Optimal Prefetching via Data Compression

JEFFREY SCOTT VITTER

Duke University, Durham, North Carolina

AND

P. KRISHNAN

Bell Laboratories, Holmdel, New Jersey

Abstract. Caching and prefetching are important mechanisms for speeding up access time to data on secondary storage. Recent work in competitive online algorithms has uncovered several promising new algorithms for caching. In this paper, we apply a form of the competitive philosophy for the first time to the problem of prefetching to develop an optimal universal prefetcher in terms of fault rate, with particular applications to large-scale databases and hypertext systems. Our prediction algorithms for prefetching are novel in that they are based on data compression techniques that are both theoretically optimal and good in practice. Intuitively, in order to compress data effectively, you have to be able to predict future data well, and thus good data compressors should be able to predict well for purposes of prefetching. We show for powerful models such as Markov sources and mth order Markov sources that the page fault rates incurred by our prefetching algorithms are optimal in the limit for almost all sequences of page requests.

Categories and Subject Descriptors: D.4.2 [**Operating Systems**]: Storage Management—*swapping, virtual memory*; D.4.8 [**Operating Systems**]: Performance—*stochastic analysis*; E.4 [**Coding and Information Theory**]—*data compaction and compression*; I.2.6 [**Artificial Intelligence**]: Learning.

General Terms: Algorithms, design, performance, theory.

Additional Key Words and Phrases: Caching, competitive analysis, databases, data compression, fault rate, hypertext, Markov source, prediction, prefetching, secondary stage, universal prefetcher.

An extended abstract of this work appears in *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, (Puerto Rico, October 1991).

The work was performed while the authors were at Brown University.

Support was provided in part by a National Science Foundation Presidential Young Investigator Award CCR 90-47466 with matching funds from IBM, by NSF research grant CCR 90-07851, by Army Research Office grant DAAL03-91-G-0035, and by the Office of Naval Research and the Defense Advanced Research Projects Agency under contract N00014-83-K-0146 and ARPA order 6320, amendment 1.

Authors' addresses: J. S. Vitter, Department of Computer Science, Duke University, Durham, NC 27708-0129, e-mail: jsv@cs.duke.edu; P. Krishnan, Bell Laboratories, 101 Crawfords Corner Road, Holmdel NJ 07733-3030, e-mail: pk@research.bell-labs.com.

1. *Introduction*

Computer memory is often modeled as a two-level memory consisting of a relatively small but fast *cache* (such as internal memory) and a relatively large but slow memory (such as disk storage). Such a two-level model of memory corresponds to the *client-server* paradigm of computing, in which the client is the database user (or application) and the server manages the database; we can think of the client workstation as a cache, and the server as slow memory. It also models 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 page fault occurs is called *demand fetching*. The problem of *cache replacement* or *caching* is to decide which pages to remove from cache to accommodate the incoming pages.

In many systems and database applications, the user spends a significant amount of time processing a page, and the computer and the I/O system are typically idle during that period. The computer can use this time to predict which page the user will request next, and fetch that page into cache (if it is not already in cache) in the background. When the user requests the page, it is available in cache, and the user perceives a faster response time. *Prefetching* is a method of improving response time performance by anticipating future user requests, and getting the necessary data into cache in the background before an explicit request is made by the application. The primary action in prefetching is predicting which page will be next requested by the user. Related issues are the timing of prefetch requests (so that the data arrives before it is requested) and cache replacement (deciding which pages to evict from cache to accommodate the prefetched pages).

In many hypertext and iterative 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 in this setting as *pure prefetching*, and we restrict our analysis in this paper to pure prefetching. Pure prefetching is an important model that helps in analyzing the benefits of fetching pages in the background. Pure prefetching is mathematically elegant since it isolates the prediction component of prefetching from the issue of cache replacement. At the same time, pure prefetchers can be converted into general prefetchers using techniques described in Curewitz et al. [1993]. We refer to this issue again in Sections 6 and 7.

Prior work on prefetching has been primarily empirical. The UNIX system uses a simple scheme for prefetching that is optimized for sequential file access; it anticipates and prefetches page $i + 1$ when a request is made for page i. Current database systems perform prefetching using such sequential prefetching techniques derived from older virtual memory prefetchers. The I/O bottleneck is seriously impeding performance in large-scale databases, and the demand for improving response time performance is growing [Brady 1986]. The older virtual memory-based prefetchers are inadequate for newer object-oriented and hypertext applications, and this has stimulated renewed interest in developing improved algorithms for prefetching.[1]

In this paper, *we give the first provable theoretical bounds on the performance of prediction algorithms for prefetching*. The first important point in our analytical study of prefetching is to develop a framework for proving the goodness of prefetchers. We say that an algorithm is *online* if it must make its decisions based only on the past history. An *offline* algorithm can use the knowledge of the future. Any implementable algorithm for caching or prefetching must clearly be online. The notion of *competitiveness* introduced by Sleator and Tarjan [1985] determines the goodness of an online algorithm by comparing its performance to that of offline algorithms. An online caching or prefetching algorithm is *c-competitive* if there exists a constant b such that, for any sequence of page requests, the number of page faults the online algorithm incurs is at most b more than c times the number of faults of an optimal offline algorithm. Competitive algorithms for caching are well examined in the literature.[2]

It is not reasonable to expect algorithms to be competitive in this sense for prefetching. An optimal offline algorithm for prefetching will never fault 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.

We resolve this matter by considering powerful probabilistic models. In our main model, the sequence of page requests is assumed to be generated by a labeled deterministic finite state automaton with probabilities on the transitions (a *probabilistic FSA* or *Markov source*). In the second model, we consider the special case of *mth order Markov sources*, in which the states correspond to the contexts of the previous m page requests. We evaluate our prefetching algorithm relative to the best online algorithm *that has complete knowledge of the structure and transition probabilities of the Markov source*. For convenience, our normalized measure of performance is the *page fault rate*, which is defined to be the total number of page faults divided by the total number of page requests.

Prefetching is a learning problem that involves predicting the page requests of the user. Our novel approach is to use optimal data compression methods to do optimal prefetching. Our motivation is recent work in computational learning theory [Blumer et al. 1987, 1989; Board and Pitt 1990], which has shown that prediction is synonymous with generalization and data compression. Our intuition in this paper is that, in order to compress data well, one has to be able to predict future data well, and hence a good data compressor should also be a good predictor: If a data compressor expects a certain character to be next with a very high probability, it will assign that character a relatively small code. In the end, if the net code length is small, then the predictions of the data compressor must have been good. Independently to our work, there is work in the information theory community on algorithms for binary sequences (corresponding to a universe of two pages) that make one prediction for the next page (correspond-

[1] See, for example, Chen and Baer [1992], Laird [1992], Mowry et al. [1992], Palmer and Zdonik [1991], and Rodgers and Li [1992].

[2] See, for example, Borodin et al. [1991], Fiat et al. [1991], Irani et al. [1992], Karlin et al. [1992], McGeoch and Sleator [1989], and Sleator and Tarjan [1985].

ing to cache size $k = 1$);[3] the binary case is not applicable to our prefetching scenario.

In this paper, we adapt an optimal data compression method to get an optimal pure prefetching algorithm for each of our models. Our models and results are summarized in the next section. In Section 3, for our main Markov source model, we apply a character-by-character version of the Ziv–Lempel data compression algorithm [Ziv and Lampel 1978]. In Section 4, we compare our online algorithm to the best algorithm that has full knowledge of the Markov source. We show that the page fault rate of our algorithm converges for almost all page request sequences to this best algorithm's page fault rate. The trick is to show that good compression results in good prefetching. In Section 5, we show faster convergence to optimality for mth order Markov sources. In Section 6, we discuss issues related to non-pure prefetching, and conclude in Section 7.

2. *Page Request Models and Main Results*

In keeping with the analogy between prefetching and text compression, we use the terms "page" and "character" interchangeably. We denote the cache size by k and the total number of different pages (or alphabet size) by α. The logarithm of x to the base 2 is denoted by $\lg x$, the natural logarithm of x is denoted by $\ln x$, and the empty string is denoted by λ.

Definition 2.1 [*Gallager* 1968]. We define a *probabilistic finite state automaton* (*probabilistic FSA*) as a quintuple (S, A, g, p, z_0), where S is a finite set of states with $|S| = s$, A is a finite alphabet of size α, g is a deterministic "next state" function that maps $S \times A$ into S, p is a "probability assignment function" that maps $S \times A$ into [0, 1] with the restriction that $\ _{i \in A} p(z, i) = 1$ for all $z \in S$, and $z_0 \in S$ is the start state. A probabilistic FSA when used to generate strings is called a *Markov source*. A Markov source M is *ergodic* if it is irreducible and aperiodic, meaning that each state can reach every other state, and the gcd of the possible recurrence times for each state is 1.

Our main model assumes the source to be a Markov source. A general Markov source models typical object-oriented and hypertext applications well, where the request sequences are generated by traversing links between objects; for example, a hypertext user moves through a document traversing hyperlinks. The Markov source model is not to be confused with a Markov *chain* on the page request sequence, which corresponds to a first-order model. A Markov source can have infinite order and is significantly more general. It has served well in modeling text for data compression purposes, for example.

For such a Markov source M we introduce the notion of *expected fault rate* F_M, a concept intuitively similar to *entropy* in data compression. More specifically, F_M is the best possible expected fault rate achievable by any online prefetching algorithm, even one with full knowledge of the "next state" function and the transition probabilities of the Markov source M. With a slight abuse of notation, we denote by M the best possible prefetching algorithm (which has full knowledge of the Markov source M). When the source is in state z, the optimal

[3] See, for example, Blackwell [1956], Cover and Shenhar [1977], Feder et al. [1992], and Hannan [1957].

prefetcher M puts into cache the k pages having the k maximum probabilities, which minimizes the probability that a page fault will occur during the next page request. This minimum fault probability, weighted by the probability of being in state z, summed over all states z, gives us F_M. This is formalized later in Definition 4.

We adapt a character-by-character version of the Ziv–Lempel [1978] data compressor to get our optimal prefetcher $\mathcal{P}$. Theorems 2.2 and 2.3 below are our main results.

THEOREM 2.2. *Let M be a Markov source. The expected page fault rate achieved by $\mathcal{P}$ approaches F_M, as the page request sequence length $n \to \infty$; the difference is $O(1/\sqrt{\log n})$. If M is ergodic, we get not only convergence of the mean but also convergence almost everywhere: for a finite size dictionary data structure, $\mathcal{P}$'s page fault rate approaches F_M arbitrarily closely for almost all page request sequences σ of length n, as $n \to \infty$.*

We can also show that $\mathcal{P}$ is optimal in the limit, even if we compare $\mathcal{P}$ to the best probabilistic FSA prefetcher tuned individually for each page request sequence σ of length n.

THEOREM 2.3. *Let M be an ergodic Markov source. For any page request sequence σ of length n, let F_σ be the best fault rate achievable by a probabilistic FSA with at most s states applied to σ. For a finite size dictionary data structure, $\mathcal{P}$'s page fault rate approaches F_σ arbitrarily closely for almost all page request sequences σ, as $n \to \infty$.*

The convergences in Theorems 2.2 and 2.3 also hold when we let the number of states s and the alphabet size α get arbitrarily large, as long as n, s, and α tend to ∞ in that relative order.

An interesting case is when the Markov source is stationary, and the start state is chosen randomly according to steady state probabilities of the states. In this case, since the start state is random, it is unclear how a "best algorithm" M (with full knowledge of the source M) would operate! However, since our prefetcher $\mathcal{P}$ is optimal when M knows the start state (which makes M "stronger"), $\mathcal{P}$ is still optimal even when the start state is random.

COROLLARY 2.4. *Theorems* 2.2 *and* 2.3 *hold even when the Markov source M is stationary.*

Our bound on the convergence rate of $\mathcal{P}$ in Theorem 2.2 is slow; the page fault rate of $\mathcal{P}$ approaches optimal as $O(1/\sqrt{\log n})$, where n is the length of the page request sequence. However, the Ziv–Lempel encoder works well in practice for data compression. Purely as a means for comparison, we consider a second less-general model in which the source is assumed to be mth order Markov; in this case, the rate of convergence is provably faster. In such sources, the probability distribution of the next page request (or character) is dependent only on the previous m page requests (characters). We can therefore build a finite state machine for the source, the states labeled by m-contexts and the transitions denoting the changes from one m-context to the next. The state structure of the source is hence known. In this situation, we develop a simple algorithm $\mathcal{M}$ that collects statistics on the transitions that converge exponentially fast to the actual transition probabilities. The fault rate of $\mathcal{M}$ converges polynomially fast to the expected fault rate of the source.

THEOREM 2.5. *If the source is an ergodic mth order Markov source, then the expected page fault rate of $\mathcal{M}$ is within an additive factor of $O(1/\sqrt{n})$ from the expected page fault rate of the source, where n is the length of the page request sequence.*

The difficulty of the main model (Theorems 2.2 and 2.3) as opposed to the mth order model (Theorem 2.5) is that the state structure of the source is unknown in the main model and the problem of prefetching is thus significantly harder than simply estimating transition probabilities.

3. *A Prefetching Algorithm Based on Ziv–Lempel*

In this section, we develop our prefetching algorithm $\mathcal{P}$ based on a character-based version $\mathcal{E}$ of the Ziv–Lempel algorithm for data compression. The original Ziv–Lempel algorithm [Ziv and Lempel 1978] is a word-based data compression algorithm. The Ziv–Lempel encoder breaks the input string into blocks of relatively large length n, and it encodes these blocks using a block-to-variable code. Let x_0 be the empty string λ. The encoder parses each block of size n in a greedy manner into distinct substrings $x_1, x_2, \ldots, x_c$ with the following property: For each $j \geq 1$, substring x_j without its last character is equal to some previous substring x_i, where $0 \leq i < j$. Substring x_j is encoded by the value i, using $\lceil \lg j \rceil$ bits, followed by the ASCII encoding of the last character of x_j, using $\lceil \lg \alpha \rceil$ bits.

Arithmetic coding [Howard and Vitter 1992; Langdon 1984; Witten et al. 1987] is a coding technique that achieves a coding length equal to the entropy of the data model. Sequences of probability p are encoded using $\lg(1/p) = -\lg p$ bits. Arithmetic coding can be thought of as using "fractional" bits, as opposed to the suboptimal Huffman coding in which all code lengths must be integral. The Ziv–Lempel encoder can be converted from a word-based method to a character-based algorithm $\mathcal{E}$ by building a probabilistic model that feeds probability information to an arithmetic coder [Bell et al. 1990; Langdon 1983], as explained in the example below. It has been shown that the coding length obtained in this character-based approach is at least as good as that obtained using the word-based approach [Bell et al. 1990; Howard and Vitter 1992; Langdon 1983]. Hence, the optimality results in Ziv and Lempel [1978] hold without change for the character-based approach.

Example 3.1. Assume for simplicity that our alphabet is $\{a, b\}$. We consider the page request sequence "$aaaababaabbbabaa \cdots$." The Ziv–Lempel encoder parses this string as "$(a)(aa)(ab)(aba)(abb)(b)(abaa) \cdots$." Each substring in the parse is encoded as a pointer followed by an ASCII character. In particular, the match "aba" of the seventh substring "$abaa$" is encoded using $\lceil \lg 6 \rceil$ bits with a value 4, since the match "aba" is the fourth substring, and the last character "a" is encoded using $\lceil \lg 2 \rceil$ bits, since the alphabet size α is 2.

In the character-based version $\mathcal{E}$ of the Ziv–Lempel encoder, a probabilistic model (or parse tree) is built for each substring when the previous substring ends. The parse tree at the start of the seventh substring is pictured in Figure 1. There are five previous substrings beginning with an "a" and one beginning with

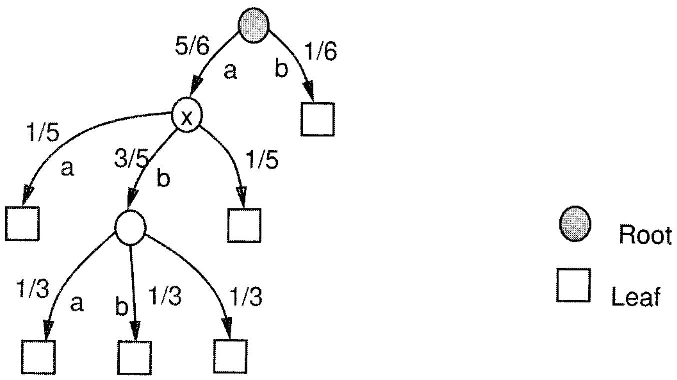

FIG. 1. The parse tree constructed by the character-based encoder $\mathcal{E}$ for Example 3.1.

a "*b*." The page "*a*" is therefore assigned a probability of 5/6 at the root, and "*b*" is assigned a probability of 1/6 at the root. Similarly, of the 5 substrings that begin with an "*a*," one begins with an "*aa*" and three begin with an "*ab*," accounting for the probabilities of 1/5 for "*a*" and 3/5 for "*b*" at node x, and so on. Any sequence that leads from the root of the model to a leaf traverses a sequence of probabilities $p_1, p_2, p_3, \ldots$ whose product $\Pi_i \, p_i$ equals 1/6. The arithmetic coder encodes the sequence with $\;_i \lg(1/p_i) = \lg(1/\Pi_i \, p_i) = \lg 6$ bits. Note that the unnamed transition with probability 1/5 at node x will never be traversed, and having this arc can only increase the code length (since this probability of 1/5 could otherwise be distributed at node x between "*a*" and "*b*" increasing their respective probabilities and reducing their code lengths). Hence, the encoding length using the character-based approach $\mathcal{E}$ will be at least as good as that obtained using the word-based Ziv–Lempel algorithm. □

Our prefetcher $\mathcal{P}$ is based on the character-based version $\mathcal{E}$ of the Ziv–Lempel encoder as follows: At the start of each substring, $\mathcal{P}$'s current node is set to be the root of $\mathcal{E}$'s parse tree. (See Figure 1.) Before each page request, $\mathcal{P}$ prefetches the pages with the top k estimated probabilities as specified by the transitions out of its current node. On seeing the actual page requested, $\mathcal{P}$ resets its current node by walking down the transition labeled by that page and gets ready to prefetch again. In addition, if the page is not in memory, a page fault is generated. When $\mathcal{P}$ reaches a leaf, it fetches in k pages at random. (Our analysis of the optimality of prefetcher $\mathcal{P}$ presented in Section 4 is independent of the strategy used for prefetching while at a leaf. In particular, not prefetching while at a leaf, or using the statistics at the root for prefetching while at a leaf which is comparable to the Welsh variation of the Ziv–Lempel algorithm for data compression, are valid variants of prefetcher $\mathcal{P}$.) The next page request ends the substring, and $\mathcal{P}$ resets its current node to be the root. Updating the model can be done dynamically while $\mathcal{P}$ traverses it. At the end of n page requests, for some appropriately large n, $\mathcal{P}$ throws away its model and starts afresh.

A pseudocode description of prefetcher $\mathcal{P}$ is given in Figure 2. The variable n in the description of algorithm $\mathcal{P}$ in Figure 2 is a user-specified parameter that determines the data structure restart time.

```
program $\mathcal{P}$;
begin
    while not end of user session do
    begin
        initialize data structure tree to be a single node (the root);
        set current_node to be the  root;
        number_of_requests := 0;
        while number_of_requests < $n$ and not end of user session c
        begin
            pure_prefetch( );
            let $r$ be the next user page request;
            update_model($r$);
            number_of_requests := number_of_requests + 1
        end
    end
end;


procedure pure_prefetch( );
begin
    $d$ := number of transitions out of current_node;
    if current_node is not a leaf then
        prefetch the $\min\{k, d\}$ pages corresponding to the transitions
          out of current_node that have the highest $\min\{k, d\}$ counts
    endif;
    prefetch $k - \min\{k, d\}$ more pages at random;
    replace the pages in cache with the $k$ prefetched pages
end;


procedure update_model($r$);
begin
    if there exists a transition $t$ labeled $r$ out of current_node then
        increment count for transition $t$;
        set current_node to be the node pointed to by transition $t$
    else
        add a transition $t$ with label $r$ out of current_node;
        set count of transition $t$ to be 1;
        set current_node to be the root
    endif
end;
```

FIG. 2. Algorithm $\mathcal{P}$. Pseudocode for pure prefetching.

4. *Analysis of our Prefetching Algorithm*

Our analysis of the fault rate achieved by our prefetcher $\mathcal{P}$ builds on an analysis of the compression rate achieved by the Ziv–Lempel character-based encoder $\mathcal{E}$ that $\mathcal{P}$ is based on. In Section 4.1 we show that $\mathcal{E}$ is optimal in terms of compression rate for almost all strings emitted by a Markov source. In Section 4.2, we build on Section 4.1 and show $\mathcal{P}$ is optimal in terms of prefetching for almost all strings emitted by a Markov source.

4.1. BOUNDS ON COMPRESSION. We let σ denote a (possibly infinite) sequence from the alphabet A, and we use the notation σ_i^j to denote the subsequence of σ starting at the ith character up to and including the jth character; in particular σ_1^n denotes the first n characters of σ. For convenience, we use $p_{z,i}$ to denote $p(z, i)$, $g(z, \sigma_1^n)$ to denote the state reached by a probabilistic FSA when processing string σ_1^n starting in state z, and $\Pr(z, \ell)$ to denote the probability that the source M is in state z after emitting ℓ characters.

Definition 4.1.1 [*Gallager* 1968]. Let M be a Markov source. The best possible average encoding length per character of M for input sequences of length n is given by

$$H_M(n) = \frac{1}{n} \sum_{z \in S} \left(\sum_{\ell=0}^{n-1} \Pr(z, \ell) \right) \left(\sum_i p_{z,i} \lg \frac{1}{p_{z,i}} \right).$$

If we take the limit of $H_M(n)$ as $n \to \infty$, we get the *entropy* H_M of M.

We now examine the performance of the Ziv–Lempel character-based encoder $\mathscr{E}$ under our probabilistic model of sources and coders. Note that an arithmetic coder can use a probabilistic FSA as a model to perform data compression, and hence probabilistic FSAs can be considered as encoders.

Definition 4.1.2. Given an encoder C, we define C's *compression rate* (or number of output bits per character) of σ_1^n by

$$\mathit{Compression}_{C,n}(\sigma_1^n) = \frac{L(y_1^n)}{n}, \tag{1}$$

where $L(y_1^n)$ is the length of C's encoding of σ_1^n. Let $M(s)$ be the set of all probabilistic FSAs with $|A| = \alpha$ and $|S| \le s$. We define $\mathit{Compression}_{M(s),n}(\sigma_1^n)$ to be $\min_{C \in M(s)}\{\mathit{Compression}_{C,n}(\sigma_1^n)\}$.

In particular, if we use a Markov source M to generate the sequence σ_1^n and also to encode σ_1^n (via arithmetic coding), the average compression rate achieved is equal to the entropy of M; that is,

$$E(\mathit{Compression}_{M,n}) = H_M(n). \tag{2}$$

The definitions in Definition 4.1.2 above are similar to those of Ziv and Lempel [1978], except that they define $M(s)$ to be a class of "information lossless" nonprobabilistic FSA encoders, use ρ in place of *Compression*, and use $n \lg \alpha$ in place of n in (1) to get a ratio of output length to input length.

We generalize Ziv and Lempel's main result [Ziv and Lempel 1978] to our model $M(s)$ of probabilistic FSAs, using an iterative analysis based on arithmetic coding, to get the following theorem:

THEOREM 4.1.3. *The compression rate of $\mathscr{E}$ on σ_1^n is no worse than the best probabilistic FSA in the limit as $n \to \infty$. In particular,*

$$\mathit{Compression}_{\mathscr{E},n}(\sigma_1^n) \le \mathit{Compression}_{M(s),n}(\sigma_1^n) + \delta(n),$$

$$\textit{where} \quad \delta(n) = O\left(\frac{1}{\log n}\right).$$

To prove Theorem 4.1.3, we first prove some facts about arithmetic coding and the way the character-based encoder $\mathscr{E}$ works. The first step is to show a lower bound on $Compression_{M(s),n}(\sigma_1^n)$, for which we need the following lemmas.

LEMMA 4.1.4. *Suppose that σ_1^n can be parsed into c "prefix-free" substrings, that is, where no substring is a prefix of another. Then we have*

$$Compression_{M(s),n}(\sigma_1^n) \geq \frac{c}{n}\, lg\, \frac{c}{s^2}.$$

PROOF. The working of an arithmetic coder using a probabilistic FSA as its model can be explained in the following way: The arithmetic coder associates a unit interval [0, 1] with each state of the model. This interval is partitioned into distinct subintervals, one per character, the size of the subinterval associated with a character proportional to the probability of its transition out of the state. Any string that takes the coder from state x to state x' of the model defines implicitly a subinterval of the original [0, 1] interval at x; also, this subinterval uniquely characterizes the string. The size of this subinterval is clearly the product of the probabilities of the transitions taken by the arithmetic coder while processing the string and all the arithmetic coder's output has to do is to identify this subinterval. To identify a subinterval of length u, an arithmetic coder has to output at least $\lg(1/u) = -\lg u$ bits; for more details, see Howard and Vitter [1992], Langdon [1984], and Witten et al. [1987]. As an example, consider the probabilistic FSA of Figure 1 being used as a model by an arithmetic coder. Starting at node x, the string "*ba*" would cause the interval to shrink by a factor of 3/5 from [0, 1] to [0.2, 0.8] and then by a factor of 1/3 from [0.2, 0.8] to [0.2, 0.4]. To identify "*ba*," which is associated with the interval [0.2, 0.4], the arithmetic coder needs to output at least $-\lg(0.4 - 0.2)$ bits (which is the same as $-\lg((3/5)(1/3))$).

It is clear that if there are c' distinct substrings processed in which M starts from state x and ends in state x', these c' substrings define c' distinct subintervals of the original [0, 1] interval at x. If these c' substrings are such that no one is a prefix of another, the subintervals corresponding to them are nonoverlapping, and the sum of the lengths of these subintervals is at most 1. By convexity arguments, for these c' substrings, the arithmetic coder has to output an average of at least $\lg c'$ bits per substring, giving a total output length of at least $c' \lg c'$ bits for these substrings.

To calculate the output length of M on σ_1^n, we can trace σ_1^n through M and sum up the output lengths for each substring in the parsing of σ_1^n. If σ_1^n can be parsed into c distinct substrings, no one being a prefix of another, by convexity arguments the code length is minimized when the c substrings are distributed equally over the s^2 state pairs (x, x'), where x is the state of M when the substring is about to be processed and x' is the state of M after the substring is processed. Substituting $c' = c/s^2$ gives us the desired bound. □

The internal path length of a tree is defined as the sum over all nodes of the length of the path from the root to that node. For a parse tree, the internal path length is the length of the original string. The branch factor of a tree is the maximum number of children that any node of the tree can have. For a parse

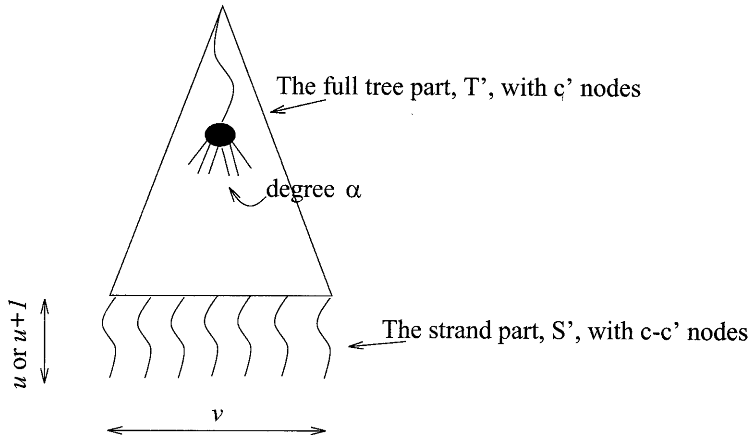
The full tree part, T', with c' nodes
degree α
u or u+1
The strand part, S', with c-c' nodes
v

FIG. 3. Structure of the tree for minimum internal path length, used in the proof of Lemma 4.1.5.

tree, the branch factor is the alphabet size α. We now show a lower bound on the number of leaves in a general tree.

LEMMA 4.1.5. *Any tree with c nodes and internal path length n has at least $c^2/20n$ leaves.*

PROOF. Let the branch factor of the tree be denoted by α. The case $\alpha = 1$ is trivial, so let us assume that $\alpha \geq 2$. We call the number v *feasible* if there exists a tree with v leaves that has at least c nodes and internal path length at most n. For any feasible v, if we create a tree T with at least c nodes and v leaves and with minimum internal path length, then T's internal path length must be at most n. This tree T must accommodate as many nodes as possible close to its root so as to minimize its internal path length. Hence, T must consist of a "full tree part" T' sitting above a "strand part" S'. (See Figure 3.)

The full tree part T' has c' nodes and v leaves, where every internal node, except possibly one, has α children. It follows that T' has at most $c' \leq v\alpha/(\alpha - 1) \leq 2v$ nodes. The strand part S' has the remaining $c - c'$ nodes distributed as v strands, each strand of length u or $u + 1$ and hanging from a leaf of T'. The number c of nodes in T is the sum of the number of nodes in T' and S', which is bounded by $2v + v(u + 1) \leq vu + 3v$. Hence, we have

$$vu + 3v \geq c. \tag{3}$$

The contribution of S' to the internal path length of T is at least $vu^2/2$, so we have $vu^2/2 \leq n$, which implies that $vu \leq \sqrt{2nv}$. Since $v \leq n$, we have $3v \leq 3\sqrt{nv}$. Substituting these bounds into (3), we get $\sqrt{nv}(\sqrt{2} + 3) \geq c$, from which it follows that $v \geq c^2/20n$. □

When the Ziv–Lempel encoder parses a string σ_1^n, it breaks up this string into distinct substrings that are closed under the prefix operation; that is, if σ' is one of the substrings in the parse, then every prefix of σ' is also one of the substrings in the parse. Thus, the substrings are not "prefix-free" and Lemma 4.1.4 cannot be applied directly to get a lower bound on $Compression_{M(s),n}(\sigma_1^n)$. The substrings in the parse can be denoted by a parse tree, like the one pictured in Figure 1. The nodes of the parse tree correspond to the substrings in the parse, and node i is a child of node j via the edge labeled "a" if substring j appended with character "a" gives substring i. For two nodes in a parse tree, if neither is an ancestor of the other, then neither can be a prefix of the other; in particular, the leaves of a parse tree are "prefix-free." In Lemma 4.1.6 below, we derive a lower bound on the compression rate of σ_1^n for any finite state encoder by applying Lemma 4.1.4 to the (prefix-free) leaves of the parse tree, stripping the leaves away, then applying Lemma 4.1.4 to the (prefix-free) leaves of what remains, stripping the leaves away, and so on. Lemma 4.1.5 ensures the number of leaves at each stage is sufficient to get the desired bound.

LEMMA 4.1.6. *For any string σ_1^n, we have*

$$\mathit{Compression}_{M(s),n}(\sigma_1^n) \geq \frac{1}{n}\,(2c(\sigma_1^n)\lg c(\sigma_1^n) - c(\sigma_1^n)\lg n$$

$$- c(\sigma_1^n)\lg s^2 - c(\sigma_1^n)\lg(20(2\alpha + 2)^4)),$$

where $c(\sigma_1^n)$ is the maximum number of nodes in any parse tree for σ_1^n.

PROOF. Consider a parse of σ_1^n into c distinct prefix-closed substrings. The corresponding parse tree for σ_1^n has c nodes. Recall that the v leaves of this tree (which are substrings of σ_1^n) are prefix-free; that is, no leaf is a prefix of another. By Lemma 4.1.4, any encoder requires at least $v\ \lg(v/s^2)$ bits to encode these leaves (substrings). We can strip off this layer of leaves from the tree and iteratively lowerbound the encoding length for the remaining nodes of the tree.

To analyze this iterative process, we consider the stripping procedure to work in phases. Each phase involves the stripping of one or more complete layers of leaves. For $1 \leq i \leq r$, the ith phase ends when the number of nodes remaining in the tree is $c_i \leq c_{i-1}/2$. By definition, we have $c_0 = c$ and $c_r = 0$. Let the number of leaves at the end of the ith phase be v_i. If we denote by $v_{i,j}$ the number of leaves removed in the jth layer of stripping within phase i, then by Lemma 4.1.4, the encoding length by any finite state encoder of the nodes stripped off in the ith phase is at least

$$\sum_j v_{i,j} \lg \frac{v_{i,j}}{s^2} \geq \sum_j v_{i,j} \lg \frac{v_i}{s^2} = (c_{i-1} - c_i)\lg \frac{v_i}{s^2}.$$

By Lemma 4.1.5, we have $v_i \geq c_i^2/20n$. Hence,

$$\mathit{Compression}_{M(s),n}(\sigma_1^n) \geq \frac{1}{n}\sum_{i=1}^{r}(c_{i-1} - c_i)\lg \frac{v_i}{s^2}$$

$$\geq \frac{1}{n}\sum_{i=1}^{r}(c_{i-1} - c_i)\lg c_i^2 - \frac{\lg 20ns^2}{n}\sum_{i=1}^{r}(c_{i-1} - c_i). \qquad (4)$$

By the definition of when a phase ends, we have $c_{i-1} - c_i \geq c_{i-1}/2$. Since the branch factor of the parse tree is the alphabet size α, we get the upper bound $c_{i-1} - c_i \leq c_{i-1}/2 + c_i\alpha$. This gives us

$$c_i \geq \frac{c_0}{(2\alpha + 2)^i}.$$

Hence, we have

$$\sum_{i=1}^{r} (c_{i-1} - c_i)\lg c_i^2 \geq 2 \sum_{i=1}^{r} (c_{i-1} - c_i)\lg c_0 - 2\lg(2\alpha + 2) \sum_{i=1}^{r} i(c_{i-1} - c_i)$$

$$\geq 2c \lg c - 2\lg(2\alpha + 2) \cdot (2c), \tag{5}$$

from simple telescoping. Substituting (5) into (4), we get

$$\mathit{Compression}_{M(s),n}(\sigma_1^n) \geq \frac{1}{n}(2c \lg c - c \lg n - c \lg s^2 - c \lg(20(2\alpha + 2)^4)). \tag{6}$$

Since (6) is true for any c, it is true when $c = c(\sigma_1^n)$, the maximum number of nodes in any parse tree for σ_1^n. □

We are now ready to present the proof of Theorem 4.1.3.

PROOF OF THEOREM 4.1.3. It has been shown in Ziv and Lempel [1978] that

$$\mathit{Compression}_{\mathscr{E},n}(\sigma_1^n) \leq \frac{c(\sigma_1^n) + 1}{n} \lg(2\alpha(c(\sigma_1^n) + 1)), \tag{7}$$

where $c(\sigma_1^n)$ is the maximum number of nodes in any parse tree[4] for σ_1^n. It is shown in Lempel and Ziv [1976] that

$$0 \leq c(\sigma_1^n) < \frac{n \lg \alpha}{(1 - \epsilon_n)\lg n}, \qquad \text{where} \quad \lim_{n\to\infty} \epsilon_n = 0. \tag{8}$$

Theorem 4.1.3 is clearly true when $c(\sigma_1^n) = o(n/\lg n)$ since $\mathit{Compression}_{\mathscr{E},n}(\sigma_1^n) \sim 0$ as $n \to \infty$. When $c(\sigma_1^n) = O(n/\lg n)$, using the lower bound for $\mathit{Compression}_{M(s),n}(\sigma_1^n)$ from Lemma 4.1.6 and the upper bound for $\mathit{Compression}_{\mathscr{E}}(\sigma_1^n)$ from (7), we get by simple arithmetic that

$$\mathit{Compression}_{\mathscr{E},n}(\sigma_1^n) - \mathit{Compression}_{M(s),n}(\sigma_1^n) = O\left(\frac{1}{\log n}\right),$$

for any fixed α and s. By (8), no more possibilities exist for $c(\sigma_1^n)$ and the theorem stands proved. □

If our Markov source M has τ states, it clearly belongs to the set $M(\tau)$, and M compresses no better than the best automaton in $M(s)$, $s \geq \tau$, for all sequences σ_1^n produced by it. Using this fact in Theorem 4.1.3, taking the expected value of both sides of the inequality, and using expression (2), we get the following

[4] This is not the definition of $c(\sigma_1^n)$ in Ziv and Lempel [1978], but it is easy to verify that the proofs in Ziv and Lempel [1978] also hold under this definition of $c(\sigma_1^n)$.

corollary that the encoder $\mathscr{E}$ compresses as well as possible, achieving the entropy of the source M in the limit.

COROLLARY 4.1.6. *Let M be a Markov source with s states. We have*

$$E(Compression_{\mathscr{E},n}) \leq H_M(n) + \delta'(n), \qquad \textit{where} \quad \delta'(n) = O\left(\frac{1}{\log n}\right).$$

4.2. BOUNDS ON FAULT RATE. Along the lines of entropy in Definition 4.1.1, we introduce the corresponding notion of the *expected fault rate* F_M of a Markov source M. It is the expected fault rate achieved in prefetching by the best algorithm that fully knows the source. As mentioned before, with a slight abuse of notation, we denote this best algorithm also by M. When the source is in some state z, algorithm M puts into the cache those k pages having the maximum probabilities for state z.

Definition 4.2.1. Let M be a Markov source. Let $K_z(M)$ be a set of pages with the maximum k probabilities at state z. Then the expected fault rate of M on inputs of length n is defined by

$$F_M(n) = \frac{1}{n} \sum_{z \in S} \left(\sum_{v=0}^{n-1} \Pr(z, v) \right) \left(\sum_{i \notin K_z(M)} p_{z,i} \right).$$

If we take the limit of $F_M(n)$ as $n \to \infty$, we get the *expected fault rate* F_M of M.

We now come to our goal: to show optimality of our prefetcher $\mathscr{P}$. The challenge is to show the correspondence between converging to the entropy and converging to the page fault rate.

Definition 4.2.2. Given a Markov source M and a sequence σ generated by M, we define the *fault rate* $Fault_{P,n}(\sigma_1^n)$ of prefetcher P to be the number of page faults incurred by P on σ_1^n, divided by n.

It is easy to prove the following lemma that M (considered as a prefetcher) has the best expected fault rate (namely, F_M) among all prefetchers when the source is M (considered as a Markov source).

LEMMA 4.2.3. *Let M be a Markov source. The expected fault rate of any* (*deterministic or randomized*) *online algorithm P on sequences of length n satisfies*

$$E(Fault_{P,n}) \geq F_M(n).$$

Our first task is to show the following important theorem that the expected fault rate of $\mathscr{P}$ is no worse than the best possible expected fault rate F_M on sequences of length n, as $n \to \infty$. This restates in detail the first part of our first main theorem (Theorem 2.2).

THEOREM 4.2.4. *Let M be a Markov source. The expected page fault rate of $\mathscr{P}$ on sequences of length n is no worse than the expected page fault rate of the Markov source in the limit as $n \to \infty$. In particular,*

$$E(Fault_{\mathscr{P},n}) \leq F_M(n) + \epsilon(n), \qquad \textit{where} \quad \epsilon(n) = O\left(\frac{1}{\sqrt{log\ n}}\right).$$

To prove the above theorem, we use the following lemmas. The first gives a bound on the probability of page fault by $\mathcal{P}$ that is independent of the cache size k.

LEMMA 4.2.5. *Suppose that at time instant θ the Markov source M is at state z and the next page request is i with probability p_i. We can think of M as a prefetching algorithm with access to the probabilities p_i. Suppose that our prefetcher $\mathcal{P}$ thinks that the next page request will be i with probability r_i. Let us denote the probability of page fault by M and $\mathcal{P}$ on the next page request by f_M and $f_{\mathcal{P}}$, respectively. We have, independently of the cache size k,*

$$f_{\mathcal{P}} - f_M \leq \sum_{i=1}^{\alpha} |p_i - r_i|.$$

PROOF. Let A be the set of k pages that M chooses to put in cache, let B be the set of k pages that $\mathcal{P}$ chooses to put in cache, and let $C = A \cap B$. For any set X of pages, let $p_X = \quad {}_{i \in X} p_i$ and let $r_X = \quad {}_{i \in X} r_i$. Then, $f_{\mathcal{P}} - f_M = (1 - p_B) - (1 - p_A) = p_A - p_B = p_{A-C} - p_{B-C}$. Let $\quad {}_{i=1}^{\alpha} |p_i - r_i| = \epsilon$. Then, $p_{A-C} = r_{A-C} + \epsilon_1$ and $p_{B-C} = r_{B-C} + \epsilon_2$, where $|\epsilon_1| + |\epsilon_2| \leq \epsilon$. Since $\mathcal{P}$ chooses those k pages that have the top k probabilities amongst the r_i, $1 \leq i \leq \alpha$, we have $r_{B-C} \geq r_{A-C}$. Hence, $f_{\mathcal{P}} - f_M = p_{A-C} - p_{B-C} \leq \epsilon_1 + \epsilon_2 \leq |\epsilon_1| + |\epsilon_2| \leq \epsilon$. □

The following lemma is well known; however, we reproduce it and its proof from Amit and Miller [1990] here for the sake of completeness. The summation on the right-hand side of the lemma is the Kullback–Leibler divergence of $(r_1, \ldots, r_\alpha)$ with respect to $(p_1, \ldots, p_\alpha)$.

LEMMA 4.2.6. *Given two probability vectors $(p_1, \ldots, p_\alpha)$ and $(r_1, \ldots, r_\alpha)$, we have*

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

PROOF. From the fact that $x \ln x - x + 1 \geq 0$ and that $3(x-1)^2 \leq (4x+2)(x \ln x - x + 1)$, we get $|x - 1| \leq \sqrt{(4x+2)/3}\,\sqrt{x \ln x - x + 1}$. Using this inequality at each term of the left-hand side of the inequality to be proved, and applying the Cauchy–Schwartz inequality, we obtain

$$\begin{aligned}
\left(\sum_{i=1}^{\alpha} \left| p_i - r_i \right| \right)^2 &= \left(\sum_{i=1}^{\alpha} r_i \left| \frac{p_i}{r_i} - 1 \right| \right)^2 \\
&\leq \frac{1}{3} \left(\sum_{i=1}^{\alpha} \left(4 \frac{p_i}{r_i} + 2 \right) r_i \right) \cdot \left(\sum_{i=1}^{\alpha} \left(\frac{p_1}{r_i} \lg \frac{p_i}{r_i} - \frac{p_i}{r_i} + 1 \right) r_i \right) \\
&= 2 \sum_{i=1}^{\alpha} p_i \ln \frac{p_i}{r_i}.
\end{aligned}$$

The last step above follows from the fact that $\quad {}_{i=1}^{\alpha} p_i = \quad {}_{i=1}^{\alpha} r_i = 1$. □

LEMMA 4.2.7. *Let* ${}_{i=1}^{u} \gamma_i x_i$ *be a convex linear combination of the nonnegative quantities* x_i; *that is*, $\gamma_i \geq 0$ *and* ${}_{i=1}^{u} \gamma_i = 1$. *Then*,

$$\left(\sum_{i=1}^{u} \gamma_i \sqrt{x_i} \right)^2 \leq \sum_{i=1}^{u} \gamma_i x_i .$$

PROOF. The lemma follows from the square root function being concave. □

We are now ready to prove Theorem 4.2.4.

PROOF OF THEOREM 4.2.4. The basic idea of the proof is to examine the behavior of the prefetcher $\mathcal{P}$ whenever the Markov source M is in a particular state z. One big difficulty is coping with the fact that the optimal prefetcher M always prefetches the same k pages at state z, whereas our prefetcher $\mathcal{P}$'s probabilities may differ significantly each time M is in state z, since it is context-dependent. To get over this problem, we map the differences over contexts to the state z and weight by the probability of being in state z.

Let z_0 be the start state and S be the set of states of the source M. In the definition of $H_M(n)$ (Definition 4.1.1), note that $\Pr(z, \ell)$ is just the sum of the probabilities of all length ℓ strings that bring M from z_0 to z. Hence[5]

$$H_M(n) = \frac{1}{n} \sum_{z \in S} \sum_{\ell=0}^{n-1} \sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell) \cdot [g(z_0, \sigma_1^\ell) = z] \cdot H_M(z)$$

$$= \frac{1}{n} \sum_{z \in S} \sum_{\ell=0}^{n-1} \sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell) \cdot [g(z_0, \sigma_1^\ell) = z] \cdot \sum_{i=1}^{\alpha} p_{z,i} \lg \frac{1}{p_{z,i}}, \qquad (9)$$

where $\Pr(\sigma_1^0) = 1$, $g(z_0, \sigma_1^0) = z_0$, and $H_M(z)$ is the average encoding length of the source at state z and is equal to ${}_{i=1}^{\alpha} p_{z,i} \lg(1/p_{z,i})$.

Let $\mathit{Compression}_{\mathscr{E}}^{\ell+1}(\sigma_1^\ell, z, \sigma_{\ell+1}^{\ell+1})$ be $\mathscr{E}$'s encoding length for the $(\ell + 1)$st character $\sigma_{\ell+1}^{\ell+1}$, given that M is in state z after emitting the first ℓ characters σ_1^ℓ. We have $\mathit{Compression}_{\mathscr{E},n}(\sigma_1^n) = (1/n) \ {}_{\ell=0}^{n-1} \mathit{Compression}_{\mathscr{E}}^{\ell+1}(\sigma_1^\ell, z, \sigma_{\ell+1}^{\ell+1})$. We would like to express $E(\mathit{Compression}_{\mathscr{E},n})$ in a form similar to (9). If we specify the first ℓ characters and leave the $(\ell + 1)$st character $\sigma_{\ell+1}^{\ell+1}$ unspecified, we get the random variable $\mathit{Compression}_{\mathscr{E}}^{\ell+1}(\sigma_1^\ell, z)$ with mean ${}_{i=1}^{\alpha} p_{z,i} \lg(1/r_{\sigma_1^\ell,i})$, where $r_{\sigma_1^\ell,i} > 0$ is the probability with which $\mathscr{E}$ expects to see character i next after having processed σ_1^ℓ in which M ends up in state z. We have

$$E(\mathit{Compression}_{\mathscr{E}}^{\ell+1}) = \sum_{z \in S} \sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell) \cdot [g(z_0, \sigma_1^\ell) = z] \cdot E(\mathit{Compression}_{\mathscr{E}}^{\ell+1}(\sigma_1^\ell, z))$$

$$= \sum_{z \in S} \sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell) \cdot [g(z_0, \sigma_1^\ell) = z] \cdot \sum_{i=1}^{\alpha} p_{z,i} \lg \frac{1}{r_{\sigma_1^\ell,i}} . \qquad (10)$$

Summing on ℓ in (10), we get

[5] We use the notation [*relation*] to denote 1 if *relation* is true and 0 if *relation* is false.

$$E(\mathit{Compression}_{\mathcal{C},n}) = \frac{1}{n} \sum_{\ell=0}^{n-1} E(\mathit{Compression}_{\mathcal{C}}^{\ell+1})$$

$$= \frac{1}{n} \sum_{z \in S} \sum_{\ell=0}^{n-1} \sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell) \cdot [g(z_0, \sigma_1^\ell) = z] \cdot \sum_{i=1}^{\alpha} p_{z,i} \lg \frac{1}{r_{\sigma_1^\ell, i}}. \tag{11}$$

Combining (11) and (9), we get

$$E(\mathit{Compression}_{\mathcal{C},n}) - H_M(n)$$

$$= \frac{1}{n} \sum_{z \in S} \sum_{\ell=0}^{n-1} \sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell) \cdot [g(z_0, \sigma_1^\ell) = z] \cdot \sum_{i=1}^{\alpha} p_{z,i} \lg \frac{p_{z,i}}{r_{\sigma_1^\ell, i}}. \tag{12}$$

Our goal is to express the quantity $E(\mathit{Fault}_{\mathcal{P},n}) - F_M(n)$ in a way similar to (12). By analogy to the expression (9) for $H_M(n)$, we have

$$F_M(n) = \frac{1}{n} \sum_{z \in S} \sum_{\ell=0}^{n-1} \sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell) \cdot [g(z_0, \sigma_1^\ell) = z] \cdot F_M(z)$$

$$= \frac{1}{n} \sum_{z \in S} \sum_{\ell=0}^{n-1} \sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell) \cdot [g(z_0, \sigma_1^\ell) = z] \cdot \sum_{i \notin K_z(M)} p_{z,i}, \tag{13}$$

where $F_M(z)$ is the expected page fault rate at state z of M, and $K_z(M)$ is a set of pages with the maximum k probabilities at state z.

By further analogy, we define $\mathit{Fault}_{\mathcal{P}}^{\ell+1}(\sigma_1^\ell, z, \sigma_{\ell+1}^{\ell+1})$ be the 0–1 quantity denoting whether $\mathcal{P}$ faults on the $(\ell + 1)$st page request $\sigma_{\ell+1}^{\ell+1}$, given that M is in state z after emitting the first ℓ page requests σ_1^ℓ. We have $\mathit{Fault}_{\mathcal{P},n}(\sigma_1^n) = (1/n)$ ${}_{\ell=0}^{n-1} \mathit{Fault}_{\mathcal{P}}^{\ell+1}(\sigma_1^\ell, z, \sigma_{\ell+1}^{\ell+1})$. If we specify the first ℓ page requests and leave the $(\ell + 1)$st page request $\sigma_{\ell+1}^{\ell+1}$ unspecified, we get the random variable $\mathit{Fault}_{\mathcal{P}}^{\ell+1}(\sigma_1^\ell, z)$ with mean ${}_{i \notin K_z(\mathcal{P}, \sigma_1^\ell)} p_{z,i}$, where $K_z(\mathcal{P}, \sigma_1^\ell)$ is the set of k pages that $\mathcal{P}$ puts into its cache after processing σ_1^ℓ in which M ends up in state z. We have

$$E(\mathit{Fault}_{\mathcal{P}}^{\ell+1}) = \sum_{z \in S} \sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell) \cdot [g(z_0, \sigma_1^\ell) = z] \cdot E(\mathit{Fault}_{\mathcal{P}}^{\ell+1}(\sigma_1^\ell, z))$$

$$= \sum_{z \in S} \sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell) \cdot [g(z_0, \sigma_1^\ell) = z] \cdot \sum_{i \notin K_z(\mathcal{P}, \sigma_1^\ell)} p_{z,i}. \tag{14}$$

Summing on ℓ in (14), we get

$$E(\mathit{Fault}_{\mathcal{P},n}) = \frac{1}{n} \sum_{\ell=0}^{n-1} E(\mathit{Fault}_{\mathcal{P}}^{\ell+1})$$

$$= \frac{1}{n} \sum_{z \in S} \sum_{\ell=0}^{n-1} \sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell) \cdot [g(z_0, \sigma_1^\ell) = z] \cdot \sum_{i \notin K_z(\mathcal{P}, \sigma_1^\ell)} p_{z,i}. \tag{15}$$

Combining (13) and (15), we get

$$E(Fault_{\mathcal{P},n}) - F_M(n)$$

$$= \frac{1}{n}\sum_{z\in S}\sum_{\ell=0}^{n-1}\sum_{\text{all }\sigma_1^\ell}\Pr(\sigma_1^\ell)\cdot[g(z_0,\sigma_1^\ell)=z]\cdot\left(\sum_{i\notin K_z(\mathcal{P},\sigma_1^\ell)}p_{z,i}-\sum_{i\notin K_z(M)}p_{z,i}\right). \tag{16}$$

From Lemma 4.2.5 and (16) we get

$$E(Fault_{\mathcal{P},n}) - F_M(n)$$

$$\leq \frac{1}{n}\sum_{z\in S}\sum_{\ell=0}^{n-1}\sum_{\text{all }\sigma_1^\ell}\Pr(\sigma_1^\ell)\cdot[g(z_0,\sigma_1^\ell)=z]\cdot\sum_{i=1}^{\alpha}|p_{z,i}-r_{\sigma_1^\ell,i}|. \tag{17}$$

Let us denote the term $^{\alpha}_{i=1}\ |p_{z,i} - r_{\sigma_1^\ell,i}|$ in (17) by $\epsilon(z, \ell, \sigma_1^\ell)$, and let us denote the term $^{\alpha}_{i=1}\ p_{z,i}\ \lg(p_{z,i}/r_{\sigma_1^\ell,i})$ in (12) by $\delta(z, \ell, \sigma_1^\ell)$. Lemma 4.2.6 bounds $\epsilon(z, \ell, \sigma_1^\ell)$ by $\sqrt{(\ln 4)\delta(z, \ell, \sigma_1^\ell)}$. By three applications of Lemma 4.2.7 to (12) and (17) and by the bound on $E(Compression_{\mathcal{C},n}) - H_M(n)$ from Corollary 4.1.6, we get our result.

From Corollary 4.1.6 and the above discussion it follows that $\epsilon(n)$ is bounded by $\sqrt{(\ln 4)\delta'(n)} = O(1/\sqrt{\log n})$. □

The following theorem is the detailed version of the second part of our first main theorem (Theorem 2.2):

THEOREM 4.2.8. *Let the page request sequence σ of length bn be generated by an ergodic Markov source M. We have*

$$Fault_{\mathcal{P},nb} \rightarrow E(Fault_{\mathcal{P},n}), \qquad \textit{for} \quad \textit{almost all sequences } \sigma, \textit{ as } b \rightarrow \infty,$$

where, by Theorem 4.2.4, *we have*

$$\lim_{n\rightarrow\infty} E(Fault_{\mathcal{P},n}) = F_M.$$

PROOF. Given a sequence σ_1^{bn}, we divide it into b blocks each of length n. The net fault rate $Fault_{\mathcal{P},nb}(\sigma)$ is $^{b}_{i=1}\ Fault_{\mathcal{P},n}(\sigma_{(i-1)n+1}^{in})/b$. Since we throw away our data structures at the end of each block, each of the b random variables $Fault_{\mathcal{P},n}$, for $1 \leq i \leq b$, depends only on the start state for each block, and our result follows by the ergodic theorem [Gallager 1968]. □

The proof of Theorem 2.3 essentially deals with showing that F_σ converges to F_M for almost all σ as $n \rightarrow \infty$. We call two prefetchers M_1 and M_2 *distinct* if there exists some time instant θ and some page request sequence σ such that M_1 and M_2 prefetch different sets of pages at time θ on σ. Let $M_{opt}(s)$ be a maximal set of probabilistic FSAs with s states that are distinct when considered as prefetchers and that are each optimal for some page request sequence. We now see that $|M_{opt}(s)|$ is finite.

LEMMA 4.2.9. *The cardinality of set $M_{opt}(s)$ is dependent only on s, α, and k, and is independent of n.*

PROOF. Let $M' \in M_{opt}(s)$ be the best prefetcher for some page request sequence σ_1^n. Let us trace σ_1^n through M', counting the number of times each transition is traversed. It is clear that the strategy of M' would be to prefetch at each state those k pages corresponding to the k maximum count transitions out of that state. Hence, M' could be considered as a finite state predictor with at most s states and k transitions out of each state corresponding to the k pages it prefetches at that state. The number of distinct FSAs with at most s states and k transitions out of each state is clearly dependent only on s, α, and k. □

PROOF OF THEOREM 2.3. To prove Theorem 2.3, we need to show that F_σ approaches F_M for almost all page request sequences σ. By Theorem 2.2, we know that the fault rate of $\mathcal{P}$ approaches F_M for almost all page request sequences. Theorem 2.3 then follows by transitivity.

Let $M = (S_m, A, g_m, p_m, z_m)$ be the Markov source, and let $M' = (S_{m'}, A, g_{m'}, p_{m'}, z_{m'}) \in M_{opt}(s)$. Let us define prefetcher $X = (S_x, A, g_x, p_x, z_x)$ to be a type of "cross product" of M and M', where $S_x = S_m \times S_{m'}$, $g_x((z_i, z_j), a) = (g_m(z_i, a), g_{m'}(z_j, a))$, and $p_x((z_i, z_j), a) = p_m(z_i, a)$. At state $(z_i, z_j) \in S_x$, X prefetches those k pages that M prefetches at $z_i \in S_m$, that is, the pages with the top k probabilities at (z_i, z_j).

Let us consider a state $z \in S_x$. Given a sequence σ of length n, let $\hat{p}_z$ be the number of times σ reaches state z, divided by n, and let $\hat{p}_{z,i}$ be the number of times σ takes transition i out of state z, divided by the total number of transitions taken by σ out of state z. From large deviation theory [Shwartz and Weiss 1995], we know that $\Pr(|\hat{p}_z - p_z| > \delta)$ is exponentially small in n for $\delta > 0$, where n is the length of the sequence. Similarly, from Shwartz and Weiss [1995], we see that $\Pr(|\hat{p}_{z,i} - p_{z,i}| > \delta)$ is exponentially small in n. (The exponentially small probability is of the form $O(\exp(-a\delta^2 n))$, where a depends on the Markov source.) Using Lemmas 4.2.5 and 4.2.8, we have

$$E(Fault_{X,n}) - F_\sigma \geq \sqrt{2\delta} \tag{18}$$

with exponentially small probability. By the definition of fault rate and Lemma 4.2.3, it follows that $E(Fault_{X,n}) = F_M(n) \leq E(Fault_{M',n})$. Since by Lemma 4.2.9 $|M_{opt}(s)|$ is finite, it follows for any $\epsilon > 0$ that

$$F_M(n) - F_\sigma \geq \epsilon$$

with exponentially small probability. Thus, F_σ converges to F_M for almost all page request sequences σ.

From Theorem 2.2, $\mathcal{P}$'s fault rate converges to F_M for almost all page request sequences σ. By transitivity, $\mathcal{P}$'s fault rate converges to F_σ for almost all page request sequences σ. □

5. *The mth Order Markov Source Model*

In this section, we prove Theorem 2.5 by describing our online prefetcher $\mathcal{M}$ for the case when the source is an mth order Markov source. It is easier to prefetch optimally under this model because we implicitly know the transitions between the states of M. The only problem that remains is to estimate the probabilities on the transitions.

In the mth order Markov source, the probability distribution of the next page request is dependent only on the previous m page requests. Our online algorithm for prefetching $\mathcal{M}$ builds the finite state machine of the source, the states labeled with m-contexts and the transitions denoting the unique change from one m-context to the next. Since the state structure of the source is known, $\mathcal{M}$ is always in the same state that the source M is in (except possibly for the first m page requests). The prefetcher $\mathcal{M}$ estimates the probability of each transition to be the frequency that it is taken. Algorithm $\mathcal{M}$ prefetches the pages with the top k estimated probabilities at the current state.

From the probability theory of Markov chains and renewal processes [Karlin and Taylor 1975], we know that at time instant θ, $E(|p_z - \hat{p}_z|) = O(1/\sqrt{\theta})$, and $E(|p_{z,i} - \hat{p}_{z,i}|) = O(1/\sqrt{\theta})$. Using this in conjunction with Lemma 4.2.5, we see that the expected difference in fault between $\mathcal{M}$ and the source for time θ is $O(1/\sqrt{\theta})$. Since $\left(\sum_{\theta=1}^{n} 1/\sqrt{\theta}\right)/n = O(1/\sqrt{n})$, it follows that the expected difference in fault rate between $\mathcal{M}$ and the source is $O(1/\sqrt{n})$, where n is the length of the page request sequence.[6] (This is in contrast to prefetcher $\mathcal{P}$ which converges to the optimal fault rate of the general Markov source as $O(1/\sqrt{\log n})$; see Theorem 2.2.) Theorem 2.5 follows. Knowing the structure of the source thus allows us to prove a faster rate of convergence.

6. *Using $\mathcal{P}$ in Practice and Nonpure Prefetching*

In this paper, we have analyzed our prefetchers under the pure prefetching model, in which we can prefetch as many pages as desired between any two page requests, limited only by the size of the cache and without regard for timing issues. As pointed out in Section 1, this provides for mathematical elegance by isolating the prediction component of prefetching from the cache replacement component. Analyzing a general prefetcher mathematically is extremely hard; in fact it is a challenging open problem to design an online caching algorithm (with no prefetching) under a Markov source that converges to the fault rate of the optimal caching algorithm.

The primary emphasis of this paper has been to establish the close connection between data compression and prediction and derive theoretically optimal algorithms for prefetching. A natural question at this stage is to understand how the algorithms perform in practice. Practical issues that arise in implementing the prefetcher $\mathcal{P}$ (or any data compression-based prefetcher) are extensive and are discussed in a separate publication [Curewitz et al. 1993] and a patent application [Vitter et al. 1996]. In particular, it is shown in Curewitz et al. [1993] and Vitter et al. [1996] how to tackle the issues arising from the limited space available for storing the data structures of the prefetcher, and from the limited time available to do the prefetching. The impact of melding good cache replacement strategies with good pure prefetchers in order to get general nonpure prefetchers, and how to deal with situations when the user's page request preempts prefetching are also studied. In Curewitz et al. [1993], significant reductions in fault rate are demonstrated (e.g., a 15–70% reduction in fault rate over using the least recently used heuristic for cache replacement) using

[6] By using the large deviation theory used to prove Theorem 2.3, we can derive a similar "almost everywhere" convergence result for mth order Markov sources.

CAD application traces and traces obtained from the OO1 and OO7 benchmarks. Many object-oriented traces and the CAD application traces show little or no benefit from prefetching using the UNIX-based "next page prefetching" heuristic. Interestingly, in practice, just as with data compression, the prediction by partial match (PPM)-based approaches perform better for prefetching than the Lempel–Ziv-based prefetcher $\mathcal{P}$. It would be interesting to see the effect of such data compression-based prefetchers for hypertext-based applications, such as Internet browsers where the caching is done at the client.

An interesting observation from Curewitz et al. [1993] is that for the prefetchers built from data compressors, the first prediction is always the most significant; in other words, there is little difference in practice between prefetching only one page between any two user page requests and pure prefetching. This suggests an interesting nonpure prefetching model to analyze that is somewhere between pure prefetching and cache replacement.

7. *Conclusions*

Our starting point in this research was the intuition that prediction is synonymous with data compression, and that a good data compressor should be able to predict well for prefetching purposes. We have constructed a universal prefetcher $\mathcal{P}$ based on the Ziv–Lempel data compression algorithm that prefetches optimally in the limit for almost all sequences emitted by a Markov source. Some practical issues regarding prefetching are addressed in Section 6.

In follow-on work, an alternative prefetching model analyzed in Krishnan and Vitter [1996] allows the source to be *worst-case*, that is, determined by an adversary. The performance of the proposed prefetcher is shown to be optimal in the worst case with respect to the optimal finite-state prefetcher. The time per prediction is also optimal. The model is along the lines of that proposed for the $\alpha = 2$, $k = 1$ case discussed in Feder et al. [1992]. The approach in Krishnan and Vitter [1996] is necessarily different from those of this paper and Feder et al. [1992] in order to handle the general case and additionally to perform the prediction in constant time per prediction.

The prefetching algorithms we have considered in this paper are adaptive and based only on the page request sequence. They do not attempt to take advantage of possible knowledge of the application that is issuing the requests or specially provided hints by the programmer, which can be used to guide prefetching, as for example in Patterson et al. [1993] and Trivedi [1979]. We could combine our prefetcher with such special-purpose techniques so as to get the best of both worlds. Methods for combining predictions for caching appear in Fiat et al. [1991].

Under the compiler-directed prefetching paradigm used for scientific programs [Chen and Baer 1992; Mowry et al. 1992; Rogers and Li 1992], the compiler reorders instructions in application code and introduces explicit prefetching instructions to reduce the effect of cache misses. The prefetch requests are issued much in advance of their anticipated use. This introduces a number of interesting timing issues dealing with when to initiate the prefetch request, and provides another possible model to study prefetching analytically.

The framework of Abe and Warmuth [1990], who investigated a quite different learning problem related to FSAs, has led us to propose a static PAC-learning framework for prefetching, in which the prefetcher is trained on several independently generated sequences of a particular length generated by a source, and the prefetcher should converge sufficiently fast. A harder model is to assume that the prefetcher is trained on one sufficiently long sequence generated by a source. For certain special cases of sources, like mth order Markov sources, we expect that the optimal prefetcher is PAC-learnable. An interesting related model is that of probabilistic concepts [Kearns and Schapire 1990]. We can modify the model so that page requests are labeled by whether or not the optimal machine M faulted. (In real life, though, we wouldn't have this feedback.) The requests are generated by a Markov source rather than independently; the distance measure corresponds to the difference in expected fault rate with the optimal fault rate.

Finally, it is important to note that the type of analysis presented here in this paper is similar to, but not the same as, competitive analysis. It should prove useful in establishing the goodness of online algorithms for problems that intuitively cannot admit a competitive online algorithm.

ACKNOWLEDGMENTS. We thank Yali Amit and Paul Howard for several helpful discussions and comments.

REFERENCES

ABE, N., AND WARMUTH, M. 1990. On the computational complexity of approximating distributions by probabilistic automata. UCSC, UCSC-CRL-90-63.

AMIT, Y., AND MILLER, M. 1990. Large deviations for coding Markov chains and Gibbs random fields. Tech. Rep. Washington Univ.

BELL, T. C., CLEARY, J. C., AND WITTEN, I. H. 1990. *Text Compression*. Prentice-Hall Advanced Reference Series. Prentice-Hall, Englewood Cliffs, N.J.

BLACKWELL, D. 1956. An analog to the minimax theorem for vector payoffs. *Pac. J. Math. 6*, 1–8.

BLUMER, A., EHRENFEUCHT, A., HAUSSLER, D., AND WARMUTH, M. K. 1987. Occam's razor. *Inf. Proc. Lett. 24*, 377–380.

BLUMER, A., EHRENFEUCHT, A., HAUSSLER, D., AND WARMUTH, M. K. 1989. Learnability and the Vapnik–Chervonenkis dimension. *J. ACM 36*, 4 (Oct.), 929–965.

BOARD, R., AND PITT, L. 1990. On the necessity of occam algorithms. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computation* (Baltimore, Md., May 14–16). ACM, New York, pp. 54–63.

BORODIN, A., IRANI, S., RAGHAVAN, P., AND SCHIEBER, B. 1991. Competitive paging with locality of reference. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computation* (New Orleans, La., May 6–8). ACM, New York, pp. 249–259.

BRADY, J. T. 1986. A theory of productivity in the creative process. *IEEE CG&A* (May), 25–34.

CHEN, T. F., AND BAER, J. L. 1992. Reducing memory latency via non-blocking and prefetching caches. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems* (Boston, Mass., Oct.). ACM, New York, pp. 51–61.

COVER, T. M., AND SHENHAR, A. 1977. Compound Bayes predictors with apparent Markov structure. *IEEE Trans. Syst. Man Cyb. SMC-7* (June), 421–424.

CUREWITZ, K. M., KRISHNAN, P., AND VITTER, J. S. 1993. Practical prefetching via data compression. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* (Washington, D.C., May 26–28). ACM, New York, pp. 257–266.

FEDER, M., MERHAV, N., AND GUTMAN, M. 1992. Universal prediction of individual sequences. *IEEE Trans. Inf. Theory IT-38* (July), 1258–1270.

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

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

HANNAN, J. F. 1957. Approximation to Bayes risk in repeated plays. In *Contributions to the Theory of Games, Vol. 3, Annals of Mathematical Studies*. Princeton, N.J., 97–139.

HOWARD, P. G., AND VITTER, J. S. 1992. Analysis of arithmetic coding for data compression. *Inf. Proc. Man. 28*, 749–763 (invited paper in Special Issue on Data Compression for Images and Texts).

IRANI, S., KARLIN, A. R., AND PHILLIPS, S. 1992. Strongly competitive algorithms for paging with locality of reference. In *Proceedings of the 3rd Annual ACM–SIAM Symposium on Discrete Algorithms* (Orlando, Fla., Jan. 27–29). ACM, New York, pp. 228–236.

KARLIN, A. R., PHILLIPS, S. J., AND RAGHAVAN, P. 1992. Markov paging. In *Proceedings of the 33rd Annual IEEE Conference on Foundations of Computer Science* (Oct.). IEEE, New York, pp. 208–217.

KARLIN, S., AND TAYLOR, H. M. 1975. *A First Course in Stochastic Processes*, 2nd ed., Academic Press, New York.

KEARNS, M. J., AND SCHAPIRE, R. E. 1990. Efficient distribution-free learning of probabilistic concepts. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science* (Oct.). IEEE, New York, pp. 382–391.

KRISHNAN, P., AND VITTER, J. S. 1996. Optimal prediction for prefetching in the worst case. *SIAM J. Comput.* to appear. (A shortened version appears in *Proceedings of the 5th ACM–SIAM Symposium on Discrete Algorithms*, Jan. 1994. ACM, New York, pp. 372–401.)

LAIRD, P. 1992. Discrete sequence prediction and its applications. AI Research Branch, NASA Ames Research Center, Moffet Field, Calif.

LANGDON, G. G. 1983. A note on the Ziv–Lempel model for compressing individual sequences. *IEEE Trans. Inf. Theory 29* (Mar.), 284–287.

LANGDON, G. G. 1984. An introduction to arithmetic coding. *IBM J. Res. Develop. 28* (Mar.), 135–149.

LEMPEL, A., AND ZIV, J. 1976. On the complexity of finite sequences. *IEEE Trans. Inf. Theory IT-22*, 1 (Jan.), 75–81.

MCGEOCH, L. A., AND SLEATOR, D. D. 1989. A strongly competitive randomized paging algorithm. CS-89-122. Carnegie-Mellon Univ., Pittsburgh, Pa.

MOWRY, T. C., LAM, M. S., AND GUPTA, A. 1992. Design and evaluation of a compiler algorithm for prefetching. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems* (Boston, Mass., Oct.). ACM, New York, pp. 62–73.

PALMER, M., AND ZDONIK, S. 1991. Fido: A cache that learns to fetch. In *Proceedings of the 1991 International Conference on Very Large Databases* (Barcelona, Spain, Sept.). Morgan-Kaufmann, San Mateo, Calif., pp. 255–264.

PATTERSON, R. H., GIBSON, G. A., AND SATYANARAYANAN, M. 1993. A status report on research in transparent informed prefetching. *ACM Oper. Syst. Rev. 27*, (Apr.), 21–34.

ROGERS, A., AND LI, K. 1992. Software support for speculative loads. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems* (Boston, Mass., Oct.). ACM, New York, pp. 38–50.

SHWARTZ, A., AND WEISS, A. 1995. *Large Deviations for Performance Analysis*. Chapman & Hall, New York.

SLEATOR, D. D., AND TARJAN, R. E. 1985. Amortized efficiency of list update and paging rules. *Commun. ACM 28*, 2 (Feb.), 202–208.

TRIVEDI, K. S. 1979. An analysis of prepaging. *Computing 22*, 191–210.

VAPNIK, V. 1982. *Estimation of Dependencies Based on Empirical Data*. Springer-Verlag, New York.

VITTER, J. S., CUREWITZ, K., AND KRISHNAN, P. 1996. Online background predictors and prefetchers. Duke Univ., United States Patent No. 5,485,609.

WITTEN, I. H., NEAL, R. M., AND CLEARY, J. G. 1987. Arithmetic coding for data compression. *Commun. ACM 30*, 6 (June), 520–540.

ZIV, J., AND LEMPEL, A. 1978. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory 24* (Sept.), 530–536.

RECEIVED JULY 1991; REVISED NOVEMBER 1995; ACCEPTED MAY 1996