Optimal Prefetching via Data Compression

Jeffrey Scott Vitter and P. Krishnan

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-91-46
July 1991

Optimal Prefetching via Data Compression*

Jeffrey Scott Vitter
Dept. of Computer Science
Brown University
Providence, R. I. 02912–1910
(401) 863–7646
jsv@cs.brown.edu

P. Krishnan
Dept. of Computer Science
Brown University
Providence, R. I. 02912–1910
(401) 863–7675
pk@cs.brown.edu

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 ratio, with particular applications to large-scale databases and hypertext systems. Our 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 accesses.

*An extended abstract appears in *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, Puerto Rico, October 1991. Support was provided in part by an National Science Foundation Presidential Young Investigator Award CCR–9047466 with matching funds from IBM, by NSF research grant CCR–9007851, 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.

1 Introduction

Most computer memories are divided into fast memory (or *cache*) and slow memory (such as disk storage). Any application requires that the pages it accesses be in cache. In the event that it is not in cache, a *page fault* occurs and the page is fetched from slow memory to cache. The application has to wait until this fetch is completed. The method of fetching pages into cache only when a fault occurs is called *demand fetching*. The problem of *caching* is to decide which pages to remove from cache to accommodate the incoming pages.

In many database applications and typically hypertext systems, users spend a significant amount of time processing a page, and the computer and I/O system are essentially idle during that period. If the computer can predict which page the user will access next, it can fetch that page into cache (if it is not already in cache) *before* the user asks for it. Thus, when the user actually asks for the page, the page is available instantaneously, and the user perceives a faster response time. This method of getting pages into cache before they are requested is called *prefetching*.

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 [SlT] 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 accesses, 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 [BIR, FKL, McS, SlT].

It is unreasonable to expect algorithms to be competitive in this sense for prefetching. An optimal offline algorithm for prefetching would 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 accesses 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 accesses. 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.*

Prefetching is a learning problem that involves predicting the page accesses of the user. The UNIX prefetcher processes the user's page accesses to detect if pages are ever accessed sequentially, in which case it prefetches the next page in sequence.

Independently to our approach, there has been recent work by Palmer and Zdonik, who use a neural network approach to prediction [PaZ], and by Salem, who computes various first-order statistics for prediction [Sal]. Their evaluations of prefetching performance are empirical.

In this paper, we give the first provable theoretical bounds on prefetching performance. Our novel approach is to use optimal data compression methods to do optimal prefetching. Recent work in computational learning theory [BEHa, BEHb, BoP] has shown that prediction is synonymous with generalization and data compression. 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.

In this paper, we adapt an optimal data compression method to get an optimal 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 [ZiL]. We assume that there is sufficient time to prefetch as many pages as wanted, limited only by the cache size k. In practice, prefetching requests would be interrupted by the user's actual read requests. 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 access 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.

2 Page Access 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 to the base two is denoted by "lg," the natural logarithm by "ln," and the empty string by λ.

Definition 1 [Gal] 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 $\sum_{i \in A} p(z, i) = 1$ 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. For such a source M we introduce the notion of *minimum 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 prefetcher M puts into cache the k pages having the k maximum probabilities, which minimizes the probability of page fault during the next page access. 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 [ZiL] data compressor to get our optimal prefetcher $\mathcal{P}$. Theorems 1 and 2 below are our main results.

Theorem 1 *Let M be a Markov source. The expected page fault rate achieved by $\mathcal{P}$ approaches F_M, as the page access sequence length $n \to \infty$. If M is ergodic, we get not only convergence of the mean but also the much stronger 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 access 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 access sequence σ of length n.

Theorem 2 *Let M be an ergodic Markov source. For any page access 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 access sequences σ, as $n \to \infty$.*

The convergences in Theorems 1 and 2 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 1 *Theorems 1 and 2 hold even when the Markov source M is stationary.*

Our bound on convergence rate is slow, but the Ziv-Lempel encoder works well in practice for data compression. To get a faster provable convergence of fault rate, we consider a second model, in which the source is assumed to be mth order Markov.

In such sources, the probability of the next character is dependent only on the past m characters. We can therefore build a finite state machine for the source, the states labeled by m-contexts and the transitions denoting the unique change 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.

Theorem 3 *If the source is an mth order Markov source, for any sequence, the page fault rate of $\mathcal{M}$ converges exponentially fast to the fault rate of the source.*

The difficulty of the main model (Theorems 1 and 2) as opposed to the mth order model (Theorem 3) 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 [ZiL] 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 in the following way: It parses each block of size n into distinct substrings $x_0 = \lambda$, x_1, x_2, ..., x_c such that for all $j \geq 1$ substring x_j without its last character is equal to some x_i, for $0 \leq i < j$. It encodes the substring x_j 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 [HoV, Lanb, WNC] is a coding technique that achieves a coding length equal to the entropy of the data model. Sequences of probability p are encoded using $\lg(1/p)$ 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 [BCW, Lana], 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 [BCW, HoV, Lana]. Hence, the optimality results in [ZiL] hold without change for the character-based approach.

Example 1 Assume for simplicity that our alphabet is $\{a, b\}$. We consider the page access sequence "$aaaababaabbbabaa\ldots$". The Ziv-Lempel encoder parses this string as "$(a)(aa)(ab)(aba)(abb)(b)(abaa)\ldots$". 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"

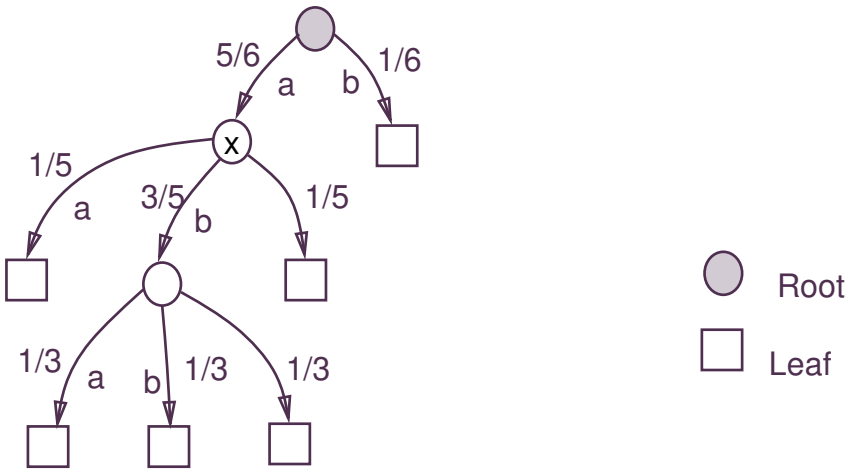

Figure 1: The parse tree constructed by the character-based encoder $\mathcal{E}$ for Example 1.

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 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, ... whose product $\prod_i p_i$ equals $1/6$. The arithmetic coder encodes the sequence with $\sum_i \lg(1/p_i) = \lg(1/\prod_i p_i) = \lg 6$ 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 access, $\mathcal{P}$ 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. 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 accesses, for some appropriately large n, $\mathcal{P}$ throws away its model and starts afresh.

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 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 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 2 [Gal] Let M be a Markov source. The minimum 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 $\mathcal{E}$ 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 3 Given an encoder C, we define C's *compression* (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 $|S| \leq s$. 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 achieved is equal to the entropy of M; that is,

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

The compression definitions in Definition 3 above are similar to those of Ziv and Lempel [ZiL], except that they define $M(s)$ to be a class of "information lossless" non-probabilistic 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 [ZiL] to our model $M(s)$ of probabilistic FSAs, using an iterative analysis based on arithmetic coding, to get the following theorem:

Theorem 4 *The compression of $\mathcal{E}$ on σ_1^n is no worse than the best probabilistic FSA in the limit as $n \to \infty$. In particular,*

$$\mathit{Compression}_{\mathcal{E},n}(\sigma_1^n) \leq \mathit{Compression}_{M(s),n}(\sigma_1^n) + \delta_s(n), \qquad \lim_{n\to\infty} \delta_s(n) = 0.$$

To prove Theorem 4, we first prove some facts about arithmetic coding and the way the character-based encoder $\mathcal{E}$ works. The first step is to show a lower bound on $\mathit{Compression}_{M(s),n}(\sigma_1^n)$, for which we need the following lemmas.

Lemma 1 *Suppose that σ_1^n can be parsed into c substrings, such that no substring is a prefix of another. We have*

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

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 $\lg(1/u)$ bits; for more details refer [HoV, Lanb, WNC]. 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," 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 non-overlapping, 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. □

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. 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 set of leaves of a parse tree is a maximal set of such "prefix-closed" substrings.

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 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 2 *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'.

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 $v \geq c^2/20n$. □

We now use Lemmas 1 and 2 to get a lower bound on the compression of σ_1^n by any finite state encoder.

Lemma 3 *For any string σ_1^n, we have*

$$\begin{aligned} \mathit{Compression}_{M(s),n}(\sigma_1^n) \;\geq\; & \frac{1}{n}\Big(2c(\sigma_1^n)\lg c(\sigma_1^n) - c(\sigma_1^n)\lg n \\ & \qquad - c(\sigma_1^n)\lg s^2 - c(\sigma_1^n)\lg\left(20(2\alpha+2)^4\right)\Big), \end{aligned}$$

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 such that no one substring is a prefix of another, and by Lemma 1, 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 recursively lowerbound the encoding length for the remaining nodes of the tree.

To analyze this recursive 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 1 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 2, we have $v_i \geq c_i^2/20n$. Hence,

$$\begin{aligned} \mathit{Compression}_{M(s),n}(\sigma_1^n) \;&\geq\; \frac{1}{n}\sum_{i=1}^{r}(c_{i-1}-c_i)\lg\frac{v_i}{s^2} \\ &\geq\; \frac{1}{n}\sum_{i=1}^{r}(c_{i-1}-c_i)\lg c_i^2 - \frac{\lg 20ns^2}{n}\sum_{i=1}^{r}(c_{i-1}-c_i). \end{aligned} \tag{4}$$

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

$$\sum_{i=1}^{r}(c_{i-1}-c_i)\lg c_i^2 \geq 2\sum_{i=1}^{r}(c_{i-1}-c_i)\lg c_0 - 2\lg(2\alpha+2)\sum_{i=1}^{r} i(c_{i-1}-c_i). \qquad (5)$$

Applying simple telescoping to (5) and substituting the result in (4) we get

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

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.

Proof of Theorem 4: It has been shown in [ZiL] that

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

where $c(\sigma_1^n)$ is the maximum number of nodes in any parse tree[1] for σ_1^n. It is shown in [LeZ] that

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

Theorem 4 is clearly true when $c(\sigma_1^n) = o(n/\lg n)$ since $\mathit{Compression}_{\mathcal{E},n}(\sigma_1^n) \sim 0$ 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 3 and the upper bound for $\mathit{Compression}_{\mathcal{E}}(\sigma_1^n)$ from (7), we get by simple arithmetic that

$$\lim_{n\to\infty}\left(\mathit{Compression}_{\mathcal{E},n}(\sigma_1^n) - \mathit{Compression}_{M(s),n}(\sigma_1^n)\right) = 0.$$

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, taking the expected value of both sides of the inequality, and using expression (2) we get the following corollary that the encoder $\mathcal{E}$ compresses as well as possible, achieving the entropy of the source M in the limit.

Corollary 2 *Let M be a Markov source with s states. Then we have*

$$E\left(\mathit{Compression}_{\mathcal{E},n}\right) \leq H_M(n) + \delta'_s(n), \qquad \lim_{n\to\infty}\delta'_s(n) = 0.$$

[1] This is not the definition of $c(\sigma_1^n)$ in [ZiL] but it is easy to verify that the proofs in [ZiL] also hold under this definition of $c(\sigma_1^n)$.

4.2 Bounds on Fault Rate

Along the lines of entropy in Definition 2, we introduce the corresponding notion of the *minimum 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, M puts into the cache those k pages having the maximum probabilities for state z.

Definition 4 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 minimum 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 *minimum expected fault rate* F_M of M.

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

Definition 5 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 *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 $\mathcal{P}$ 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 1).

Theorem 5 *Let M be a Markov source with s states. We have*

$$E\left(Fault_{\mathcal{P},n}\right) \leq F_M(n) + \epsilon_s(n), \qquad \lim_{n \to \infty} \epsilon_s(n) = 0.$$

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 5 *Suppose that at time instant θ the Markov source M is at state z and the next page access 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 access will be i with probability r_i. Let us denote the probability of page fault by M and $\mathcal{P}$ on the next page access 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 $C = A \bigcap B$. For any set X of pages, let $p_X = \sum_{i \in X} p_i$ and let $r_X = \sum_{i \in X} r_i$. Then $f_{\mathcal{P}} - f_M = (1 - p_B) - (1 - p_A) = p_A - p_B = p_{A-C} - p_{B-C}$. Let $\sum_{i=1}^{\alpha} |p_i - r_i| = \epsilon$. 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 summation on the right-hand side of the next lemma is the Kullback-Leibler divergence of $(r_1, \ldots, r_\alpha)$ with respect to $(p_1, \ldots, p_\alpha)$.

Lemma 6 *Given two probability vectors $(p_1, \ldots, p_\alpha)$ and $(r_1, \ldots, r_\alpha)$, we have*

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

Proof: The above lemma is well known; however, we reproduce the proof from [AmM] here for the sake of completeness. 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 the left hand side of the inequality to be proved, and applying the Cauchy Schwartz inequality, we obtain

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

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

Lemma 7 *Let $\sum_{i=1}^{u} \gamma_i x_i$ be a convex linear combination of the nonnegative quantities x_i; that is, $\gamma_i \geq 0$ and $\sum_{i=1}^{u} \gamma_i = 1$. 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 5.

Proof of Theorem 5: 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 2), 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[2]

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

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

Let $\mathit{Compression}_{\mathcal{E}}^{\ell+1}(\sigma_1^\ell, z, \sigma_{\ell+1}^{\ell+1})$ be $\mathcal{E}$'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 $\mathit{Compression}_{\mathcal{E},n}(\sigma_1^n) = (1/n) \sum_{\ell=0}^{n-1} \mathit{Compression}_{\mathcal{E}}^{\ell+1}(\sigma_1^\ell, z, \sigma_{\ell+1}^{\ell+1})$. We would like to express $E(\mathit{Compression}_{\mathcal{E},n})$ 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 $\mathit{Compression}_{\mathcal{E}}^{\ell+1}(\sigma_1^\ell, z)$ with mean $\sum_{i=1}^{\alpha} p_{z,i} \lg(1/r_{\sigma_1^\ell, i})$, where $r_{\sigma_1^\ell, i} > 0$ is the probability with which $\mathcal{E}$ expects to see character i next after having processed σ_1^ℓ in which M ends up in state z. We have

$$\begin{aligned} E(\mathit{Compression}_{\mathcal{E}}^{\ell+1}) &= \sum_{z \in S} \sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell) \cdot [g(z_0, \sigma_1^\ell) = z] \cdot E(\mathit{Compression}_{\mathcal{E}}^{\ell+1}(\sigma_1^\ell, z)) \\ &= \sum_{z \in S} \sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell) \cdot [g(z_0, \sigma_1^\ell) = z] \cdot \sum_{i=1}^{\alpha} p_{z,i} \lg \frac{1}{r_{\sigma_1^\ell, i}}. \end{aligned} \tag{10}$$

Summing on ℓ in (10), we get

$$\begin{aligned} E(\mathit{Compression}_{\mathcal{E},n}) &= \frac{1}{n} \sum_{\ell=0}^{n-1} E(\mathit{Compression}_{\mathcal{E}}^{\ell+1}) \\ &= \frac{1}{n} \sum_{z \in S} \sum_{\ell=0}^{n-1} \sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell) \cdot [g(z_0, \sigma_1^\ell) = z] \cdot \sum_{i=1}^{\alpha} p_{z,i} \lg \frac{1}{r_{\sigma_1^\ell, i}}. \end{aligned} \tag{11}$$

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

Combining (11) and (9), we get

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

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

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

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 access $\sigma_{\ell+1}^{\ell+1}$, given that M is in state z after emitting the first ℓ page accesses σ_1^ℓ. We have $\mathit{Fault}_{\mathcal{P},n}(\sigma_1^n) = (1/n)\sum_{\ell=0}^{n-1}\mathit{Fault}_{\mathcal{P}}^{\ell+1}(\sigma_1^\ell, z, \sigma_{\ell+1}^{\ell+1})$. If we specify the first ℓ page accesses and leave the $(\ell+1)$st page access $\sigma_{\ell+1}^{\ell+1}$ unspecified, we get the random variable $\mathit{Fault}_{\mathcal{P}}^{\ell+1}(\sigma_1^\ell, z)$ with mean $\sum_{i\notin K_z(\mathcal{P},\sigma_1^\ell)} p_{z,i}$, 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

$$\begin{aligned} E(\mathit{Fault}_{\mathcal{P}}^{\ell+1}) &= \sum_{z\in S}\sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell)\cdot[g(z_0,\sigma_1^\ell)=z]\cdot E(\mathit{Fault}_{\mathcal{P}}^{\ell+1}(\sigma_1^\ell, z)) \\ &= \sum_{z\in S}\sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell)\cdot[g(z_0,\sigma_1^\ell)=z]\cdot\sum_{i\notin K_z(\mathcal{P},\sigma_1^\ell)} p_{z,i}. \end{aligned} \quad (14)$$

Summing on ℓ in (14), we get

$$\begin{aligned} E(\mathit{Fault}_{\mathcal{P},n}) &= \frac{1}{n}\sum_{\ell=0}^{n-1} E(\mathit{Fault}_{\mathcal{P}}^{\ell+1}) \\ &= \frac{1}{n}\sum_{z\in S}\sum_{\ell=0}^{n-1}\sum_{\text{all } \sigma_1^\ell} \Pr(\sigma_1^\ell)\cdot[g(z_0,\sigma_1^\ell)=z]\cdot\sum_{i\notin K_z(\mathcal{P},\sigma_1^\ell)} p_{z,i}. \end{aligned} \quad (15)$$

Combining (13) and (15), we get

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

From Lemma 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 $\sum_{i=1}^{\alpha}|p_{z,i}-r_{\sigma_1^\ell,i}|$ in (17) by $\epsilon(z,\ell,\sigma_1^\ell)$, and let us denote the term $\sum_{i=1}^{\alpha}p_{z,i}\lg(p_{z,i}/r_{\sigma_1^\ell,i})$ in (12) by $\delta(z,\ell,\sigma_1^\ell)$, . Lemma 6 bounds $\epsilon(z,\ell,\sigma_1^\ell)$ by $\sqrt{(\ln 4)\delta(z,\ell,\sigma_1^\ell)}$. By three applications of Lemma 7 to (12) and (17) and by the bound on $E(Compression_{\mathcal{E},n}) - H_M(n)$ from Corollary 2, we get our result. The quantity $\epsilon_s(n)$ is bounded from above by $\sqrt{(\ln 4)\delta'_s(n)}$. □

The following theorem is the detailed version of the second part of our first main theorem (Theorem 1):

Theorem 6 *Let the page access sequence σ of length bn be generated by an ergodic Markov source M. We have*

$$Fault_{\mathcal{P},nb} \to E\left(Fault_{\mathcal{P},n}\right) \qquad \text{for almost all sequences } \sigma, \text{ as } b\to\infty$$

where by Theorem 5

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

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 $\sum_{i=1}^{b} Fault_{\mathcal{P},n}(\sigma_{(i-1)n+1}^{in})/b$. 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 [Gal]. □

The proof of Theorem 2 essentially deals with showing that F_σ converges to F_M for almost all σ as $n\to\infty$. We call two prefetchers M_1 and M_2 *distinct* if there exists some time instant θ and some page access 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 access sequence. We now see that $|M_{opt}(s)|$ is finite.

Lemma 8 *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 access 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. □

To prove Theorem 2, we first show that F_σ approaches F_M for almost all page access sequences σ. By Theorem 1, we know that the fault rate by $\mathcal{P}$ approaches F_M for almost all page access sequences. Theorem 2 then follows by transitivity.

Proof of Theorem 2: 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 f_z be the number of times σ reaches state z, divided by n, and let $f_{z,i}$ 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. The probability of sequences of length n for which

$$\sum_{z \in S_x, i \in A} f_z \, f_{z,i} \ln \frac{f_{z,i}}{p_{z,i}} \geq \delta$$

is exponentially small in n [Nat, Theorem 2], for $\delta > 0$. By Lemmas 5 and 6, we have

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

with exponentially small probability. By the definition of fault rate and Lemma 4, it follows that $E(Fault_{X,n}) = F_M(n) \leq E(Fault_{M',n})$. Since by Lemma 8 $|M_{opt}(s)|$ is finite, it follows that for any $\epsilon > 0$,

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

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

From Theorem 1, $\mathcal{P}$'s fault rate converges to F_M for almost all page access sequences σ. By transitivity, $\mathcal{P}$'s fault rate converges to F_σ for almost all page access sequences σ. □

5 The mth order Markov Source Model

It is easier to prefetch optimally under the second model (when M is an mth order Markov source) because we implicitly know the transitions between the states of M. The only problem that remains is to estimate the probabilities on the transitions. The probability of the next character is dependent only on the past m characters. 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 transitions from one m-context to the next. The prefetcher estimates the probability of each

transition to be the frequency that it is taken. These frequencies converge to the actual probabilities exponentially fast [Nat].

Since the state structure of the source is known, $\mathcal{M}$ is always in the same state that the source M is in. Let the algorithm $\mathcal{M}$ prefetch the pages with the top k estimated probabilities. The convergence of the fault rate of $\mathcal{M}$ to the fault rate of the source is related to the convergence of the estimated probabilities on all transitions to the actual probabilities, as given in Lemma 5. Hence the fault rate convergence also takes place exponentially fast. Theorem 3 follows. Knowing the structure of the source thus allows us to prove a faster rate of convergence.

6 Conclusions

Our starting point in this research was the intuition that prediction is 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. Our approach of getting a good prefetcher by using a good data compressor is novel, and we expect this approach and its refinements to have several practical applications. One open problem is to extend this result to arbitrary sources.

The prefetcher $\mathcal{P}$ as stated uses much storage space for its data structures, some of which may have to be stored on disk. We are currently investigating more storage efficient versions of $\mathcal{P}$. We have assumed for simplicity in this paper that at each instant there is sufficient time to prefetch k pages, as in typical hypertext applications. If the user's page request arrives sooner, we need to preempt the prefetching in favor of the page requested. Such limitations on I/O would introduce the problem of caching into the system, since for each prefetch we would have to decide which page to displace in the cache. We are currently investigating models incorporating these limitations.

Our prefetching algorithms are adaptive and based only on the page access sequence. They do not attempt to take advantage of possible knowledge of the application that is issuing the requests. In practice, when such knowledge is available, we could combine our prefetcher with a logical prefetcher based on the semantics of the application, so as to get the best of both worlds. Similar techniques for caching appear in [FKL].

We expect that PPM-based algorithms (prediction by partial match) [ClW], which are among the best text compressors in practice, will also perform well in practice for prefetching. PPM-based algorithms are practical implementations of mth order coding methods, and Theorem 3 indicates that performance should be good. Empirical work is progressing to compare PPM-based methods with the Ziv-Lempel methods.

The framework of Abe and Warmuth [AbW], 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 [KeS]. We can modify the model so that page accesses are labeled by whether or not the optimal machine M faulted. (In real life, though, we wouldn't have this feedback.) The accesses 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 is similar to, but not exactly the same as, competitive analysis. Such a type of analysis as has been presented here 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

[AbW] N. Abe and M. Warmuth, "On the Computational Complexity of Approximating Distributions by Probabilistic Automata," UCSC, UCSC-CRL-90-63, December 1990.

[AmM] Y. Amit and M. Miller, "Large Deviations for Coding Markov Chains and Gibbs Random Fields," Washington University, Technical Report, 1990.

[BCW] T. C. Bell, J. C. Cleary, and I. H. Witten, "Text Compression," 1990.

[BEHa] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Occam's Razor," *Information Processing Letters* 24 (1987).

[BEHb] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Learnability and the Vapnik Chervonenkis Dimension," *Journal of the ACM* (October 1989).

[BoP] R. Board and L. Pitt, "On the Necessity of Occam Algorithms," *Proceedings of the 22nd Annual ACM Symposium on Theory of Computation* (May 1990), 54–63.

[BIR] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, "Competitive Paging with Locality of Reference," *Proceedings of the 23rd Annual ACM Symposium on Theory of Computation* (May 1991).

[ClW] J. G. Cleary and I. H. Witten, "Data Compression using Adaptive Coding and Partial String Matching," *IEEE Transactions on Communication* 32 (April 1984), 396–402.

[FKL] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, "Competitive Paging Algorithms," Carnegie-Mellon University, CS–88–196, November 1988.

[Gal] R. G. Gallager, *Information Theory and Reliable Communication*, Wiley, 1968.

[HoV] P. G. Howard and J. S. Vitter, "Analysis of Arithmetic Coding for Data Compression," *Proceedings of the 1991 IEEE Data Compression Conference* (April 1991), invited paper.

[KeS] M. J. Kearns and R. E. Schapire, "Efficient Distribution-Free Learning of Probabilistic Concepts," *Proceedings of 31st Annual IEEE Symposium on Foundations of Computer Science* (October 1990), 382–391.

[Lana] G. G. Langdon, "A note on the Ziv-Lempel model for compressing individual sequences," *IEEE Transactions on Information Theory* 29 (March 1983), 284–287.

[Lanb] G. G. Langdon, "An Introduction to Arithmetic Coding," *IBM J. Res. Develop.* 28 (March 1984), 135–149.

[LeZ] A. Lempel and J. Ziv, "On the Complexity of Finite Sequences," *IEEE Transactions on Information Theory* 22 (January 1976).

[McS] L. A. McGeoch and D. D. Sleator, "A Strongly Competitive Randomized Paging Algorithm," Carnegie-Mellon University, CS–89–122, March 1989.

[Nat] S. Natarajan, "Large Deviations, Hypothesis Testing, and Source Coding for Finite Markov Sources ," *IEEE Transactions on Information Theory* 31 (1985), 360–365.

[PaZ] M. Palmer and S. Zdonik, "Fido: A Cache that Learns to Fetch," *Proceedings of the 1991 International Conference on Very Large Databases* (September 1991).

[Sal] K. Salem, "Adaptive Prefetching for Disk Buffers," CESDIS, Goddard Space Flight Center, TR–91–64, January 1991.

[SlT] D. D. Sleator and R. E. Tarjan, "Amortized Efficiency of List Update and Paging Rules," *Communications of the ACM* 28 (February 1985), 202–208.

[WNC] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic Coding for Data Compression," *Communications of the ACM* 30 (June 1987), 520–540.

[ZiL] J. Ziv and Abraham Lempel, "Compression of Individual Sequences via Variable-Rate Coding," *IEEE Transactions on Information Theory* 24 (September 1978), 530–536.