching approach to prediction [30], computing various first-order statistics for prediction [32], a growing-order Markov predictor [24], prefetching in a parallel environment [22], and research projects at a lower level of abstraction including compiler-directed prefetching [6, 29, 31].

In this paper, we develop a randomized algorithm for pure prefetching and show its optimality in the limit under the following analysis strategy. Putting *no* restrictions on the generator of page requests, we compare the page fault rate of our prefetcher *for every sequence of page requests* with that of the *best finite state prefetcher for the sequence.* We also show how to implement each prefetch in *constant expected time,* independent of the number of pages in the database and the cache size, which is optimal.

The analysis strategy used in this paper is much stronger than the ones in [20, 36] for the following reasons. First, our comparison is for all sequences, without any assumption about how the sequences are generated. Second, although the comparison in [20, 36] is against the optimal computationally unlimited online algorithm with full a priori knowledge of the source, it is the case for prefetching and cache replacement that when the page request sequences are generated by a finite state Markov source (or a Markov chain), the optimal online algorithm is finite state. (See [36, Definition 4] and [20, Theorem 2].) In particular, the fault rate of the prefetcher we develop in this paper is asymptotically the same as the fault rate of the optimal prefetcher from [36] when the source is finite state Markov.

Pure prefetching can be looked upon as the following prediction problem: given an arbitrary alphabet of size α (the set of α pages in the database) and a sequence of (page) requests drawn from this alphabet, at each time instant we have to predict the best k choices for the k pages to prefetch into cache. Randomness is required in order for a predictor or prefetcher to be optimal for every sequence when compared with finite state prefetchers (FSPs) [7]. (Also see Appendix A.) In the fields of information theory and statistics [4, 8, 12, 17] interesting algorithms for binary sequences (corresponding to an alphabet size of $\alpha = 2$ pages) that make one prediction for the next page (corresponding to cache size $k = 1$) have been developed independently of [36], and the comparison is made with the best finite state predictor. However, the $\alpha = 2$, $k = 1$ case is clearly unsuitable for our prefetching scenario. The procedure in [17] may possibly be generalizable to the arbitrary alphabet case $\alpha \geq 2$ for cache size $k = 1$, but it cannot possibly make a prediction in constant time independent of α, and the $k > 1$ case is open. In [28], predictors are developed for various continuous loss functions, but they are not relevant to the harder-to-analyze discontinuous 0–1 loss functions associated with cache replacement and prefetching. The solution to the general case thus requires a fundamentally different approach from those mentioned above.

1.2. This paper. Our major contribution in this paper is a randomized algorithm for pure prefetching *that achieves the optimal fault rate almost surely in the limit against the class of FSPs and that is simultaneously optimal in terms of running*

time for the general case of $\alpha \geq 2$ pages and cache size $k \geq 1$. ("Almost surely" means that the probability that convergence does not occur for an arbitrary sequence converges to 0 as the sequence length n gets larger.)

Our analysis model and main results are summarized in the next section. In section 3, we present our core prefetching algorithm P_1, which makes use of sampling without replacement, and we analyze it in section 4 by comparing it with the best one-state prefetcher. In section 5 we draw on ideas from information theory [9] applied to predicting [8, 12, 28] and generalize P_1 to get a universal prefetcher P that is optimal in the limit against a general FSP. The resulting optimal prefetcher P is a blend of P_1 and the prefetcher [36] based on the Lempel–Ziv data compressor [18, 25, 37]. We show in section 6 how to implement the prefetcher in constant expected time per prefetched page, independent of alphabet size α and cache size k, by using the optimal dynamic algorithm for generating discrete random variates of Matias, Vitter, and Ni [26], which uses a table lookup method of Hagerup, Mehlhorn, and Munro [16]. Other issues are discussed in section 7.

2. Analysis model and main results. We denote the cache size by k and the total number of different pages (or alphabet size) by α. We use the notation σ_i^j to denote the subsequence of a (possibly infinite) sequence σ starting at the ith page request up to and including the jth page request; in particular, σ_1^n denotes the first n page requests of σ. Given a parsing of σ_1^n into subsequences, we will denote the jth subsequence by σ_j.

DEFINITION 1. *An FSP is represented as a quintuple* (S, A, g, D, z_0), *where S is a finite set of states,* $A = \{0, 1, 2, \ldots, \alpha-1\}$ *is a finite alphabet of cardinality* $|A| = \alpha$, *g is a deterministic "next state" function that maps $S \times A$ into S, D is a (possibly randomized) decision strategy function that maps S into a k-tuple A^k, and $z_0 \in S$ is the start state. The FSP prefetches at state $z \in S$ the k pages specified by $D(z)$, and upon seeing the next page request i, it changes state from z to $g(z, i)$. We denote the set of all FSPs with at most s states by* $\mathcal{F}(s)$.

We next define the best fault rate achieved on a sequence by the class of FSPs.

DEFINITION 2. *Given an FSP F and a sequence σ_1^n, we denote by* $Fault_F(\sigma_1^n)$ *the fault rate of F on σ_1^n, that is, the number of page faults of F on σ_1^n (expected number of page faults if F has a randomized decision strategy), divided by the length n of the sequence. We define* $Fault_{\mathcal{F}(s)}(\sigma_1^n)$ *to be* $\inf_{F \in \mathcal{F}(s)} Fault_F(\sigma_1^n)$. *With a little abuse of notation we also denote by* $Fault_B(\sigma_1^n)$ *the fault rate of a nonfinite state prefetcher B.*

Intuitively, we can think of $Fault_{\mathcal{F}(s)}(\sigma_1^n)$ as being given by an optimal *offline* algorithm restricted by the finite state requirement. This means that unlike an offline algorithm, the FSP does not know the sequence σ_1^n beforehand. However, it knows how many times each of its transitions will be traversed when it is used to prefetch on the sequence σ_1^n. In other words, the optimal FSP is a "weak" offline algorithm. For example, the optimal one-state FSP for a sequence σ_1^n does not know σ_1^n but knows how many times each page appears in σ_1^n. By simple convexity arguments it can be verified that the optimal one-state FSP for σ_1^n will, when at state z, deterministically prefetch the k pages corresponding to the k transitions out of z that are traversed the maximum number of times. (Hence $Fault_{\mathcal{F}(s)}(\sigma_1^n) = \min_{F \in \mathcal{F}(s)} Fault_F(\sigma_1^n)$, the minimum fault rate achieved by any FSP with at most s states on σ_1^n.) For example, $Fault_{\mathcal{F}(1)}(\sigma_1^n)$ is attained by the following one-state (zero-order) prefetcher F_1: count the number of times that page i, for $0 \leq i \leq \alpha - 1$, appears in σ_1^n. Let $C_1(\sigma_1^n) = \{i_1, i_2, \ldots, i_k\}$ be k pages with the maximum k counts. At every time t, for $1 \leq t \leq n$,

predict the next page to be one of the k pages in $C_1(\sigma_1^n)$ (that is, we always keep the same pages in cache). We develop an online randomized prefetcher P_1 that achieves on average the best single-state (zero-order) prefetching fault rate $Fault_{\mathcal{F}(1)}(\sigma_1^n)$ on every sequence σ_1^n of length n in the limit as $n \to \infty$.

THEOREM 1. *For every sequence σ_1^n of length n drawn from alphabet A, the fault rate of prefetcher P_1 on σ_1^n converges almost surely to $Fault_{\mathcal{F}(1)}(\sigma_1^n)$ as $n \to \infty$. In particular,*

$$(1) \qquad Fault_{P_1}(\sigma_1^n) \le Fault_{\mathcal{F}(1)}(\sigma_1^n) + O\left(\frac{\log n}{\sqrt{n}}\right).$$

The main difficulty in developing P_1 and its proof of optimality is that the alphabet size α and the cache size k are arbitrary. We note that even for the $\alpha = 2$, $k = 1$ case, the convergence rate cannot be faster than $O(1/\sqrt{n})$ [7].

The importance of the above theorem lies in its generalization to higher order using techniques from information theory [9]. The approach of [12] allows us to combine P_1 with a prefetcher [36] based on the Lempel–Ziv data compressor [18, 25, 37] to get a prefetcher P that is optimal in the limit against the class of FSPs.

THEOREM 2. *For every sequence σ_1^n of length n drawn from alphabet A, and any $s \ge 0$, the fault rate of prefetcher P on σ_1^n converges almost surely to $Fault_{\mathcal{F}(s)}(\sigma_1^n)$ as $n \to \infty$.*

From the observation in section 1.1 that under the model from [36] (where the sequences of page requests are generated by a finite state Markov source), the optimal prefetcher is also an FSP, we get the following corollary.

COROLLARY 1. *Under the model from [36], where the sequences of page requests are generated by a finite state Markov source M, the fault rate of prefetcher P converges almost surely to the minimum fault rate of any online prefetcher with complete a priori knowledge of the source M.*

The expected running time for prefetcher P can be made optimal by using the optimal dynamic random variate generator of [26].

THEOREM 3. *The prefetcher P runs in constant expected time (independent of α and k) for each page prefetched; that is, it requires an average of $O(k)$ time to determine which k pages to prefetch.*

The rate of convergence of Theorems 1 and 2 depends on the alphabet size α. For example, the error term is $O(\alpha k^2(\log n)/\sqrt{n})$ in Theorem 1; for simplicity we suppress the αk^2 term in our discussion since it is insignificant w.r.t. n in the limit. (Note that in general $k \ll \alpha$.) However, the constant time bound for each prediction is entirely independent of α and k, which is important from a computational point of view.

3. The prefetching algorithm P_1. In this section we give the algorithm P_1 that matches the best one-state prefetcher in the limit. Before introducing P_1, we present the more intuitive algorithm P_1' upon which algorithm P_1 is based.

Let t be the current time and let σ_1^t be the sequence of t pages requested until now. Let $f_i(\sigma_1^t)$, $0 \le i \le \alpha - 1$, denote the number of times page i appears in σ_1^t. Define $r_t = 2^j$ when $4^{j-1} < t \le 4^j$. That is, the integer r_t is "close to" $\sqrt{t}$; it doubles at discrete time steps (when t is one greater than a power of 4).

The key idea that prefetcher P_1' (and prefetcher P_1) uses is to reduce the problem of prediction to the problem of generating random variates. Intuitively P_1' should choose for the cache the page i with the highest or nearly highest frequency

count $f_i(\sigma_1^t)$. (Randomness in the picking is required, by the remark in section 1 [7], so it does not suffice to simply pick the page with the highest frequency count; see Appendix A.1. Further, the "natural" randomized algorithm that prefetches page i at time t with probability proportional to $f_i(\sigma_1^t)$ is also not optimal as shown in Appendix A.2.) In P_1' we get the effect of choosing the pages with the highest or nearly highest counts by "boosting" the frequency counts of the page by a large power and then choosing a page with probability proportional to its boosted count. (The boosted counts will be very large but can be represented with $O(\log n)$ bits, using the scheme discussed in section 6.) Efficient random variate generation with dynamically changing weights can be done using [26], as discussed in section 6.

The algorithm P_1' is a simple randomized weighting algorithm that makes k predictions at each time step for the next page request. At each time t and $0 \le i \le \alpha - 1$, P_1' assigns to page i a probability $p_{i,t}$ proportional to the boosted frequency count

$$(2) \qquad\qquad\qquad \left(f_i(\sigma_1^t) \right)^{r_t},$$

which is the r_tth power of frequency $f_i(\sigma_1^t)$. It predicts k items for the next request by choosing without replacement from the distribution $p_{0,t}, p_{1,t}, \ldots, p_{\alpha-1,t}$.

Example 1. Let $\alpha = 3$, $k = 2$. If the sequence σ_1^t of $t = 9$ pages seen until now is 210011102, we have $f_0(\sigma_1^9) = 3$, $f_1(\sigma_1^9) = 4$, $f_2(\sigma_1^9) = 2$, and $r_t = 2^2 = 4$. Algorithm P_1' assigns probability $p_{0,9} = 3^4/(3^4 + 4^4 + 2^4) \approx 0.229$ to page 0, probability $p_{1,9} = 4^4/(3^4 + 4^4 + 2^4) \approx 0.725$ to page 1, and probability $p_{2,9} = 2^4/(3^4 + 4^4 + 2^4) \approx 0.045$ to page 2. It predicts two pages out of these three by choosing without replacement based on the above probability distribution. Pages 0 and 1 are the likely pages to be chosen. □

Algorithm P_1' can be shown to be optimal against the best one-state FSP for general $\alpha \ge 2$ but only when $k = 1$ [23]. We can modify algorithm P_1' to get algorithm P_1 that is optimal against the best one-state FSP for general $\alpha \ge 2$, $k \ge 1$.

DEFINITION 3. *We define subsequence $\sigma_0 = \sigma_1^2$, and $\sigma_j = \sigma_{4^{j-1}+2}^{4^j+1}$ for $j \ge 1$. We call σ_j the jth r-subsequence of σ_1^n.*

Notice from the definition of r_t that P_1' predicts each page in an r-subsequence using the same value for r_t in (2) when $t > 2$.

Algorithm P_1 works like algorithm P_1', except that the frequency counts for the pages are reset to 0 at the start of each r-subsequence. That is, at time $t > 1$, $4^{j-1} < t \le 4^j$, algorithm P_1 assigns to page i a probability $p_{i,t}$ proportional to

$$\left(f_i(\sigma_{4^{j-1}+2}^t) \right)^{r_t}.$$

Example 2. As in Example 1, let $\alpha = 3$, $k = 2$, and let the sequence of $t = 9$ pages σ_1^t seen until now be 210011102. We have $\sigma_0 = 21$, $\sigma_1 = 001$, and the portion of σ_2 seen until now is 1102. The counts of pages 0, 1, and 2 in the current r-subsequence are 1, 2, and 1, respectively, and $r_t = 2^2 = 4$. Algorithm P_1 assigns pages 0, 1, and 2 the probabilities $p_{0,9} = 1^4/(1^4 + 2^4 + 1^4) \approx 0.056$, $p_{1,9} = 2^4/(1^4 + 2^4 + 1^4) \approx 0.889$, and $p_{2,9} = 1^4/(1^4 + 2^4 + 1^4) \approx 0.056$, respectively. It predicts two pages out of these three by choosing without replacement based on the above probability distribution. □

This regular throwing away of past information by algorithm P_1 makes the proof of optimality more elegant. Algorithm P_1 may also perform better than algorithm P_1' in practice, since it captures the effect of locality of reference found in page request sequences [3, 5, 11, 19, 20, 33].

4. One-state case: Optimality of P_1 vs. $\mathcal{F}(1)$. In this section we prove an important special case of Theorem 1, namely, that the expected value of P_1's fault

rate $Fault_{P_1}(\sigma_1^n)$ converges to $Fault_{\mathcal{F}(1)}(\sigma_1^n)$; the almost-sure convergence follows by using the Borel–Cantelli lemma. As pointed out in section 2, given a sequence σ_1^n, the following prefetcher $F_1 \in \mathcal{F}(1)$ attains the minimum fault rate among all one-state prefetchers: count the number of times page i, for $0 \le i \le \alpha - 1$, appears in σ_1^n. Let $C_1(\sigma_1^n) = \{i_1, i_2, \ldots, i_k\}$ be the pages with the maximum k counts. For each time instant t, $1 \le t \le n$, F_1 prefetches the k pages in $C_1(\sigma_1^n)$. We have

$$(3) \qquad Fault_{\mathcal{F}(1)}(\sigma_1^n) = 1 - \frac{f_{i_1}(\sigma_1^n) + f_{i_2}(\sigma_1^n) + \cdots + f_{i_k,n}(\sigma_1^n)}{n}.$$

We now define a *balanced form* of a subsequence and an *approximately balanced form* of a sequence. These notions are useful in showing the optimality of P_1.

DEFINITION 4. *A subsequence $\widehat{\sigma}_a^b$ is a* balanced form *of σ_a^b if*
1. *$\widehat{\sigma}_a^b$ has the same composition of pages as σ_a^b, that is, for all $0 \le i \le \alpha - 1$, page i appears the same number of times in $\widehat{\sigma}_a^b$ as it does in σ_a^b;*
2. *for each $a \le t \le b$, page $\widehat{\sigma}_t^t$ occurs the maximal number of times in $\widehat{\sigma}_a^t$ in comparison with the other pages.*

For example, if $\sigma_1^{10} = 1111211321$, a balanced form is $\widehat{\sigma}_1^{10} = 123121211$.

DEFINITION 5. *A sequence $\widetilde{\sigma}_1^n$ is an* approximately *or* piecewise balanced *form of σ_1^n if*

$$\widetilde{\sigma}_1^n = \widehat{\sigma}_0 \widehat{\sigma}_1 \widehat{\sigma}_2 \ldots,$$

where $\widehat{\sigma}_j$ is a balanced form of σ_j, and σ_j is the jth r-subsequence of σ_1^n as defined in Definition 3.

By (3) and the first condition in Definition 4, we have $Fault_{\mathcal{F}(1)}(\sigma_1^n) = Fault_{\mathcal{F}(1)}(\widetilde{\sigma}_1^n)$. Our strategy to show optimality of P_1 (Theorem 1) is a two-step process described by the following two theorems. First, we show that the fault rate of P_1 on σ_1^n is never more than the fault rate of P_1 on the approximately balanced $\widetilde{\sigma}_1^n$.

THEOREM 4. *For every sequence σ_1^n, we have*

$$(4) \qquad Fault_{P_1}(\sigma_1^n) \le Fault_{P_1}(\widetilde{\sigma}_1^n).$$

We then compute the fault rate of P_1 on the approximately balanced $\widetilde{\sigma}_1^n$ and show that it is close to the fault rate of the best one-state machine for σ_1^n.

THEOREM 5. *For every sequence σ_1^n, we have*

$$(5) \qquad Fault_{P_1}(\widetilde{\sigma}_1^n) - Fault_{\mathcal{F}(1)}(\sigma_1^n) = O(\log n/\sqrt{n}).$$

The proofs of the above two theorems are dealt with in the next two subsections.

4.1. The approximately balanced sequence is sufficiently worst case. In this subsection we prove Theorem 4 using an interesting extension of the switch analysis of [12] in conjunction with the important notion of boosted frequency counts (2).

We denote the jth r-subsequence σ_j by π_1^η, where $\eta = 4^j - 4^{j-1}$ is the length of σ_j. The sequence π_1^η can be converted to a balanced form $\widehat{\pi}_1^\eta$ by an iterative balancing strategy. Without loss of generality, let the $(\tau + 2)$nd page request in π_1^η be 1, and let the $(\tau + 1)$st page request of the balanced sequence $\widehat{\pi}_1^{\tau+1}$ be 0. We denote by f_i the number of times page i appears in $\widehat{\pi}_1^\tau$. In particular, we denote the number of 0's in $\widehat{\pi}_1^\tau$ by f_0, and the number of 1's in $\widehat{\pi}_1^\tau$ by f_1. We consider the following iterative balancing strategy to convert $\widehat{\pi}_1^{\tau+1}$ to $\widehat{\pi}_1^{\tau+2}$.

Balancing strategy. Since $\widehat{\pi}_1^{\tau+1}$ is balanced, $f_1 \leq f_0 + 1$. If $f_1 \geq f_0$, then $\widehat{\pi}_1^{\tau+1}$ appended with a 1 gives $\widehat{\pi}_1^{\tau+2}$. If $f_1 < f_0$, we perform a "01" → "10" *switch at position* $(\tau + 1, \tau + 2)$ by moving the 1 in front of the 0. We continue this process of "bubbling" the 1 forward through $\widehat{\pi}_1^{\tau}$ by performing similar switches, until the subsequence of the first $\tau + 2$ page requests of π_1^{η} is balanced.

Our proof of Theorem 4 consists in showing that each switch in the balancing strategy does not lower the page fault rate of the entire sequence.

A similar but simpler idea worked in the binary case for a different algorithm [12], in which the sequence did not need to be broken up into subsequences, and the sequence $\widehat{\sigma}_1^n$ could be shown to be strictly worst case. We break σ_1^n into subsequences as part of our method for achieving optimal computational efficiency (as discussed in section 6).

We now show that a switch within an r-subsequence can only increase the fault rate for algorithm P_1. The fact that we allow $k \geq 1$ predictions before each page request makes the probability terms in the analysis conditional on the previous prefetches at that time step, and that complicates the analysis.

LEMMA 1. *Each switch involved in converting π_1^{η} to $\widehat{\pi}_1^{\eta}$ creates a subsequence on which P_1 has a larger fault rate (that is, switches within an r-subsequence increase the fault rate).*

Proof. Let us denote by A the probability of predicting the 0 in $\pi_1^{\tau} 01 \pi_{\tau+3}^{\eta}$ and by B the probability of predicting the 1 in $\pi_1^{\tau} 01 \pi_{\tau+3}^{\eta}$, where the 0 and 1 are in the same r-subsequence. Similarly, let us denote by C the probability of predicting the 1 in $\pi_1^{\tau} 10 \sigma_{\tau+3}^{\eta}$, and by D the probability of predicting the 0 in $\pi_1^{\tau} 10 \sigma_{\tau+3}^{\eta}$, where the 1 and 0 are in the same r-subsequence. The number of faults algorithm P_1 makes on the portions π_1^{τ} and $\pi_{\tau+3}^{\eta}$ will be the same before and after the switch, since the probability of fault by P_1 at position x of π_1^{η} depends only on the composition of pages in π_1^x. (Recall that P_1 throws away all previous counts for pages at the beginning of an r-subsequence.) To show that a switch increases the fault rate, we must show that the increase in the number of faults caused by moving the 0 to later in the sequence overshadows the decrease in the number of faults caused by moving the 1 to earlier in the sequence; that is, we need to show that $(1 - A) + (1 - B) \leq (1 - C) + (1 - D)$. This is equivalent to showing that

$$(6) \qquad\qquad\qquad A - D \geq C - B.$$

If P_1 makes only one prediction at each step, the proof of (6) is easy. (The proof follows directly from (7) for the special case $k = 1$.) However, P_1 makes $k \geq 1$ predictions at each time instant.

Let $A = A_1 + A_2 + \cdots + A_k$, where A_i is the probability of predicting the 0 in $\pi_1^{\tau} 01 \pi_{\tau+3}^{\eta}$ in the ith prediction. The probabilities B, C, D are similarly partitioned. Each A_i can be further broken up into a "good part" G_i^A and a "bad part" R_i^A. The good part G_i^A is the probability of the event that we predict the 0 in $\pi_1^{\tau} 01 \pi_{\tau+3}^{\eta}$ in the ith prediction and that none of the previous $i - 1$ predictions was a 1. The bad part R_i^A is the probability of the event that we predict the 0 in $\pi_1^{\tau} 01 \pi_{\tau+3}^{\eta}$ in the ith prediction and that a 1 was predicted in one of the previous $i-1$ predictions. Clearly, $A_i = G_i^A + R_i^A$, and $R_1^A = 0$. The quantities $G_i^B, R_i^B, G_i^C, R_i^C, G_i^D, R_i^D$ are defined similarly. We now show the following two facts:

Fact 1. $G_i^A - G_i^D \geq G_i^C - G_i^B$ for $1 \leq i \leq k$;

Fact 2. $(G_i^A - G_i^D) + (R_{i+1}^A - R_{i+1}^D) = 0$ and $(G_i^C - G_i^B) + (R_{i+1}^C - R_{i+1}^B) = 0$ for $1 \leq i \leq k - 1$.

Facts 1 and 2 say intuitively that the good parts of the ith prediction maintain the relationship that we want. The bad parts of the ith prediction *exactly* cancel the gain from the good parts of the $(i-1)$st prediction. The lemma follows from the above two facts as follows:

$$A - D = \sum_{1 \leq i \leq k} A_i - D_i = \sum_{1 \leq i \leq k} (G_i^A - G_i^D) + (R_i^A - R_i^D) = G_k^A - G_k^D$$

by repeated application of Fact 2. Similarly, $C - B = G_k^C - G_k^B$. By Fact 1, $G_k^A - G_k^D \geq G_k^C - G_k^B$, which implies $A - D \geq C - B$.

We now prove Facts 1 and 2 by induction. Since π_1^τ is an r-subsequence, at each time step, algorithm P_1 boosts frequencies by the same exponent r_t; for convenience, we denote this exponent by r. Let $d = \sum_0^{\alpha-1} (f_i)^r$, $d_1 = d - (f_1)^r + (f_1 + 1)^r$, and $d_0 = d - (f_0)^r + (f_0 + 1)^r$. These quantities are involved in the denominators of the rational expressions for the predictions. Let $u_i^A = u_i^A(x_1, \ldots, x_{i-1})$ be the term in G_i^A that corresponds to predicting $x_1, \ldots, x_{i-1}$, none of them a 0 or a 1 as the first $i-1$ predictions and 0 as the ith prediction. (The order of the first $i-1$ predictions is important. For example, when $i = 3$, the probability of predicting a 0 following x_1, x_2 is different from the probability of predicting a 0 following x_2, x_1.) The quantities u_i^B, u_i^C, u_i^D are defined similarly. The expressions in Fact 1 can be expressed in terms of the u's; for example,

$$G_i^A - G_i^D = \sum_{\substack{x_1, \ldots, x_{i-1} \\ \text{distinct}, \neq 0, 1}} u_i^A - u_i^D.$$

Let $v_i^A = v_i^A(x_1, x_2, \ldots, x_{i-2})$ be the term in R_i^A that corresponds to predicting a 0 in the ith prediction with a 1 as one of the first $i-1$ predictions and the other $i-2$ predictions being $x_1, \ldots, x_{i-2}$, none of them a 0 or a 1. (The order of these $i-2$ predictions is important, as it is with u_i, but the relative point at which the 1 is predicted is arbitrary. In effect, v_i^A is the sum of $i-1$ probability terms, where the $i-1$ terms correspond to the $i-1$ positions that the "1" is predicted.) The quantities v_i^B, v_i^C, v_i^D are defined similarly. The expressions in Fact 2 can be expressed in terms of the u's and the v's; for example,

$$(G_i^A - G_i^D) + (R_{i+1}^A - R_{i+1}^D) = \sum_{\substack{x_1, \ldots, x_{i-1} \\ \text{distinct}, \neq 0, 1}} (u_i^A - u_i^D) + (v_{i+1}^A - v_{i+1}^D).$$

Let $den(d, 0) = d$, and let $den(d, i-1) = d - (f_{x_1})^r - \cdots - (f_{x_{i-1}})^r$ for $i \geq 2$. Let $\pi den(d, i-1)$ for $i \geq 2$ be the $(i-1)$-term falling product $d \times (d - (f_{x_1})^r) \times \cdots \times (d - (f_{x_1})^r - \cdots - (f_{x_{i-2}})^r) = \Pi_{j=1}^{i-1} den(d, j-1)$.

Proof of Fact 1. It suffices to show by induction that $u_i^A - u_i^D \geq u_i^C - u_i^B$. For the base case when $i = 1$, $u_1^A - u_1^D = (f_0)^r/d - (f_0)^r/d_1$, and $u_1^C - u_1^D = (f_1)^r/d - (f_1)^r/d_0$. Hence

$$(7) \qquad \frac{u_1^A - u_1^D}{u_1^C - u_1^B} = \frac{(f_0)^r}{(f_1)^r} \times \frac{(f_1+1)^r - (f_1)^r}{(f_0+1)^r - (f_0)^r} \times \frac{d_0}{d_1} = \frac{(1+1/f_1)^r - 1}{(1+1/f_0)^r - 1} \times \frac{d_0}{d_1}.$$

Since the function $g(x) = x^r$ is convex and $f_0 \geq f_1$, the above quantity is at least 1. From the induction hypothesis that $u_{i-1}^A - u_{i-1}^D \geq u_{i-1}^C - u_{i-1}^B$ we get

$$(f_{x_1} f_{x_2} \cdots f_{x_{i-2}} f_0)^r \left(\frac{1}{\pi den(d, i-1)} - \frac{1}{\pi den(d_1, i-1)} \right)$$

$$(8) \qquad \geq (f_{x_1} f_{x_2} \cdots f_{x_{i-2}} f_1)^r \left(\frac{1}{\pi den(d, i-1)} - \frac{1}{\pi den(d_0, i-1)} \right).$$

As in (7), since $g(x) = x^r$ is convex, and $f_0 \geq f_1$, we have

$$\frac{(f_0)^r}{(f_1)^r} \times \frac{d_1 - d}{d_0 - d} \times \frac{\pi den(d_0, i-1)}{\pi den(d_1, i-1)} \times \frac{den(d_0, i-1)}{den(d_1, i-1)} \geq 0.$$

It follows that

$$\frac{(f_{x_1} f_{x_2} \cdots f_{x_{i-1}} f_0)^r}{\pi den(d_1, i-1)} \left(\frac{1}{den(d, i-1)} - \frac{1}{den(d_1, i-1)} \right)$$

$$(9) \qquad \geq \frac{(f_{x_1} f_{x_2} \cdots f_{x_{i-1}} f_1)^r}{\pi den(d_0, i-1)} \left(\frac{1}{den(d, i-1)} - \frac{1}{den(d_0, i-1)} \right).$$

Multiplying (8) by $(f_{x_{i-1}})^r / den(d, i-1)$ and adding to (9) gives us $u_i^A - u_i^D \geq u_i^C - u_i^B$.

Proof of Fact 2. To prove that $(G_i^A - G_i^D) + (R_{i+1}^A - R_{i+1}^D) = 0$, it suffices to show that $(u_i^A - u_i^D) + (v_{i+1}^A - v_{i+1}^D) = 0$. For the base case when $i = 1$, $u_1^A - u_1^D = (f_0)^r (1/d - 1/d_1)$, and

$$v_2^A - v_2^D = \frac{(f_1)^r (f_0)^r}{d(d - (f_1)^r)} - \frac{(f_1 + 1)^r (f_0)^r}{d_1(d_1 - (f_1 + 1)^r)}.$$

Using the fact that $d_1 = d + (f_1 + 1)^r - (f_1)^r$, it follows from simple algebra that $v_2^A - v_2^D + (u_1^A - u_1^D) = 0$.

For the inductive step, recall that v_{i+1}^A is a sum of i probability terms, where the i terms correspond to the i positions that the "1" is predicted. (The terms are each rational expressions with different denominators.) In particular, v_{i+1}^A equals

$$\frac{(f_{x_1} f_{x_2} \cdots f_{x_{i-1}} f_0 f_1)^r}{den(d, i-1) - (f_1)^r} \left(\frac{1}{\pi den(d, i)} + \frac{1}{den(d, i-2) - (f_1)^r} \left(\frac{1}{\pi den(d, i-1)} + \cdots \right) \right),$$

which can be simplified to

$$v_{i+1}^A = \frac{(f_{x_1} f_{x_2} \cdots f_0 f_1)^r}{den(d, i-1) - (f_1)^r} \left(\frac{1}{\pi den(d, i)} + \frac{v_i^A}{(f_{x_1} f_{x_2} \cdots f_{x_{i-2}} f_0 f_1)^r} \right).$$

Since $d_1 - (f_1 + 1)^r = d - (f_1)^r$, we find that $v_{i+1}^A - v_{i+1}^D$ equals

$$(10) \qquad \frac{(f_{x_1} f_{x_2} \cdots f_{x_{i-1}} f_0)^r}{den(d, i-1) - (f_1)^r} \left(\frac{(f_1)^r}{\pi den(d, i)} - \frac{(f_1 + 1)^r}{\pi den(d_1, i)} + \frac{v_i^A - v_i^D}{(f_{x_1} f_{x_2} \cdots f_{x_{i-2}} f_0)^r} \right).$$

By the induction hypothesis, $v_i^A - v_i^D = -(u_{i-1}^A - u_{i-1}^D)$. The value for $u_{i-1}^A - u_{i-1}^D$ is the expression on the left-hand side of (8). Substituting for $u_{i-1}^A - u_{i-1}^D$ in (10) we get

$$(11) \qquad v_{i+1}^A - v_{i+1}^D = \frac{(f_{x_1} f_{x_2} \cdots f_{x_{i-1}} f_0)^r}{d - (f_{x_1})^r - \cdots - (f_{x_{i-1}})^r - (f_1)^r} \times U,$$

where

$$U = \frac{(f_1)^r}{\pi den(d, i)} - \frac{(f_1 + 1)^r}{\pi den(d_1, i)} - \frac{1}{\pi den(d, i-1)} + \frac{1}{\pi den(d_1, i-1)}.$$

The quantity $u_i^A - u_i^D$ can be expressed as

$$u_i^A - u_i^D = (f_{x_1} f_{x_2} \cdots f_{x_{i-1}} f_0)^r \left(\frac{1}{\pi den(d, i)} - \frac{1}{\pi den(d_1, i)} \right).$$

Adding the above expression to (11) and simplifying we find that $(u_i^A - u_i^D) + (v_{i+1}^A - v_{i+1}^D) = 0$. A similar analysis shows that $(G_i^C - G_i^B) + (R_{i+1}^C - R_{i+1}^B) = 0$. □

4.2. $Fault_{P_1}(\widetilde{\sigma}_1^n)$ is close to $Fault_{\mathcal{F}(1)}(\sigma_1^n)$. In this subsection we prove Theorem 5. Let $F_1 \in \mathcal{F}(1)$ be the best one-state prefetcher for σ_1^n. Let $F_1^j \in \mathcal{F}(1)$ be the best one-state prefetcher tuned for the jth r-subsequence σ_j. Let $NumFaults_B(\sigma_a^b)$ be the number of faults incurred by algorithm B on subsequence σ_a^b.

It is clear by definition that prefetcher F_1^j incurs at most as many faults on σ_j as F_1 incurs on σ_j. In other words,

$$(12) \quad NumFaults_{F_1}(\widehat{\sigma}_j) = NumFaults_{F_1}(\sigma_j) \geq NumFaults_{F_1^j}(\sigma_j) = NumFaults_{F_1^j}(\widehat{\sigma}_j).$$

Equation (12) directly implies the following lemma.

LEMMA 2. *The fault rate incurred for σ_1^n by using prefetcher F_1^j to prefetch for the jth r-subsequence σ_j, for each $j \geq 0$, is no greater than $Fault_{\mathcal{F}(1)}(\sigma_1^n)$. That is,*

$$Fault_{\mathcal{F}(1)}(\sigma_1^n) = Fault_{\mathcal{F}(1)}(\widetilde{\sigma}_1^n) = \frac{NumFaults_{F_1}(\widehat{\sigma}_1^n)}{n} \geq \frac{\sum_{j \geq 0} NumFaults_{F_1^j}(\widehat{\sigma}_j)}{n}.$$

The above lemma is useful since it is easier to compare algorithm P_1 with algorithm F_1^j on page request sequence $\widehat{\sigma}_j$ than it is to compare P_1 with F_1.

LEMMA 3. *The number of faults that algorithm P_1 incurs on $\widehat{\sigma}_j$ is close to the number of faults of the optimal one-state machine tuned for $\widehat{\sigma}_j$. In particular,*

$$NumFaults_{P_1}(\widehat{\sigma}_j) - NumFaults_{F_i^j}(\widehat{\sigma}_j) = O((\alpha k^2) j 2^j).$$

From Lemmas 2 and 3 we get

$$Fault_{P_1}(\widetilde{\sigma}_1^n) - Fault_{\mathcal{F}(1)}(\widetilde{\sigma}_1^n) \leq \sum_j \frac{NumFaults_{P_1}(\widehat{\sigma}_j) - NumFaults_{F_1^j}(\widetilde{\sigma}_j)}{n}$$

$$= O\left(\frac{\alpha k^2 \sum_j j 2^j}{n} \right)$$

$$(13) \qquad\qquad = O\left(\frac{\alpha k^2 \log n}{\sqrt{n}} \right).$$

Theorem 5 follows from (13) and the observation following Definition 5.

We now give the proof of Lemma 3.

Proof of Lemma 3. For simplicity, we denote by π_1^η the balanced jth r-subsequence $\widehat{\sigma}_j$, where $\eta = 4^j - 4^{j-1}$ is the length of $\widehat{\sigma}_j$. Divide π_1^η into α subsequences $\pi_0, \pi_1, \ldots, \pi_{\alpha-1}$, where exactly $\alpha - i$ different pages appear in π_i. (Some of the π_i's may be empty. Notice that since π_1^η is balanced, each π_i has the same number of occurrences of each of the $\alpha - i$ pages in it.) We explicitly compute the difference between the expected number of faults of P_1 and the number of faults of F_1^j for each subsequence π_i. We need to be careful with the asymptotics involved, since the counts for the

pages are small in the earlier part of π_1^η, but $r_t = O(\sqrt{\eta})$ is relatively larger. For simplicity, we drop the subscript t from r_t in the following discussion; recall that r_t does not change in an r-subsequence.

Let $|\pi_i|$ be the length of subsequence π_i. Algorithm F_1^j incurs $|\pi_i| \times \max\{0, 1 - k/(\alpha - i)\}$ faults on π_i. Define L_i as

$$
L_i = \begin{cases} \dfrac{|\pi_0|}{\alpha} + \dfrac{|\pi_1|}{\alpha - 1} + \cdots + \dfrac{|\pi_{i-1}|}{\alpha - (i-1)} & \text{if } i \geq 1, \\ 0 & \text{if } i = 0. \end{cases}
$$

The quantity L_i tracks the number of times a page that appears in π_i has already appeared in $\pi_0, \pi_1, \ldots, \pi_{i-1}$. The expected number of faults incurred by algorithm P_1 on π_i is

$$
(14) \qquad NumFaults_{P_1}(\pi_i) = |\pi_i| - \sum_{u=1}^{|\pi_i|/(\alpha-i)} \sum_{v=0}^{\alpha-i-1} \sum_{k_1=1}^{k} \Pr(u, v, k_1),
$$

where $\Pr(u, v, k_1)$ is the probability of predicting the $((u-1) \times (\alpha - i) + v + 1)$st page request of π_i in the k_1th prediction. For notational simplicity, let $t = (u-1) \times (\alpha - i) + v + 1$. Clearly, the tth page has appeared $L_i + u - 1$ times in π_1^{t-1}, and no page in π_i has appeared more than $L_i + u$ times in π_1^{t-1}. Since we require an upper bound on $NumFaults_{P_1}(\pi_i)$, we determine a lower bound for $\Pr(u, v, k_1)$ as follows. We only consider $k_1 \leq \alpha - i$, and we only consider that subset of events when pages with count greater than or equal to that of the tth page (i.e., pages appearing in π_i) were predicted in the previous $k_1 - 1$ predictions. Hence

$$
\Pr(u, v, k_1) \geq (k_1 - 1)! \binom{\alpha - i - 1}{k_1 - 1} \prod_{w=0}^{k_1-1} \frac{(L_i + u - 1)^r}{(\alpha - i - w)(L_i + u)^r + \sum_{q=0}^{i-1}(L_{q+1})^r}.
$$

Notice that multiplying by $(k_1 - 1)!$ in the above equation is essential, since the order of the first $(k_1 - 1)$ predictions is significant. We get

$$
(15) \quad \Pr(u, v, k_1) \geq (\alpha - i - 1)^{\underline{k_1-1}} \prod_{w=0}^{k_1-1} \frac{(L_i + u - 1)^r}{(\alpha - i - w)(L_i + u)^r + \sum_{q=0}^{i-1}(L_{q+1})^r},
$$

where the expression $x^{\underline{y}}$ stands for the "falling power" $x \times (x - 1) \times \cdots \times (x - y + 1)$.

In our following analysis, the important intuition is that asymptotically $L_i + u - 1 \approx L_i + u$, but $(L_i)^r \ll (L_i + u)^r$. Replacing $\sum_{q=0}^{i-1}(L_{q+1})^r$ by $i(L_i)^r$, we get

$$
(16) \qquad \Pr(u, v, k_1) \geq (\alpha - i - 1)^{\underline{k_1-1}} \prod_{w=0}^{k_1-1} \frac{(L_i + u - 1)^r}{(\alpha - i - w)(L_i + u)^r + i(L_i)^r}.
$$

Adding and subtracting $(L_i + u)^r$ to the numerator in (16) and simplifying, we get

$$
(17) \quad \Pr(u, v, k_1) \geq (\alpha - i - 1)^{\underline{k_1-1}} \left(\prod_{w=0}^{k_1-1} \frac{(L_i + u)^r}{(\alpha - i - w)(L_i + u)^r + i(L_i)^r} - \frac{\delta_1(u, i)}{\alpha - i - w} \right),
$$

where $\delta_1(u, i) = ((L_i + u)^r - (L_i + u - 1)^r)/(L_i + u)^r$. Adding and subtracting $(i/\alpha - i - w) \times (L_i)^r$ to the numerator of the first rational term in (17) and simplifying, we get

$$
(18) \qquad \Pr(u, v, k_1) \geq \frac{(\alpha - i - 1)^{\underline{k_1-1}}}{(\alpha - i)^{\underline{k_1}}} \left(1 - (\delta_1(u, i) + \delta_2(u, i)) \right)^{k_1},
$$

where $\delta_2(u, i) = i(L_i)^r/(L_i + u)^r$. With the expression for $\Pr(u, v, k_1)$ from (18) it is easy to verify that the leading term of $NumFaults_{P_1}(\pi_i)$ is $|\pi_i| \times \max\{0, 1 - k/(\alpha - i)\}$, which is the number of faults incurred by F_1^j on π_i. The error term $\epsilon = NumFaults_{P_1}(\pi_i) - NumFaults_{F_1^j}(\pi_i)$ equals

$$(19) \qquad \frac{1}{\alpha - i} \sum_{u=1}^{|\pi_i|/(\alpha-i)} \sum_{v=0}^{\alpha-i-1} \sum_{k_1=1}^{k} \epsilon(u, i, k_1),$$

where

$$\epsilon(u, i, k_1) = 1 - (1 - \delta_1(u, i) - \delta_2(u, i))^{k_1}.$$

The following facts can be verified by using the asymptotic techniques from [15, Chapter 9]:
1. $\delta_1(u, i) \le r/(L_i + u - 1)$ if $L_i + u - 1 \ge r$;
2. $\delta_2(u, i) \le i \times \exp(-ru/2L_i)$ if $u \le L_i$, and $\delta_2(u, i) \le i2^{-r}$ if $u \ge L_i$.

The lower-order terms arising from the binomial expansion of $(1 - (\delta_1(u, i) + \delta_2(u, i))^{k_1}$ from (19) (i.e., terms of degree 2 or greater) can be disregarded if $L_i + u - 1 \ge kr$ and $u \ge \sqrt{3\eta} \log(\alpha k)$. When $L_i + u - 1 < kr$ or $u < \sqrt{3\eta} \log(\alpha k)$, we can bound $\epsilon(u, i, k_1)$ by 1; the net contribution of this to ϵ is $O(\alpha r \ln(\alpha k))$. Disregarding lower-order terms in (19) and using the expressions from Facts 1 and 2 above, we get $\epsilon = O(\alpha k^2 r \ln \eta + \alpha k^2 \eta/r) = O(\alpha k^2 \sqrt{\eta} \log \eta) = O(\alpha k^2 j 2^j)$. $\qquad \square$

5. Generalizing P_1 to get P. In this section we prove Theorem 2 by constructing our optimal prefetcher P. The prefetcher P is a mix of P_1 and the character-based version of the Lempel–Ziv algorithm for data compression. The original Lempel–Ziv algorithm is a word-based data compression algorithm that parses the input string x_1^n into distinct substrings $x_0, x_1, x_2, \ldots, x_c$ such that, for all $j \ge 1$, substring x_j without its last character is equal to some x_i for $0 \le i < j$. (We use the convention that $x_0 = \lambda$, the empty substring.) It encodes the string one substring at a time. Since the substrings are prefix-closed, they can be represented by a dynamically growing tree T (the "LZ tree"), with the nodes of the tree representing the substrings and node x_i being an ancestor of node x_j if substring x_i is a prefix of substring x_j; λ is the root of the tree. An example of the LZ tree is given in Figure 1a.

Let $x(z)$ be the sequence of pages seen until now by P when at state z. At the end of a parse, prefetcher P positions itself at the root of the LZ tree. It looks at the subsequence $x(z)$ at its current state z and simulates P_1 on $x(z)$ to prefetch for the next page. (Algorithm P_1 breaks $x(z)$ into r-subsequences and prefetches based on the current r-subsequence at state z as described in section 3. Note that P_1 does not have to maintain $x(z)$ explicitly; it only has to maintain counts for the different pages.) On observing the next page request j, it updates $x(z)$, moves down the transition labeled by j, and prefetches the next page similarly by simulating P_1 on the sequence of pages seen at the new current state. On reaching a leaf state, it prefetches k pages at random, and the next request ends a parse. The important point is that although the counts for some or all of the transitions can be 0 (since algorithm P_1 resets the counts for all pages to 0 at the beginning of an r-subsequence), the transitions themselves are retained in the tree. An example snapshot of the data structure of P is given in Figure 1b.

We now briefly explain why P is optimal against an arbitrary s-state machine (Theorem 2) using the interesting approach of [12]. An mth-order Markov prefetcher
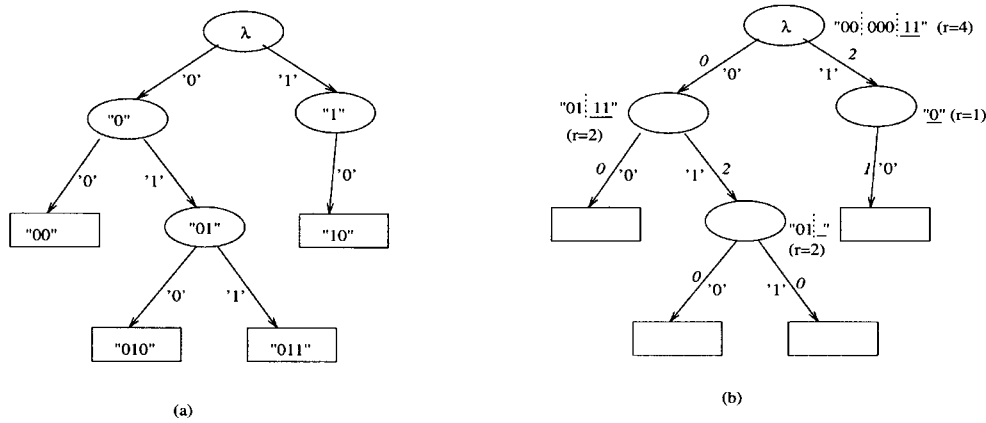
FIG. 1. *Snapshot of data structure for algorithm P. Assume for simplicity that our alphabet is* $\{0,1\}$. *We consider the page request sequence* x_1^t = *"00001010011110....". The Ziv–Lempel encoder parses this string as "(0)(00)(01)(010)(011)(1)(10)....". The tree T that is built at the end of the seventh parse is pictured above in* (a). *In* (b), *next to each node/state z of the tree we give the sequence of page requests* $x(z)$ *seen at that state. For example, for any page request sequence* x_1^t *that is parsed by the Lempel–Ziv data compressor into distinct substrings* λ, x_1, x_2, ..., x_c, *the first page of each substring* x_i, $1 \leq i \leq c$, *forms* $x(\lambda)$, *the sequence of pages requested when the current state is the root of the tree. In* (b), $x(\lambda) = 0000011$. *The dotted vertical lines in the sequences delimit the r-subsequences, and the underlined portion is the current r-subsequence. The counts (given in italics) on the transitions out of each state z are the counts obtained by simulating* P_1 *on* $x(z)$.

predicts its k choices for the next page based solely on the previous m page requests of the sequence. In particular, an mth-order prefetcher can be described by an FSP having α^m states, where each state is labeled by an m-page context (denoting the last m pages requested), and the transitions denote the unique change from one m-context to the next. The pages to prefetch are determined solely by the state that the mth-order Markov prefetcher is in, which is equivalent to the most recent m pages requested. The basic idea of the proof is to compare both prefetcher P and the best s-state prefetcher with mth-order Markov prefetchers.

If we let m be large, an mth-order Markov prefetcher achieves, for every sequence σ_1^n and any s, a fault rate close to the fault rate of the best s-state prefetcher. In particular, by simple extensions to [12, Theorem 2] as shown in Appendix B, we see that

$$(20) \qquad \qquad Fault_{M(\alpha^m)}(\sigma_1^n) \leq Fault_{\mathcal{F}(s)}(\sigma_1^n) + O\left(\sqrt{\frac{\log s}{m+1}}\right).$$

The idea of the proof in [12] is to consider a "cross-product" machine of the mth-order Markov predictor and the s state prefetcher and to show that the cross-product machine is not much better than either of its constituents.

The prefetcher P can be looked upon as a Markov prefetcher of growing order. In particular, most nodes in the tree T built by P (i.e., nodes below a depth of m) have a context of length greater than m. Hence one would intuitively expect that in the limit as $n \to \infty$, prefetcher P will "beat" any mth-order Markov predictor. Theorem 1 can be applied to each node of T, and by carefully summing the errors

over each node we get

$$(21) \qquad\qquad Fault_P(\sigma_1^n) \leq Fault_{M(\alpha^m)}(\sigma_1^n) + \delta(n, m),$$

where for a fixed m, $\delta(n, m) = O((\log \log n)/\sqrt{\log n})$. This idea was used in [12, Theorem 4] for a binary alphabet, and the simple changes required to obtain (21) are summarized in Appendix B. Theorem 2 follows from (20) and (21).

Given that P_1 is optimal against $\mathcal{F}(1)$ (Theorem 1), in order to show optimality of P against $\mathcal{F}(s)$ by the approach described above, we need to extend some results of [12] to hold for the prefetching problem. These extensions are simple as described in Appendix B. The intuition for why these extensions are simple is because the comparisons are primarily between two "offline" algorithms, and the online algorithm is not much involved, as opposed to the more complex analysis of section 4.

6. Constant-time prediction. In this section we prove Theorem 3 by showing that our prefetcher P runs in constant time (independent of α, k) on the average for each of the pages it prefetches into cache.

In section 5, we showed that it suffices to consider one-state prefetchers; the prefetcher at each step uses the appropriate P_1 to generate random variates according to a dynamically changing set of weights. We showed earlier that P_1's prediction strategy is optimal, in which we successively pick a page at random (without replacement) with probabilities in proportion to the boosted frequency counts $(f_0)^r$, $(f_1)^r$, $\ldots$, $(f_{\alpha-1})^r$, where $r \approx \sqrt{t}$. (Actually, we use $r = 2^j$, where $4^{j-1} < t \leq 4^j$, so that r seldom changes. The frequency counts f_i are reset to zero when r changes.)

The general problem of generating a random variate with a value in the range $\{0, 1, 2, \ldots, \alpha - 1\}$ and distributed according to α dynamically changing weights $w_0, w_1, w_2, \ldots, w_{\alpha-1}$ is solved optimally by Matias, Vitter, and Ni [26, section 5]. The idea at an intuitive level is to group the weights into ranges according to their values. Range j stores weights in the range $[2^j, 2^{j+1})$. Each range is said to have a weight equal to the sum of the weights it contains. With high probability, the individual weight chosen during the generation will be within the first $O(\log \alpha)$ ranges, so each successive group of $O(\log \alpha)$ ranges should be processed in a recursive data structure according to the weights of the ranges. The use of the rejection method [21] is used to adjust the probabilities of generation appropriately, since the weights in each bucket may vary by a factor of 2. After two recursive levels, the problem reduces to generating one of $O(\log \log \alpha)$ weights, each in the range $[1, \log \alpha]$, which can be done dynamically in constant time by the clever table lookup method of Hagerup, Mehlhorn, and Munro [16].

There is also extensive concern in [26] about the choice of hashing parameters in the universal hashing schemes used to get linear space, since no a priori bound on the key values is known. (In fact, a constant-time solution to the general dictionary problem is proposed in [26].) The model of computation allows arithmetic computation and truncated logarithms of quantities up to value $O(W)$, where W is the maximum weight.

In our application, the computation assumption of [26] is unreasonable, since it allows constant-time operations on arbitrary numbers of bits. We make the stronger requirement often used in algorithm design that the standard arithmetic operations (such as addition, multiplication, division, and using exponents and logarithms) take constant time with finite-precision quantities of $O(\log n)$ bits, where n is the length of the sequence of page requests. However, the boosted frequencies $w_i = (f_i)^r$ used in the random variate generation can be as large as $n^{\sqrt{n}}$ in value, which cannot be

manipulated efficiently. Fortunately we can get around the precision problem by approximating w_i by $2^{\lceil r \lg f_i \rceil}$. The first level of the algorithm in [26] is applied to these approximated weights, using arithmetic on the exponents, which involves only $O(\log n)$ bits. For example, we can determine the bucket j that contains $2^{\lceil r \lg f_i \rceil}$ in constant time using operations on $O(\log n)$ bits by noting that j can be represented with only about $\lg \lg((f_i)^r) = \lg r + \lg \lg f_i \leq 2 \lg n$ bits. The range j can be computed, therefore, in constant time using $O(\log n)$-bit arithmetic. The resulting recursive subproblems have polynomial-sized weights, and the rest of the construction continues as in [26].

Because of the initial approximation, if the "approximated" page i is selected for generation, a final acceptance–rejection test must be done before actually choosing page i; the acceptance probability is $(f_i)^r/2^{\lceil r \lg f_i \rceil}$, which is at least $1/2$. This test can be done conceptually by generating a uniform random integer U in the range $[1, 2^{j+1})$ and testing if $U \leq (f_i)^r$, but handling quantities of that magnitude is infeasible, as mentioned above. It suffices to determine if $\lg U \leq r \lg f_i$, which can be done in constant time using finite precision by generating the exponentially distributed random variate $\lg U$ directly [21, p. 128]. The expected number of steps needed before the acceptance or rejection is determined is a small constant, so finite precision suffices. This completes the proof of Theorem 3.

7. Conclusions. We have studied the problem of prediction of sequences (of pages requests, for example) drawn from a finite but arbitrary alphabet of cardinality α, in which we can make, at each time step, k predictions for the next item (page). This corresponds to the problem of pure prefetching in databases. We have developed a simple randomized weighting algorithm P_1 and have combined it with the Lempel–Ziv data compressor to get an efficient prefetcher P. We have shown analytically that P's fault rate converges almost surely to that of the best FSP for every (worst-case) sequence of page requests. It has been shown in [7] that any optimal algorithm for the binary alphabet case has to be necessarily randomized. Because of the way our algorithm is designed, we need to spend at most constant expected time making the random choices for each prediction, which is optimal. Thus the algorithm is simultaneously optimal with respect to fault rate and running time.

An open problem is to study if there are stronger analysis models closer to the competitive model that would permit prediction problems such as prefetching to be studied. It would also be interesting to improve the convergence bounds while maintaining optimal running time.

We have also investigated nonpure prefetching in [10, 35], in which there may not always be enough time to load the cache with k pages before the user issues the next page request, and prefetching requests may have to be done in advance. We have also described therein a nice way of gathering the statistics with no I/O overhead. The resulting prefetcher is practical in terms of time, disk accesses, and fault rate and outperforms other known prefetchers. We also expect that our results apply to prefetchers based on data compression methods other than Lempel–Ziv that are optimal in various models.

Appendix A. Nonoptimal prefetchers. In section A.1 we give a simple example which illustrates that no deterministic algorithm can be optimal for prefetching in the worst case. In section A.2, we show that the *Proportional* algorithm, which at time t prefetches page i with probability proportional to $f_i(\sigma_1^t)$, where $f_i(\sigma_1^t)$, for $0 \leq i \leq \alpha - 1$, denotes the number of times page i appears in σ_1^t, is not optimal.

For both proofs of nonoptimality, we develop a page request sequence with alphabet $A = \{0, 1\}$, cache size $k = 1$, and compare it with the best 1-state prefetcher.

From an intuitive standpoint, algorithm *Proportional* is too conservative, while deterministic algorithms are "too naive" and hence susceptible to worst-case-type adversaries (like FSPs). Algorithm P_1, by choosing the high probability pages with very high likelihood, closely tracks the optimal. The tricky issues are proving the optimality of P_1 and making predictions in a computationally efficient fashion.

A.1. Deterministic algorithms. It is easy to see that the fault rate of an optimal 1-state prefetcher for any sequence with $\alpha = 2$ and $k = 1$ is at most $1/2$, since it predicts at each time instant the overall most frequent page. We can make a deterministic algorithm fault at every page request by creating a sequence such that the next page request is for the page not in cache.

A.2. Algorithm *Proportional*. Consider the page request sequence

$$\sigma_1^n = 01010101 \ldots 010000 \ldots 00,$$

where the first "0101..." subsequence is of length $n/2$, and it is followed by $n/2$ 0's. The fault rate of the optimal 1-state prefetcher (which always predicts "0" to be the next page request) is $1/4$. Algorithm *Proportional* is expected to incur at least $1/2 \times n/2$ faults in the first $n/2$ page requests. Since the proportion of 0's in the entire sequence is $3/4$, algorithm *Proportional* incurs, on the average, more than $1/4 \times n/2$ faults for the last $n/2$ page requests. This implies a net average fault rate of more than $3/8$, which is clearly suboptimal.

Appendix B. Proof of Theorem 2. In this section we continue the discussion from section 5 and present the required extensions to the results of [12] in order to prove optimality of prefetcher P. (Recall that [12] deals with the binary alphabet case, which corresponds to $\alpha = 2$, $k = 1$.) The main objective of this section is to show the necessary extensions required to prove (20) and (21) (which are the counterparts for Theorems 2 and 4 from [12]).

We start with a definition of mth-order Markov prefetchers.

DEFINITION 6. *An mth-order Markov prefetcher prefetches for its next page based solely on the previous m page requests of the sequence. Using the notation from Definition 1, an mth-order Markov prefetcher has α^m states, where each state is (represents) an m-context $(x_1, x_2, \ldots, x_m)$, and $g((x_1, \ldots, x_m), u) = (x_2, \ldots, x_m, u)$. We denote the fault rate of an mth-order prefetcher by $Fault_{M(\alpha^m)}(\sigma_1^n)$, where $M(\alpha^m) \subseteq \mathcal{F}(\alpha^m)$.*

We also introduce notation to help describe faults easily.

DEFINITION 7. *Given an α-probability vector $\vec{p} = (p_0, \ldots, p_{\alpha-1})$, we denote by $\min_{\alpha-k}(\vec{p})$ the sum of the minimum $\alpha - k$ elements of $\vec{p}$. In other words if $p_0 \geq p_1 \geq \cdots \geq p_{\alpha-1}$, then $\min_{\alpha-k}(\vec{p}) = \sum_{i=k}^{\alpha-1} p_i$.*

Proving (20) is equivalent to showing that Theorem 2 from [12] holds for prefetching. As mentioned in section 5, the proof of [12, Theorem 2] is based on comparing the s-state prefetcher and the mth-order Markov prefetcher with a cross product of these two machines.[1] Theorem 2 of [12] is strongly dependent on [12, Lemma 1], where the mth-order Markov machine is compared with the cross-product machine. The basic idea of [12, Lemma 1] is to bound prediction error by a function of the

---

[1] The term "cross product" is taken from [36], where a similar product of two machines was used to prove the optimality of prefetchers under a Markov source model.

empirical entropies, and it is based on the following fact: for every $0 \leq p, q \leq 1$,

$$(22) \qquad p \log \frac{p}{q} + (1 - p) \log \frac{1 - p}{1 - q} \geq \frac{2}{\ln 2} \left(\min\{p, 1 - p\} - \min\{q, 1 - q\} \right)^2.$$

It is not hard to see that the left-hand side of (22) is closely related to the entropy and the right-hand side of (22) to the prediction error. (The idea of bounding error for prefetching by bounding coding length differences was used independently in [36] to derive optimal prefetchers from data compressors under a Markov source input model.) A fact equivalent to (22) that we prove for our prefetching problem (via Lemmas 4 and 5 below) is the following: given two probability vectors $\vec{p}$ and $\vec{q}$,

$$(23) \qquad \sum_1^\alpha p_i \ln \frac{p_i}{q_i} \geq \frac{1}{2} \left(\min_{\alpha - k}(\vec{p}) - \min_{\alpha - k}(\vec{q}) \right)^2.$$

The proof of [12, Lemma 1] follows from (22), Jensen's inequality, and the convexity of the square function. The proof of [12, Theorem 2] follows from [12, Lemma 1], Jensen's inequality, the concavity of the square root function, and the chain rule of conditional entropies. Except for (22), the other aspects of the proof of [12, Theorem 2] are effectively independent of the alphabet and cache size. To derive (20), we consider the proof of [12, Theorem 2] and uniformly replace $\min\{p_0, p_1\}$, where p_0 and p_1 are the probabilities of a 0 and a 1, by $\min_{\alpha - k}(\vec{p})$, where $\vec{p}$ is the corresponding α-probability vector for prefetching, replace summations over $\{0, 1\}$ by summations over the alphabet A, and use (23) in place of (22).

We now prove (23) using the following two lemmas.

LEMMA 4. *Given two α-probability vectors $\vec{p}$ and $\vec{q}$, we have*

$$\min_{\alpha - k}(\vec{p}) - \min_{\alpha - k}(\vec{q}) \leq \sum_{i=0}^{\alpha - 1} |p_i - q_i|.$$

Proof. Without loss of generality assume $p_0 \geq p_1 \geq \cdots \geq p_{\alpha - 1}$. Let $X = \{0, 1, \ldots, k - 1\}$, and let $Y = \{k, k + 1, \ldots, \alpha - 1\}$. Hence $\min_{\alpha - k}(\vec{p}) = \sum_{i \in Y} p_i$. Let $Z = \{i_1, i_2, \ldots, i_{\alpha - k}\}$ be the $\alpha - k$ pages with minimum count in $\vec{q}$. Let $U = Z \cap Y$, and $V = Z \cap X$. By definition, $\min_{\alpha - k}(\vec{p}) - \min_{\alpha - k}(\vec{q}) = \sum_{i \in Y} p_i - \sum_{i \in Z} q_i$. Since by assumption $p_0 \geq p_1 \geq \cdots \geq p_{\alpha - 1}$, we have

$$\min_{\alpha - k}(\vec{p}) - \min_{\alpha - k}(\vec{q}) \leq \sum_{i \in U} (p_i - q_i) + \sum_{i \in V} (p_i - q_i) \leq \sum_{i \in A} |p_i - q_i|,$$

where A is the alphabet as given in Definition 1.   □

The next lemma is well known; the proof can be found in [2, 36]. The summation on the right-hand side of the equation in the lemma is the Kullback–Leibler divergence of $\vec{q}$ w.r.t. $\vec{p}$.

LEMMA 5. *Given two probability vectors $(p_0, \ldots, p_{\alpha - 1})$ and $(q_0, \ldots, q_{\alpha - 1})$, we have*

$$\left(\sum_{i=0}^\alpha |p_i - q_i| \right)^2 \leq 2 \sum_{i=1}^\alpha p_i \ln \frac{p_i}{q_i}.$$

To verify (21) (i.e., the equivalent of [12, Theorem 4] for prefetching), the proof technique presented in [12] carries over with virtually no change; we need to use

Theorem 1 of our paper in place of [12, Theorem 1]. (As done earlier, we do need to uniformly replace $\min\{p_0, p_1\}$ by $\min_{\alpha-k}(\vec{p})$ and replace summations over $\{0, 1\}$ by summations over the alphabet A.) The basic idea of the proof is to consider separately each node of the tree T created by P, look at the faults for the pages requested when at that node, and take the average of these individual faults weighted by the number of times the node is visited. The main observation is that for most nodes (i.e., nodes below a depth of m), there is a mapping from the nodes of T to the states of the mth-order Markov prefetcher; hence bounding the error at each node of T involves comparing with the best one-state prefetcher, which is done in Theorem 1. Since there are at most $c = O(n/\log n)$ nodes in T [37], and the one-state error from Theorem 1 is $O((\log n)\sqrt{n}/n)$, the net error turns out to be $O((\log(n/c))\sqrt{n/c}/(n/c)) = O((\log\log n)/\sqrt{\log n})$. (This is as opposed to the case of $\alpha = 2$, $k = 1$ studied in [12], where the one-state error is $O(1/\sqrt{n})$, yielding a net error of $O(1/\sqrt{\log n})$.)

This completes our description of the extensions to [12] required to prove Theorem 2.

REFERENCES

[1] D. ALDOUS AND U. VAZIRANI, *A Markovian extension of Valiant's learning model*, in Proc. 31st Annual IEEE Symposium on Foundations of Computer Science, October 1990, pp. 392–396.

[2] Y. AMIT AND M. MILLER, *Large Deviations for Coding Markov Chains and Gibbs Random Fields*, Technical Report, Washington University, St. Louis, MO, 1990.

[3] L. A. BELADY, *A study of replacement algorithms for virtual storage computers*, IBM Systems J., 5 (1966), pp. 78–101.

[4] D. BLACKWELL, *An analog to the minimax theorem for vector payoffs*, Pacific J. Math., 6 (1956), pp. 1–8.

[5] A. BORODIN, S. IRANI, P. RAGHAVAN, AND B. SCHIEBER, *Competitive paging with locality of reference*, in Proc. 23rd Annual ACM Symposium on Theory of Computation, May 1991.

[6] T. F. CHEN AND J. L. BAER, *Reducing memory latency via non-blocking and prefetching caches*, in Proc. 5th Internat. Conf. on Architectural Support for Programming Languages and Operating Systems, Department of Computer Science and Engineering, University of Washington, Boston, MA, October 1992.

[7] T. M. COVER, *Behavior of predictors of binary sequences*, in Proc. 4th Prague Conference on Information Theory, Statistical Decision Functions, Random Processes, Publishing House of the Czechoslovak Academy of Sciences, Prague, 1967, pp. 263–272.

[8] T. M. COVER AND A. SHENHAR, *Compound Bayes predictors with apparent Markov structure*, IEEE Trans. Systems Man Cybernet., V SMC-7 (1977), pp. 421–424.

[9] T. M. COVER AND J. A. THOMAS, *Elements of Information Theory*, Wiley, New York, 1991.

[10] K. CUREWITZ, P. KRISHNAN, AND J. S. VITTER, *Practical prefetching via data compression*, in Proc. 1993 ACM SIGMOD International Conference on Management of Data, May 1993, pp. 257–266.

[11] P. J. DENNING, *Working sets past and present*, IEEE Trans. Software Engrg., SE-6 (1980), pp. 64–84.

[12] M. FEDER, N. MERHAV, AND M. GUTMAN, *Universal prediction of individual sequences*, IEEE Trans. Inform. Theory, IT-38 (1992), pp. 1258–1270.

[13] A. FIAT, R. M. KARP, M. LUBY, L. A. MCGEOCH, D. D. SLEATOR, AND N. E. YOUNG, *On competitive algorithms for paging problems*, J. Algorithms, 12 (1991), pp. 685–699.

[14] R. G. GALLAGER, *Information Theory and Reliable Communication*, Wiley, New York, 1968.

[15] R. L. GRAHAM, D. E. KNUTH, AND O. PATASHNIK, *Concrete Mathematics*, Addison–Wesley, Reading, MA, 1989.

[16] T. HAGERUP, K. MEHLHORN, AND I. MUNRO, *Optimal algorithms for generating time varying discrete random variables*, in Proc. of the 20th Annual International Coll. on Automata Languages and Prog., Lecture Notes in Comput. Sci. 700, Springer, New York, 1993, pp. 253–264.

[17] J. F. HANNAN, *Approximation to Bayes Risk in Repeated Plays*, Contributions to the Theory of Games, Vol. 3, Annals of Mathematical Studies, Princeton University Press, Princeton, NJ, 1957, pp. 97–139.

[18] P. G. HOWARD AND J. S. VITTER, *Analysis of arithmetic coding for data compression*, Invited paper in Special Issue on Data Compression for Images and Texts, Inform. Process. Management, 28 (1992), pp. 749–763.

[19] S. IRANI, A. R. KARLIN, AND S. PHILLIPS, *Strongly competitive algorithms for paging with locality of reference*, in Proc. 3rd Annual ACM-SIAM Symposium of Discrete Algorithms, January 1992.

[20] A. R. KARLIN, S. J. PHILLIPS, AND P. RAGHAVAN, *Markov paging*, in Proc. 33rd Annual IEEE Conference on Foundations of Computer Science, October 1992, pp. 208–217.

[21] D. KNUTH, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 2nd ed., Addison-Wesley, Reading, MA, 1981.

[22] D. F. KOTZ AND C. S. ELLIS, *Prefetching in File Systems for MIMD Multiprocessors*, IEEE Transactions on Parallel and Distributed Systems, Vol. 1, April 1990, pp. 218–230.

[23] P. KRISHNAN AND J. S. VITTER, *Optimal Prediction for Prefetching in the Worst Case*, Technical Report DUKE–CS–93–26, Duke University , Durham, NC, 1993.

[24] P. LAIRD, *TDAG: An Algorithm for Learning to Predict Discrete Sequences*, FIA-92-01, NASA Ames Research Center, AI Research Branch, Moffet Field, CA, 1992.

[25] G. G. LANGDON, *A Note on the Ziv–Lempel Model for Compressing Individual Sequences*, IEEE Trans. Inform. Theory, Vol. 29, March 1983, pp. 284–287.

[26] Y. MATIAS, J. S. VITTER, AND W. C. NI, *Dynamic generation of discrete random variates*, in Proc. 4th Annual SIAM/ACM Symposium on Discrete Algorithms, Austin, TX, January 1993, pp. 361–370.

[27] L. A. MCGEOCH AND D. D. SLEATOR, *A strongly competitive randomized paging algorithm*, Algorithmica, 6 (1991), pp. 816–825.

[28] N. MERHAV AND M. FEDER, *Universal Sequential Learning and Decision from Individual Data Sequences*, in Proc. 5th ACM Workshop on Computational Learning Theory, Santa Cruz, July 1992.

[29] T. C. MOWRY, M. S. LAM, AND A. GUPTA, *Design and evaluation of a compiler algorithm for prefetching*, in Proc. 5th International Conference on Architectural Support for Programming Languages and Operating Systems Computer Systems Laboratory, Boston, MA, October 1992.

[30] M. PALMER AND S. ZDONIK, *Fido: A cache that learns to fetch*, in Proc. 1991 International Conference on Very Large Databases, Barcelona, September 1991.

[31] A. ROGERS AND K. LI, *Software support for speculative loads*, in Proc. 5th International Conference on Architectural Support for Programming Languages and Operating Systems, Department of Computer Science, Boston, MA, October 1992.

[32] K. SALEM, *Adaptive Prefetching for Disk Buffers*, CESDIS, Goddard Space Flight Center, Greenbelt, MD, TR–91–46, January 1991.

[33] G. S. SHEDLER AND C. TUNG, *Locality in page reference strings*, SIAM J. Comput., 1 (1972), pp. 218–241.

[34] D. D. SLEATOR AND R. E. TARJAN, *Amortized efficiency of list update and paging rules*, Communications of the ACM, 28 (1985), pp. 202–208.

[35] J. S. VITTER, K. CUREWITZ, AND P. KRISHNAN, *Online Background Predictors and Prefetchers*, Duke University, United States Patent No. 5,485,609, January 16, 1996.

[36] J. S. VITTER AND P. KRISHNAN, *Optimal prefetching via data compression*, J. Assoc. Comput. Mach., 143 (1996), pp. 771–793.

[37] J. ZIV AND A. LEMPEL, *Compression of individual sequences via variable-rate coding*, IEEE Trans. Inform. Theory, 24 (1978), pp. 530–536.