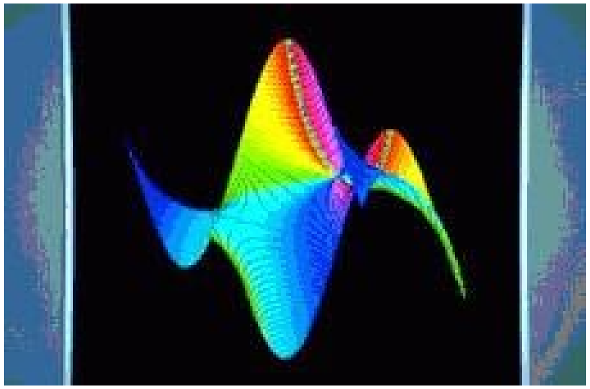
A Universal Online Caching Algorithm Based on Pattern Matching*

G. Pandurangan and W. Szpankowski
Department of Computer Science,
Purdue University

August 2, 2005



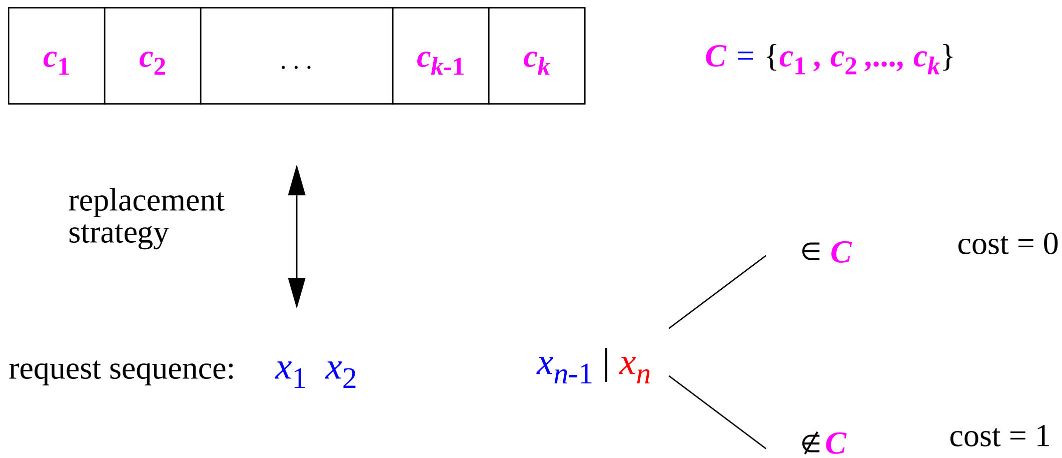
*This research is supported by NSF, NSA and NIH.

Outline of the Talk

1. Prediction, Prefetching, and Caching
2. Online Decision Problems
3. Previous Results
   - Memoryless Source
   - Markov Sources
4. Our Algorithm Based on Pattern Matching (SPMC)
5. Sketch of Proof

Prefetching, Prediction, and Caching

Prefetching/Caching



Prefetching (Prediction with $k = 1$):
at $t = n^-$ **action**

$$b_n : \ (c_1, \ldots, c_k) \leftrightarrow (y_1, \ldots, y_k) \in \mathcal{A}^k.$$

Caching:
at $t = n^+$ **action**

$$b_n : \text{ if } x_n \notin \mathcal{C}_n, \text{ then } c_i \leftrightarrow y_i \in \mathcal{A}, \quad 1 \le i \le k.$$

Online Decision Problems

In general, we are given:

sequence of requests: $x_1^n = x_1, \ldots x_n \in \mathcal{A}^n$:

actions: $b_1^n = b_1, \ldots, b_n$

loss function: $l(b_t, x_t)$.

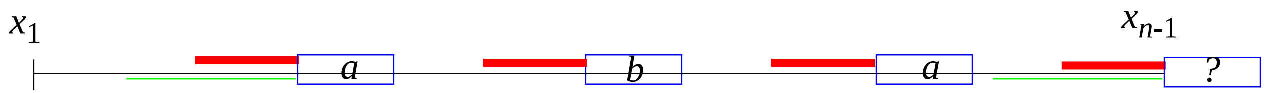
Objective: Find an online algorithm (set of actions) that minimizes the total average cost

$$\min_{b_1^n} \quad L = \frac{1}{n} \sum_{t=1}^{n} l(b_t, x_t).$$

Prefetching and prediction:
a sequential decision problem with memoryless loss function.

Caching:
a sequential decision problem with memory loss function (cf. Merhav, Ordenlich, Seroussi and Weinberger, *IT*, 2002).

ttern matching) **LZ'78** called the
npled Pattern Matching (SPM) predictor.



example, consider

SLJZGGDL $\boxed{\text{YGSJSLJZ}}$ KGSSLJZIDSLJZJGZ $\boxed{\text{YGSJSLJZ}}$,

$\boxed{\text{SLJZ}}$ GGDLYGSJ $\boxed{\text{SLJZ}}$ KGS $\boxed{\text{SLJZ}}$ KLJZJGZYGSJ $\boxed{\text{SLJZ}}$

ce the sampled sequence is GKK and the SPM predicts K.

quet, S. Apostol proved that SPM asymptotically predicts as well as the imal predictor that knows the underlying distribution.

Previous Results: Memoryless Source

Assume X_1^n is generated by a memoryless source over alphabet $\mathcal{A} = \{a_1, \ldots, a_M\}$, $M := |\mathcal{A}| \gg k$, with $p_i = P(a_i)$.
Without loss of generality we assume

$$p_1 \geq p_2 \geq \cdots p_M.$$

We assume that probabilities p_i are **known**.

Optimal Policy (cf. Aho, Denning& Ullman, 1971): Keep in the cache the first $k-1$ items with the highest probability, that is, items $\{1, \ldots, k-1\}$.

Optimal Loss:

$$L_{min} = B - \frac{1}{B} \sum_{i=k}^{M} p_i^2$$

where $B = \sum_{i=k}^{M} p_i$.

Previous Results: Markov Source

The request sequence X is generated by a Markov Source with **known** transition probabilities.

What is the optimal on-line policy?

Let:
C_t – be the content of the cache at time t,
X_t – be the t-th request,
A_t – be the set of actions (“evict page i”, $1 \leq i \leq k$).

The process $Y_t = (X_t, C_t; A_t)$ is a Markov Decision Process with the cost (weight) function

$$w_{(x,C)}(a) = \begin{cases} 1 & \text{if } x \notin C \\ 0 & \text{otherwise} \end{cases}$$

where $a \in A_t$.

Theorem 1. *There is an on-line optimal replacement policy that is a solution of a (Markov Decision Process) linear programming problem of $M\binom{M}{k}$ variables.*

Worst-Case Complexity: may be exponential in k.

Goal: **Find a near-optimal policy that runs in polynomial time in k.**

Negative Results

Consider the following "natural" on-line strategies:

LAST – on a fault, evict the page that has the highest probability of being the last of the k pages in the cache to be requested;

MAX-REACH-TIME – on a fault for a page, evict that page whose expected time to be reached from x_n is maximum.

These policies, and many others, perform poorly on Markov chains (cf. Karlin, Phillips, & Raghavan, 2000).

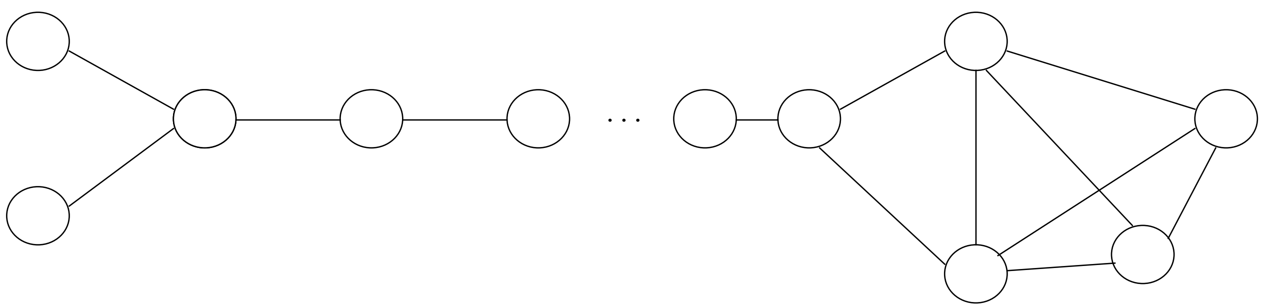


Figure 1: Example of a Markov-transition graph for which LAST and MAX-REACH perform poorly.

Lund et al. DOM Algorithm

We describe now Lund, Phillips & Reingold (1999) DOM algorithm:

1. We assume that one can precompute efficiently the probability:
$p(a,b)$: b be requested before a.

2. By solving the following Linear Programming (LP) problem:

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

we find the so called dominating probability $p(a)$ satisfying the following property:
for every a if b is selected with probability $p(b)$, then

$$\mathbf{E}[p(a,b)] \leq \frac{1}{2}.$$

DOM Algorithm

3. DOM: Evict b with probability $p(b)$.
In other words, for every a in the cache, if b in the cache is chosen with probability $p(b)$, then with probability $\geq 1/2$ a's next request will occur no later than b's next request.

4. Lund, Philipps and Reingold proved the following result.

Theorem 2 (Lund et al.). *For all request sequences x, the following holds*

$$\mathbf{E}[DOM(x)] \leq 4 \cdot ON(x),$$

where $ON(x)$ is the cost of the optimal online algorithm that has the full knowledge of the underlying distributions.
The complexity per page fault is bounded by a polynomial in k.

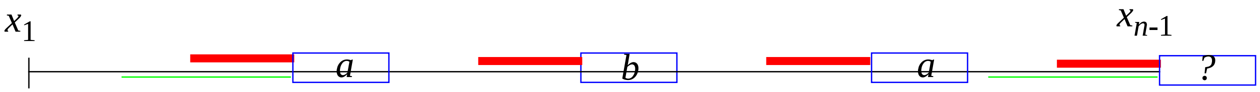
Our Universal Caching Algorithm

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

1. Find the largest suffix of x_1^n whose copy appears somewhere in the string x_1^n. Call this the maximal suffix and let its length be D_n.

2. Take an α fraction of the maximal suffix of length $k_n = \lceil \alpha D_n \rceil$, i.e., the suffix $x_{n-k_n+1} \ldots x_n$. Each occurrence of this suffix in the string x_1^n is called a marker. Let $K_n \geq 2$ be the number of occurrences of the marker in x_1^n.

$$P(a, b) = \frac{\quad}{K_n}.$$



Finishing ...

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

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

It is shown that the above LP has a feasible solution for $z \in [0,1]$ such that

$$z \leq 1/2 + 1/n^{\theta}$$

for some $\theta > 0$.
Thus, for each page a in C, if b is chosen according to p, then

$$\mathbf{E}[\tilde{P}(a,b)] \leq z.$$

5. SPM Caching Policy: Choose a page to evict from C according to the distribution p.

Main Results

Assume the source is strongly mixing, that is,

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

$$(1 - \phi(g)) \Pr(A) \Pr(B) \leq \Pr(AB) \leq (1 + \phi(g)) \Pr(A) \Pr(B).$$

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

Our main result is as follows.

Theorem 3. *Let A_n and ON_n denote the number of page faults incurred by our SPM Caching algorithm and the optimal online algorithm (ON), respectively after n requests from a strongly mixing source. Then*

$$\mathbf{E}[A_n] \leq (4 + o(1))\mathbf{E}[ON_n]$$

as $n \to \infty$.

Sketch of Proof

1. Let L denote the maximum delay before we see all symbols in the (current) cache C after any marker, i.e.,

$$L = \max_{1 \leq j \leq K_n} L_j$$

where $L_j$the delay before we see all symbols after the jth marker.

Lemma 1. *The following holds:* $L = O(\log^2 n)$ *whp.*

2. The next lemma guarantees that the estimator is consistent with "good" rate of convergence.

Lemma 2. *Let* $\theta \in (0,1)$ *be a suitably small positive constant. The estimators* $\tilde{P}(a,b)$ *for every pair of symbols* a *and* b *in cache are within* $1/n^\theta$ *of the true estimates whp for sufficiently large* n*, that is,*

$$|\tilde{P}(a,b) - P(a,b)| \leq \frac{1}{n^\theta}, \quad whp.$$

Sketch of Proof

3. Consider the following LP:
minimize z

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

By Lemma 2 we can rewrite the first constraint (whp) as:

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

where $P(a,b)$ is the true estimate of the probability that b will be requested before a and θ is a suitably small positive constant.

4. By considering the dual LP, we can show that the solution of the above LP is at most

$$z \leq 1/2 + O(1/n^{\theta}).$$

By Lemma 2, this holds with probability at least

$$1 - 1/n^{\nu}$$

for some $\nu > 0$.

Sketch of Proof

5. When our SPMC algorithm has a page fault and must evict a page, let a be a random variable denoting the page that is evicted. The following property holds: for every page b in C, the probability that b is next requested no later than a is at least

$$1/2 - O(1/n^{\theta}) - O(1/n^{\nu}).$$

By Lemma 2.5 in Lund et al., we conclude that the expected number of page faults is at most $4+o(1)$ times the optimal online algorithm as $n \to \infty$.