Caching for Web Searching

Bala Kalyanasundaram *
Computer Science Department
University of Pittsburgh
kalyan@cs.pitt.edu

John Noga †
Department of Mathematics
Technical University of Graz
noga@opt.math.tu-graz.ac.at

Kirk R. Pruhs *
Computer Science Department
University of Pittsburgh
kirk@cs.pitt.edu

Gerhard J. Woeginger †
Department of Mathematics
Technical University of Graz
gwoegi@opt.math.tu-graz.ac.at

Abstract

We study web caching when the input sequence is a depth first search traversal of some tree. There are at least two good motivations for investigating tree traversal as a search technique on the WWW: First, empirical studies of people browsing and searching the WWW have shown that user access patterns commonly are nearly depth first traversals of some tree. Secondly, (as we will show in this paper) the problem of visiting all the pages on some WWW site using anchor clicks (clicks on links) and back button clicks — by far the two most common user actions — reduces to the problem of how to best cache a tree traversal sequence (up to constant factors).

We show that for tree traversal sequences the optimal offline strategy can be computed efficiently. In the bit model, where the access time of a page is proportional to its size, we show that the online algorithm LRU is $(1 + \frac{1}{\epsilon})$-competitive against an adversary with *unbounded* cache as long as LRU has a cache of size at least $(1 + \epsilon)$ times the size of the largest item in the input sequence. In the general model, where pages have arbitrary access times and sizes, we show that in order to be constant competitive, any online algorithm needs a cache large enough to store $\Omega(\log n)$ pages; here n is the number of distinct pages in the input sequence. We provide a matching upper bound by showing that the online algorithm Landlord is constant competitive against an adversary with an unbounded cache if Landlord has a cache large enough to store the $\Omega(\log n)$ largest pages. This is further theoretical evidence that Landlord is the "right" algorithm for web caching.

*Supported in part by NSF Grant CCR-9734927.

†Supported by the START program Y43-MAT of the Austrian Ministry of Science.

1 Introduction

1.1 Problem Statement and Motivation

Web caching is the temporary local storage of WWW pages by a browser for later retrieval. From the user's point of view, the primary benefit of caching is reduced latency, as the time to access locally stored objects is minimal. We adopt the following standard general model of web caching [1, 5, 8, 14]:

Web Caching Problem Statement: The browser is given an online sequence S of page requests, where each page $p_i \in S$ has a size $s(i)$ (say, in bytes) and an access time $t(i)$ that is required if p_i is not cached. If the requested page p_i is not in the cache (this is called a cache miss), then the time to access p_i is $t(i)$. Otherwise, if p_i is in the cache (this is called a cache hit), then p_i may be accessed instantaneously. After the request of page p_i, but before the next request, the algorithm may evict/decache any arbitrary collection of pages and put p_i in its cache. At no time can the aggregate sizes of the pages in cache exceed the fixed cache size k. The objective function is to minimize the total access time.

Note that we adopt the non-forced caching model here, that is, the algorithm need not cache an accessed page. The differences between the results for forced caching and for non-forced caching models are negligible.

All of the previous work Web Caching that we are aware of assumes that the sequence S may be arbitrary. In this paper we consider the case that S is a depth first traversal of some tree T of pages. (Note that this restriction on the input allows us to obtain results that are stronger in a fundamental way.) We are motivated to consider this problem for two reasons. The first reason is that empirical studies of people browsing and searching the WWW have shown that user access patterns are commonly nearly depth first tree traversals [4, 7, 12]. That is, people tend to visit new pages via an anchor click (more than 50% of user actions are anchor clicks [4]), and to revisit pages using the back button (more than 40% of user actions are back button clicks [4]). No other action accounts for more than 2% of users' actions [4]. Secondly, we show that the problem of visiting all the pages on some WWW site using anchor clicks and the back button essentially reduces to the problem of how to best cache a tree traversal sequence. This may be viewed as providing theoretical justification for tree traversal as a search technique on the WWW.

Site Search Problem Statement: Informally, the searcher starts at the home page p_h (say for example, www.microsoft.com) of some WWW site (say Microsoft's WWW site) with unknown topology. The searcher's goal is to visit every page reachable from the home page using anchors and the back button. More formally, an online algorithm starts at some node in an initially unknown directed graph G. Each node in G is a page p_i with size $s(i)$ and access time $t(i)$. We assume that every node in G is reachable from the start page. When the online algorithm visits a node p_i it learns $s(i)$, $t(i)$, and the names of each page p_j such that (p_i, p_j) is a directed edge in G. If p_i is not in the cache then the online algorithm must pay $t(i)$, otherwise the online algorithm pays nothing for this visit. After visiting p_i, the algorithm may decache any arbitrary collection of pages and put p_i in its cache. At no time may the aggregate sizes of the pages in the cache exceed k. After making its caching decision the online algorithm may make one of two moves. First, it can push p_i onto a stack S, and then move to a page p_j with the property that (p_i, p_j) is an edge in G. Second, it can pop the top page p_j off of S and return to p_j. The online algorithm must visit every page and return to the the initial home page p_h. (Note that the requirement that the online algorithm return to the initial page is for convenience. Dropping this requirement will only change the competitive ratio by at most a factor of two.) The objective function is to minimize the aggregate access time of those visits where the visited page was not cached at the time of the visit.

Thus Site Search requires that the online algorithm must specify both a search strategy and a caching strategy. We show that, without loss of generality, online algorithms may restrict themselves to search strategies that traverse trees. That is, we show that the competitive ratio of every online algorithm A for the Site Search Problem is $\Theta\left(\max_{T \in \mathcal{T}_n} \frac{A(T)}{t(T)}\right)$, where $\mathcal{T}_n$ is the collection of all directed rooted trees on n nodes with edges directed away from the root, $A(T)$ is the total access time for algorithm A on the tree T assuming that it starts at the root of T, and $t(T) = \sum_{p_i \in T} t(i)$ is the aggregate access times of the nodes in T.

Note that there are some differences between Site Search on trees and Web Caching on depth first tree traversal sequences. The online algorithm in Site Search may decide how it will traverse the tree T (this traversal need not be a depth first search traversal), while the online algorithm for Web Caching does not have this power. In Site Search, the online algorithm learns the degree of a node when it visits that node, which is not the case in Web Caching on depth first tree traversals. And most importantly, the competitive ratio for an online algorithm A for Site Search on a tree compares $A(T)$ against the aggregate access times $t(T)$ of the pages in T, while for Web Caching on depth first tree traversal sequences S of T, the competitive ratio compares $A(S)$ against the optimal offline cost. We will show that the optimal offline cost may be much higher than $t(T)$.

There are three special cases of the caching models that have been studied previously [1, 8]. In the *bit model* the access time is assumed to equal the size of the page. This model would be appropriate if the pages are large and the delay in the network is small. In the *cost model* the size of each page is one, while the access times are allowed to be arbitrary. This is an appropriate model if the page sizes are roughly equal. For our purposes in this paper, the cost model is really no easier for online algorithms than the general model. In the *fault model* the access time for each page is constant, while page sizes may be arbitrary. The fault model will not interest us here since it is obvious that every online algorithm without a cache is constant competitive against an adversary with an unbounded cache in the fault model for Web Caching on depth first tree traversal sequences.

1.2 Our Results

Our results differ from prior work on caching in a fundamental way. In particular, we bound the size of cache required by an online algorithm in order to be constant competitive against an offline optimal algorithm that uses an *unbounded* amount of cache.

In section 2, we give the following foundational results. We show that the competitive ratio of any online algorithm for Site Search is $\Theta\left(\max_{T \in \mathcal{T}_n} \frac{A(T)}{t(T)}\right)$. We give a pseudo-polynomial time offline dynamic programming algorithm to compute $\textsc{Opt}(S)$ when S is a depth first tree traversal. This stands in contrast to offline Web Caching for general sequences, where no pseudo-polynomial time algorithm is known [1].

In section 3, we investigate Site Search and Web Caching under the bit model. For Web Caching, we show that the online algorithm LRU is $(1 + \frac{1}{\epsilon})$-competitive against an adversary with *unbounded* cache as long as LRU has a cache of size at least $(1 + \epsilon)L$, where L is the size of the largest item in the input sequence. Note that an algorithm with unbounded cache only has to pay to access each item once; so another way to state this result is that the total access time for LRU is at most $(1+\frac{1}{\epsilon})$ times the aggregate access times of the pages regardless of how often these pages are accessed. Similarly, for Site Search we show that the online algorithm that uses a depth first traversal and LRU is $(1+\frac{1}{\epsilon})$-competitive against an adversary with an unbounded cache as long as LRU has a cache of size at least $(1+\epsilon)L$.

In section 4, we give lower bounds on the competitive ratios for Web Caching and Site Search in the cost model (obviously these also hold in the general model). We first show a lower bound of $\Omega\left(\min(k, n^{1/(k+1)})\right)$ on the competitive ratio of any deterministic online algorithm for Web Caching. We then show a lower bound of $\Omega\left(\max(\frac{1}{k}, \frac{k}{\log n})n^{1/(k+1)}\right)$ on the competitive ratio of any deterministic online algorithm for Site Search. We accomplish this by showing that $\max_{t\in\mathcal{T}_n} \frac{\text{OPT}_k(T)}{t(T)}$ is $\Omega\left(\max\left(\frac{1}{k}, \frac{k}{\log n}\right) n^{1/(k+1)}\right)$, where $\text{OPT}_k(T)$ is the optimal offline cost for Site Search on T. Thus these results show that for both Web Caching and Site Search, an online algorithm needs a logarithmically sized cache to be constant competitive.

In section 5, we analyze the online algorithm Landlord (this algorithm is a generalization of LRU and is also called Greedy-Dual-Size in the literature) [3, 5, 14]. Although we will state all results in the cost model, the results hold for the general model if k is replaced by $\frac{k}{L}$. We show that Landlord is $O\left(\min\left(k, \frac{\log n}{k}\right) n^{1/(k+1)}\right)$-competitive for Web Caching on depth first tree traversal sequences. We also show that the online algorithm that uses a depth first traversal and Landlord is $O\left(\min\left(k, \frac{\log n}{k}\right) n^{1/(k+1)}\right)$-competitive for Site Search. The proper way to interpret this result is that for both Site Search and for Web Caching on tree sequences, Landlord is constant competitive against an adversary with an unbounded cache if Landlord has a cache large enough to hold at least $\log n$ pages. That is, a multiplicative increase in the number of pages only requires an additive increase in cache size to remain competitive against an adversary with infinite cache. Yet another way to interpret this result is that the number of pages that an adversary has to use to really fool Landlord is exponential in the cache size.

To date, Landlord appears to be the theoretical champion for Web Caching on arbitrary sequences [3, 5, 14]. Section 4 and section 5 together show that even if Landlord is not the theoretical champion for depth first tree traversal sequences, then at least it is not far away from being the champion. That is, even if one was going to design an online algorithm specifically for depth first tree traversal sequences, one could not do a whole lot better than Landlord. We take this as further theoretical evidence that Landlord is the "right" algorithm for Web Caching.

Notice that LRU is what we will call an *oblivious* algorithm, in that it ignores the access times of the pages. In section 6, we consider oblivious algorithms in the cost model. We show that the online algorithm Least Frequently Evicted (LFE) is optimally competitive among oblivious online algorithms for Web Caching on tree traversal sequences. Furthermore, for Site Search we show that the online algorithm that uses a depth first traversal and LFE is strongly competitive if $k = O(1)$. An online algorithm A is *strongly competitive* for a problem $\mathcal{P}$ if the competitive ratio of A is at most a constant factor worse than the competitive ratio of any other online algorithm for $\mathcal{P}$. This is in contrast to Site Search in the cost model with $k = \omega(1)$, and to Web Caching in the cost model over all ranges of k, where we show that there are no strongly competitive oblivious algorithms.

1.3 Related Results

We first discuss known results for offline Web Caching. It is easy to see that Web Caching in the bit model (and in the general model) is NP-hard in the ordinary sense. In [8] polynomial-time offline $O(\log k)$-approximation algorithms are given for the bit model and for the fault model. In [1] a polynomial-time offline $O(1)$-approximation is given, provided that the polynomial time algorithm is given additional $O(L)$ cache, where L is the size of the largest page. Additionally in [1] a polynomial time offline $O(\log(k+L))$-approximation algorithm is derived.

Next, let us consider online Web Caching. The algorithm Greedy-Dual-Size is introduced in [3], where it is shown to be k-competitive. Greedy-Dual-Size is a generalization of the algorithm Greedy-Dual in [13] that is specific for the cost model. In [14] it is shown that Greedy-Dual-Size (this paper introduces the name Landlord for this algorithm) is $\frac{k}{k-h+1}$-competitive against an adversary with cache size h assuming forced caching. In [14] it is also shown that in some sense for most choices of k, the retrieval cost is either insignificant or the competitive ratio is constant. In [5] it is shown, using linear programming duality, that Greedy-Dual-Size is $\frac{k+1}{k-h+1}$-competitive against an adversary with cache size h assuming non-forced caching. In [8] online randomized $O\left(\log^2 k\right)$-competitive algorithms are given for the bit and fault models .

Previous researchers have theoretically studied the caching problem with uniform times and uniform sizes under particular input patterns. In [2] (and in several follow-up papers) the input is assumed to be a walk in a graph, and in [11] the input is assumed to be the output of a Markov chain. In [9] it is shown that if the input sequence is a depth first traversal of a tree then LRU will have $2n-k$ cache misses, and that *LRU* always performs better than Most Recently Used on depth first traversal sequences.

In [6] the direct-mapped caching problem was studied with sequential access sequences. Perhaps the most closely related result to the search part of Site Search is in [10]; Recasting the results from a geometric setting to the Site Search setting, it is shown in [10] that there is an online algorithm that is constant competitive if $k=0$, G is planar, and the edge relation in G is symmetric.

2 Foundational Results

Theorem 1 For Site Search in any model (general, cost, or bit), the competitive ratio of every deterministic online algorithm A is $\Theta\left(\max_{T\in\mathcal{T}_n}\frac{A(T)}{t(T)}\right)$.

Proof: The competitive ratio is at most $\max_{T\in\mathcal{T}_n}\frac{A(T)}{t(T)}$, since the online searcher may perform a depth first traversal of the site and the offline searcher has to access every page at least once.

To see why the competitive ratio is at least $\frac{1}{4}\max_{T\in\mathcal{T}_n}\frac{A(T)}{t(T)}$, let T be an arbitrary directed rooted tree on n nodes with all edges directed away from the root. Let p_n be the last page in T visited by A. Create a directed graph G that includes each directed edge in T and directed edges going from p_n to every other node in T. Then A's actions on G are identical to A's actions on T until p_n is visited. From p_n, A may return directly to the root; hence, $A(G)\geq\frac{1}{2}A(T)$. The offline adversary may visit all of G incurring cost at most $2t(T)$ by traversing the shortest path from the root to p_n, then visiting each remaining unvisited node in a hub and spoke pattern from p_n, and then backing up to the root. ∎

For an instance $T\in\mathcal{T}_n$ of Site Search, we define $\mathrm{OPT}_k(T)$ to be a minimum access time strategy for visiting all the nodes in T and returning to the root of T assuming that the cache size is k. We show for Site Search on trees that the optimal offline algorithm may use any depth first search that it likes. Note that this in no way implies that the optimally competitive *online* algorithm uses depth first search.

Lemma 2 For every depth first traversal S of a tree T, there is an optimal Site Search strategy that uses S to traverse the tree T.

Proof: First notice that OPT never profits from caching a node p_i and then decaching p_i before it is accessed again. Secondly, we may assume without loss of generality that OPT always decaches a node p_i at the time that it returns to p_i's parent in T.

We first prove by a local replacement argument that there is an optimal strategy that uses *some* depth first search. Let T_c be a subtree that is visited twice from p_c's parent p_r. Assume that the sequence of nodes visited by OPT is

$$\alpha p_r p_c \beta p_c p_r \gamma p_r p_c \delta p_c p_r \epsilon$$

where α, β, γ, δ, and ϵ are subsequences of nodes. We consider two cases.

In the first case we assume that OPT had more cache unfilled after αp_r than after $\alpha p_r p_c \beta p_c p_r \gamma p_r$. In this case we modify the strategy to get a new strategy OPT$'$ by letting the sequence of pages be

$$\alpha p_r p_c \beta p_c \delta p_c p_r \gamma p_r \epsilon$$

The caching strategy by OPT$'$ within the subsequences αp_r, $p_c\beta$, $p_c\delta p_c$, $p_r\gamma$, and $p_r\epsilon$ is the same as used by OPT within these subsequences.

In the second case we assume that OPT had no more cache unfilled after αp_r than after $\alpha p_r p_c \beta p_c p_r \gamma p_r$. In this case we modify the strategy to get a new strategy OPT$'$ by letting the sequence of pages be

$$\alpha p_r \gamma p_r p_c \beta p_c \delta p_c p_r \epsilon$$

The caching strategy by OPT$'$ within the subsequences α, $p_r\gamma$, $p_r p_c\beta$, $p_c\delta$, and $p_r\epsilon$ is the same as used by OPT within these subsequences.

Its easy to see that in both cases the total access time for OPT$'$ is at most the total access time for OPT. Notice also that there are strictly fewer pages accessed in OPT$'$ than OPT. Hence, by repeating this local replacement argument a finite number of times we get an optimal tour that does not visit any edge more than twice. Consequently, there is an optimal strategy that uses *some* depth first traversal. By noting that the caching strategies used at various subtrees of a node are independent, one can see that the optimal solution may use *any* depth first search traversal. ■

Lemma 3 For Web Caching on depth first tree traversals in the cost model, and for Site Search in the cost model, $\text{OPT}_k(T)/t(T) \leq n^{\frac{1}{k+1}} + 1$ holds for any tree T with n nodes.

Proof: We will only prove the statement on Web Caching. By lemma 2, this also yields the statement for Site Search. We now describe an offline algorithm KNOWS for Web Caching such that $\text{KNOWS}(T) \leq (n^{\frac{1}{k+1}} + 1)t(T)$. The following recursive procedure takes as parameters a tree T, an upper bound n on the number of nodes in this tree, and the number k of cache locations available for traversing T.

$\textsc{Knows}(n, k, T)$: The procedure follows the request sequence through T, starting at the root p_r. Let p_s be some child of p_r, and consider the moment in time just after p_r has been requested and just before p_s is to be requested. If the number of nodes in T_s is at most $n^{\frac{k}{k+1}}$ then p_r is cached, and the recursive call $\textsc{Knows}(n^{\frac{k}{k+1}}, k-1, T_s)$ is made. If the number of nodes in T_s is more than $n^{\frac{k}{k+1}}$ then p_r is not cached, and the recursive call $\textsc{Knows}(n, k, T_s)$ is made.

We now argue that the total access time of $\textsc{Knows}(n, k, T)$ is at most $(n^{\frac{1}{k+1}} + 1)t(T)$. The proof is by induction on k and the height h of the tree. Notice that the bound follows easily for $k = 0$ or $h = 1$. By induction, we assume that the bound holds whenever $k \leq i - 1$ or $k = i$ and $h \leq j - 1$. We will prove that the bound also holds for $k = i$ and $h = j$. Consider an arbitrary tree T of height j. Let p_r be the root of T and let $T_1, T_2, \ldots, T_m$ be the subtrees rooted at the children of T. We say that a subtree T_a is *heavy* if T_a has at least $n^{\frac{k}{k+1}}$ nodes. Note that at most $n^{\frac{1}{k+1}}$ of the trees T_a with $1 \leq a \leq m$ are heavy. As a consequence, the total access time incurred on page p_r is at most $(n^{\frac{1}{k+1}} + 1)t(r)$, where the 1 accounts for the first request to p_r. By induction, the total access time incurred for traversing a heavy subtree T_a is at most $(n^{\frac{1}{k+1}} + 1)t(T_a)$. By induction, the total access time incurred for traversing a non-heavy subtree T_a is at most $(n^{\frac{k}{k+1}} + 1)^{1/k} t(T_a) \leq (n^{\frac{1}{k+1}} + 1)t(T_a)$. Hence, the total access time for $\textsc{Knows}$ in T_r is at most $(n^{\frac{1}{k+1}} + 1)t(T_r)$. ■

We now give an optimal offline algorithm for Site Search on trees and for Web Caching on depth first tree traversal sequences in the general model. Let p_r be a node in T with children $p_{c_1}, \ldots, p_{c_m}$. If $s(r) > k$ then obviously

$$\textsc{Opt}_k(T_r) = t(r) + \sum_{i=1}^{m} [\textsc{Opt}_k(T_{c_i}) + t(r)]$$

So now consider the case that $s(r) \leq k$. We say that p_{c_i} is cheap if $\textsc{Opt}_{k-s(r)}(T_{c_i}) - \textsc{Opt}_k(T_{c_i}) < t(r)$, and otherwise we say that p_{c_i} is expensive. It easy to see that one should cache p_r before visiting a cheap child p_{c_i} since the time savings from having additional $s(r)$ cache is less than the access time for p_r. Similarly, one should not cache p_r before visiting an expensive child p_{c_i} since one can reap a time savings of more than $t(p_r)$ by having additional $s(r)$ cache during the traversal of T_{c_i}. Hence,

$$\textsc{Opt}_k(T_r) \;=\; t(r) + \sum_{\text{cheap } p_{c_i}} \textsc{Opt}_{k-s(r)}(T_{c_i}) + \sum_{\text{expensive } p_{c_i}} [\textsc{Opt}_k(T_{c_i}) + t(r))]$$

The obvious dynamic programming implementation of this recurrence runs in time $O(kn)$. Summarizing, this dynamic program yields a pseudo-polynomial time algorithm for Web Caching of tree traversal input sequences in the bit model and in the general model, and a polynomial time algorithm for the cost model.

3 Bit Model

The algorithm Least Recently Used (LRU) evicts the least recently used items until there is room to fit the most recently requested item in the cache. We show that in the bit model the online algorithm LRU is $(1 + \frac{1}{\epsilon})$-competitive against an adversary with unbounded cache as long as LRU has a cache of size at least $(1 + \epsilon)L$. Recall L is the size of the largest item in the input sequence.

Theorem 4 Suppose $0 < \epsilon \leq 1$ and that that LRU is equipped with a cache of size $k \geq (1+\epsilon)L$. Then for Web Caching in the bit model where the input sequence S is a depth first traversal of some tree T, the algorithm LRU guarantees that $\frac{LRU(S)}{t(T)} \leq 1 + \frac{1}{\epsilon}$.

Proof: We split the cost of LRU into the cost incurred while moving downwards (from a parent down to a child) and the cost incurred while moving upwards (from a child up to its parent). We show by an amortization argument, that the total cost for upward moves is at most $\frac{t(T)}{\epsilon}$. There is an account Acc_i associated with each page p_i, and there is an account $\text{Acc}(\text{LRU})$ for LRU. Initially, $\text{Acc}_i = \frac{t(i)}{\epsilon}$ for each page p_i and $\text{Acc}(\text{LRU}) = 0$. When a page p_i is requested in a downward move, all accounts remain unchanged. When a node p_i is requested in an upward move and p_i is not cached, then $t(i)$ is deducted from $\text{Acc}(\text{LRU})$. If the request sequence is next going to visit another child of p_i, then all the funds in $\text{Acc}(\text{LRU})$ are moved to Acc_i, and LRU enters this subtree with an empty account. Otherwise, if the request sequence returns to p_i's parent, then all the funds in Acc_i are transferred to $\text{Acc}(\text{LRU})$.

Our first goal is to show that during an upward move from a node p_i towards its parent, $\text{Acc}(\text{LRU}) \geq \min(\frac{t(T_i)}{\epsilon}, L)$ always holds. The proof is done by induction. The base case is if p_i is a leaf. In this case the account of p_i with value $\frac{t(i)}{\epsilon}$ has just been transferred to LRU, and thus $\text{Acc}(\text{LRU}) \geq \frac{t(i)}{\epsilon} = \frac{t(T_i)}{\epsilon}$ holds. Next assume that the claim holds for each of the children $p_{c_1}, \ldots, p_{c_m}$ of p_i. We break the proof into two cases: (Case 1) First, assume that for all j, $1 \leq j \leq m$, $t(T_{c_j}) \leq \epsilon L$ holds. Then every T_{c_j} can be traversed without evicting p_i, and p_i will be kept cached throughout the traversal of T_i. Since no charges are deducted from the searchers account at p_i, the inductive claim yields that $\text{Acc}(\text{LRU}) \geq \frac{t(i)}{\epsilon} + \sum_{j=1}^{m} \frac{t(T_{c_j})}{\epsilon} = \frac{t(T_i)}{\epsilon}$ holds at the moment when LRU leaves p_i upwards to its parent. (Case 2) Now assume that there exists a j, $1 \leq j \leq m$, with $t(T_{c_j}) > \epsilon L$ and consider the moment in time when the searcher returns from p_{c_j} up to p_i. At this moment, p_i need not be in the cache. By induction, the value of $\text{Acc}(\text{LRU})$ is at least $L = \min(\frac{t(T_{c_j})}{\epsilon}, L)$. Hence, after (possibly) paying the charge for visiting p_i, $\text{Acc}(\text{LRU}) \geq L - t(i)$ holds. If the request sequence now returns to p_i's parent, then $\text{Acc}(\text{LRU}) \geq (L - t(i)) + \frac{t(i)}{\epsilon} \geq L$ (where the second term is the original amount in Acc_i). Otherwise, if the request sequence moves on to the next child of p_i, then $\text{Acc}(\text{LRU}) \geq L - t(i)$ will be added to Acc_i and so $\text{Acc}_i \geq (L - t(i)) + \frac{t(i)}{\epsilon} \geq L$ and this amount will eventually be transferred to $\text{Acc}(\text{LRU})$ before it moves up to p_i's parent.

Next, we argue that $\text{Acc}(\text{LRU})$ is never negative, and that therefore LRU can always pay for the revisits. Consider visiting a parent p_i from a child p_c. If $t(T_c) \leq \epsilon L$ then p_i is still cached at this moment and no charge is taken. Otherwise, if $t(T_c) > \epsilon L$ then $\text{Acc}(\text{LRU}) \geq L$ and LRU can afford the charge since $t(i) \leq L$ by the definition of L. Summarizing, the account of LRU stays non-negative throughout the traversal of the tree. Since the total amount of funds available in the beginning is $\frac{t(T)}{\epsilon}$ and since LRU is able to finance all its upward moves from these funds, the total incurred cost indeed is at most $\frac{t(T)}{\epsilon}$. Since the total cost of LRU for downward moves equals $t(T)$, the proof of the theorem is complete. ■

This corollary is an immediate consequence of theorem 1 and theorem 4.

Corollary 5 For Site Search in the bit model, the algorithm that uses LRU and a depth first traversal guarantees that $\frac{LRU(S)}{t(T)} \leq 1 + \frac{1}{\epsilon}$.

It is easy to see that the above bounds are tight for $\epsilon = 1$ by considering trees where each internal node has one child, and all pages have access time L. Note that in Site Search that this is not merely an artifice of our requirement that the searcher return to the root as you could always enforce this condition by adding a second leaf-child of the root.

4 Lower Bounds in the General Model

We show that in the cost model (and hence also in the general model), every online algorithm for Web Caching and every online algorithm for Site Search requires a cache of size $\Omega(\log n)$ in order to be constant competitive.

Theorem 6 For Web Caching in the cost model, any deterministic online algorithm A fulfills the following statements.

(i) Let k and n be integers such that $k+1 \leq \lg n$. Then there exists a tree T with $\Theta(n)$ nodes on which A is $\Omega\left(\min(k+1, n^{1/(k+1)})\right)$-competitive.

(ii) Let k and n be integers such that $k+1 \leq \lg n$. Then there exists a tree T with $\Theta(n)$ nodes such that $A(T)/t(T) \geq \frac{1}{4} n^{\frac{1}{k+1}}$.

Proof: The adversary constructs a tree T with $k+2$ levels numbered $0, 1, \ldots, k+1$. Level 0 only contains the root of T, level 1 contains all the children of the root, and so on. Every page at level ℓ has access time $x^{k+1-\ell}$, where $x = \frac{1}{2} n^{1/(k+1)}$. Note that $x \geq 1$ since $k+1 \leq \lg n$. Hence, every node has access time at least one. The exact shape of T is determined by the adversary in dependence on the behavior of the online algorithm A. The adversary follows a simple *Hit-Where-It-Hurts* strategy. Let p be the last requested page, and let ℓ be the level that contains p.

Expand: If $\ell < k+1$, then the adversary creates a path of $k+1-\ell$ new pages at levels $\ell+1, \ldots, k+1$ that are descendents of page p. The pages on this path are then requested one by one.

Hit: Otherwise, $\ell = k+1$ holds. The adversary requests the ancestors of p until it reaches a page that is currently not cached by the online algorithm.

The adversary alternates between expansions and hits until it has created n nodes (if this happens in the middle of an expansion or hit, this move is still completed and then the process stops). Clearly, the thus created tree T has $\Theta(n)$ nodes. By n_ℓ, $0 \leq \ell \leq k+1$, we denote the total number of nodes at the ℓ-th level of tree T. Note that $n_0 = 1$.

Now let p be a page at level $\ell \leq k$ with m children. Since all leaves of T are at level $k+1$, $m \geq 1$ holds. When the online algorithm pays for accessing p then either the adversary is expanding the tree (and p is created) or the adversary is hitting (and the request sequence returns from one of the m children). When the request sequence returns from one of the first $m-1$ children, the adversary just has done a hit. The online algorithm pays for accessing p, and then the next child is created in the following expansion. When the request sequence returns from the last child, it immediately moves on to the parent of p and we are in the middle of some hit. Altogether, for accessing page p, the algorithm A pays m times the size of p, and for all the accesses to all the pages in level ℓ, it pays the total number $n_{\ell+1}$ of their children times their access time $x^{k+1-\ell}$. For the pages in level $k+1$, A altogether pays n_{k+1} times access time 1. Summarizing, this yields

$$A(T) \;=\; n_{k+1} + \sum_{\ell=0}^{k} n_{\ell+1} x^{k+1-\ell} \;\geq\; x t(T) - x^{k+2} \;=\; x(t(T) - n/2^{k+1}) \;\geq\; \frac{x}{2} t(T). \tag{1}$$

In the last inequality, we used that $t(T) \geq n$. This inequality holds since every node has access time at least one.

One possible offline strategy always keeps all the predecessors of the currently requested page in cache, with the exception of the pages at some fixed level λ with $0 \leq \lambda \leq k$. Since T has only $k+2$ levels and since there is no need to cache the pages at level $k+1$, this strategy can always be carried out with a cache of size k. This offline strategy has to pay for accessing a page (a) if the page is requested for the first time, or (b) if the page is at level λ and if the request sequence moves from a page at level $\lambda+1$ up to level λ. The total cost for (a) is $t(T)$, and the total cost for (b) is $n_{\lambda+1}x^{k+1-\lambda}$. Hence, $\textsc{Opt}(T) \leq t(T) + \min_{\lambda=0}^{k}\{n_{\lambda+1}x^{k+1-\lambda}\}$, and a simple averaging argument yields

$$\textsc{Opt}(T) \;\leq\; t(T) + \frac{1}{k+1}\sum_{\ell=0}^{k} n_{\ell+1}x^{k+1-\ell} \;\leq\; t(T) + \frac{1}{k+1}x \cdot t(T). \tag{2}$$

By combining (1) and (2), we conclude that the competitive ratio of A is at least

$$A(T)/\textsc{Opt}(T) \;\geq\; \frac{(k+1)x \cdot t(T)}{2(k+1+x)t(T)} \;=\; \frac{(k+1)x}{2(k+1+x)} \tag{3}$$

Now let us prove statement (i) of the theorem. If $k+1 \leq n^{1/(k+1)}$, then $k+1 \leq 2x$ and we derive from (3) that

$$A(T)/\textsc{Opt}(T) \;\geq\; \frac{(k+1)x}{2(k+1+x)} \;\geq\; \frac{(k+1)x}{6x} \;=\; \frac{1}{6}(k+1).$$

If on the other hand $k+1 \geq n^{1/(k+1)}$ holds, then $k+1 \geq x$ and we derive in a similar way that

$$A(T)/\textsc{Opt}(T) \;\geq\; \frac{(k+1)x}{2(k+1+x)} \;\geq\; \frac{(k+1)x}{4(k+1)} \;=\; \frac{1}{8}n^{1/(k+1)}.$$

This proves (i). Finally, statement (ii) follows from the inequality in (1) and from $x = \frac{1}{2}n^{1/(k+1)}$. With this, the proof of the theorem is complete. ∎

Now we turn to the Site Search problem. We know from theorem 1 that without loss of generality (and up to constant factors) we only need to consider online algorithms A that traverse some subtree T of G. However, we do not necessarily know that A performs a depth first traversal of T. To get around this difficulty, we consider the following Modified Site Search problem. It is easy to see that a lower bound on the competitive ratio for any online algorithm for Modified Site Search also yields a lower bound on the competitive ratio for any online algorithm for the original Site Search problem.

Modified Site Search Problem: The online algorithm is told that the topology of G consists of a directed tree T, rooted at the initial page with the edges directed away from the initial page, and edges directed from a secret page p_s to every other page. The online algorithm is told T a priori, but is not told the identity of the secret node p_s (and actually, the adversary will make p_s the last node that the online algorithm visits). The goal of the online algorithm is still to visit all the nodes and to return to the initial page.

Recall that $\textsc{Opt}_k(T)$ is the minimum access time strategy, with cache size k, for visiting all the nodes in T and returning to the root of T. Also recall that by lemma 2 we may assume that $\textsc{Opt}_k(T)$ uses a depth first search. Note that $\textsc{Opt}_k(T)$ can be computed by the online algorithm before it begins its traversal. The competitive ratio of any online algorithm for Modified Site Search is then $\Omega\left(\max_T \frac{\textsc{Opt}_k(T)}{t(T)}\right)$. We show that $\max_T \frac{\textsc{Opt}_k(T)}{t(T)}$ is at least $\Omega\left(\max\left(\frac{1}{k}, \frac{k}{\log n}\right) n^{\frac{1}{k+1}}\right)$.

Theorem 7 For Modified Site Search, the competitive ratio of every deterministic online algorithm A is at least $\Omega\left(\max\left(\frac{1}{k}, \frac{k}{\log n}\right) n^{\frac{1}{k+1}}\right)$.

Proof: We assume that n and k are integers such that $(k+1) \leq \lg n$, otherwise the claimed lower bound of $\Omega(1)$ is trivial. We now construct a directed rooted tree T, the edges pointing away from the root, that depends on n, k, and a parameter y. Here y is a non-negative integer such that $d := \lceil n^{\frac{1}{k+1+y}} \rceil \geq 2$ holds; the exact value of y will be fixed later. Every non-leaf node in T has out-degree d. The number of nodes in any root to leaf path is $k+y+2$. We consider the root to be on level 0, its immediate children to be on level 1, and so on; all leaves are at level $k+y+1$. Each node on level ℓ, $0 \leq \ell \leq k+y+1$, has access time $d^{-\ell}$. Note that the number of nodes at level ℓ is d^{ℓ}, that the total number of nodes in T is $\Theta(n)$, and that $t(T) = k+y+2$.

Our first claim is that $\mathrm{OPT}_k(T) \geq (y+1)d$. To prove this claim, we associate with each leaf node a penalty account that is initialized with 0. The sum of the penalty accounts will be a lower bound on $\mathrm{OPT}_k(T)$. Consider an arbitrary node p_a on level ℓ and some child p_b of p_a. In case p_a is not cached when the traversal returns from p_b to p_a, then the cost $t(a) = d^{-\ell}$ is equally split among all leaves in the subtree T_b. The number of leaves in subtree T_b is $d^{k+y-\ell}$. Therefore, the penalty account of each leaf in T_b gets a charge of d^{-k-y} which is independent of the level ℓ. Now consider a leaf p_i and let N be the set of $k+y+2$ nodes in the path from the root of T to p_i. At the time that OPT_k visits p_i, at most k nodes in N are cached and at least $y+1$ nodes in $N - \{p_i\}$ are uncached. Each uncached node will charge d^{-k-y} to the penalty account at p_i, and so in the end the penalty account at p_i is at least $(y+1)d^{-k-y}$. Since the total number of leaves is d^{k+y+1}, we conclude that indeed $\mathrm{OPT}_k(T) \geq (y+1)d^{-k-y} \cdot d^{k+y+1} = (y+1)d$.

Finally, we show that for a proper choice of y, $\mathrm{OPT}_k(T)/t(T) = \Omega\left(\max\left(\frac{1}{k}, \frac{k}{\log n}\right) n^{\frac{1}{k+1}}\right)$. Then by theorem 1 the desired bound follows. If $k \leq \sqrt{\lg n}$, we set $y = 0$. Then by the above discussion, $\mathrm{OPT}_k(T) \geq d$ and $t(T) = k+2$, and the claim holds. In the remaining case $k > \sqrt{\lg n}$, and we set $y = \frac{(k+1)^2}{\lg n}$. Note that $y \leq (k+1)$ since $(k+1) \leq \lg n$ by assumption. Since $\mathrm{OPT}_k(T) \geq (y+1)d$ and $t(T) = k+y+2$, the ratio $\mathrm{OPT}_k(T)/t(T)$ is at least

$$\frac{y+1}{k+y+2} n^{\frac{1}{k+y+1}} \geq \frac{y}{4(k+1)} n^{\frac{1}{k+y+1}} \geq \frac{k+1}{4 \lg n} n^{\frac{1}{k+y+1}}.$$

Hence, we are left to verify that the ratio $n^{\frac{1}{k+y+1}}/n^{\frac{1}{k+1}}$ is $\Omega(1)$. But that follows easily, since the exponent fulfills

$$\frac{1}{k+y+1} - \frac{1}{k+1} = \frac{-y}{(k+y+1)(k+1)} = \frac{-1}{\lg n + k + 1} \geq \frac{-1}{\lg n}.$$

■

Corollary 8 For Site Search in the cost model, the competitive ratio of every deterministic online algorithm A is at least $\Omega\left(\max\left(\frac{1}{k}, \frac{k}{\log n}\right) n^{\frac{1}{k+1}}\right)$.

5 Analysis of Landlord

We show that for Web Caching on depth first tree traversal sequences and for Site Search, Landlord is constant competitive against an adversary with unbounded cache if Landlord has a cache large enough to hold at least $\log n$ pages.

Landlord Description: [5] The algorithm maintains a non-negative credit $c(i)$ for each page p_i in the cache. Given a request for p_i, if p_i is in the cache the algorithm resets $c(i)$ to $t(i)$. Otherwise, the algorithm sets $c(i) = t(i)$ and "pretends" p_i is in the cache. Then it repeats the following eviction step while the total size of the items in the cache exceeds k.
Eviction step: Let p_m be a page in the cache that minimizes the ratio $\frac{c(m)}{s(m)}$ and let $\delta = \frac{c(m)}{s(m)}$. For every p_i in the cache, the algorithm decreases $c(i)$ by $\delta s(i)$, and then evicts p_m.

Proposition 9 [5, 14] For Web Caching in the general model, Landlord is $\frac{k+1}{k-h+1}$-competitive against an adversary with a cache of size $h \leq k$.

Theorem 10 For Site Search in the cost model, the online algorithm that uses depth first search for traversing and Landlord for caching is

(i) $O\left(kn^{\frac{1}{k+1}}\right)$-competitive if $k \leq \sqrt{\log n}$,

(ii) $O\left(\frac{\log n}{k} n^{\frac{1}{k+1}}\right)$-competitive if $\sqrt{\log n} < k < \frac{1}{2}\log n$,

(iii) $O(1)$-competitive if $k \geq \frac{1}{2}\log n$.

Proof: Let h be an integer in the range $0 \leq h \leq k$. From proposition 9, we know that the competitive ratio of Landlord against an offline adversary with a cache of size h is at most $\frac{k+1}{k-h+1}$. From lemma 3, we know that $\frac{\mathrm{OPT}_h(T)}{t(T)} = O\left(n^{\frac{1}{h+1}}\right)$. Therefore, the competitive ratio of Landlord is $O\left(\min_{h=0}^{k} \frac{k+1}{k-h+1} n^{\frac{1}{h+1}}\right)$.

In case (i) we choose $h = k$ and get the desired bound. In case (ii) we set $h = \lfloor k - \frac{k^2}{\log n} \rfloor$. Note that $0 \leq h \leq k$. Observe that in this case

$$\frac{k+1}{k-h+1} n^{\frac{1}{h+1}} \leq \frac{2k}{k^2/\log n} n^{\frac{1}{h+1}} \leq \frac{2\log n}{k} n^{\frac{1}{h+1}}$$

Thus it is sufficient to establish that $n^{\frac{1}{h+1}} = O(n^{\frac{1}{k+1}})$. Note that if $k \geq \sqrt{\log n}$ then $n^{1/k} = \Theta(n^{\frac{1}{k+1}})$. The exponent of the ratio $n^{1/h+1}/n^{1/k}$ fulfills

$$\frac{1}{h+1} - \frac{1}{k} \leq \frac{\log n}{k\log n - k^2} - \frac{1}{k} = \frac{1}{\log n - k} = O\left(\frac{1}{\log n}\right).$$

Hence, $n^{1/h+1}/n^{1/k}$ indeed is $O(1)$. Finally, in case (iii) we set $h = \lfloor \frac{1}{4}\log n \rfloor$. ■

Theorem 11 For Web Caching in the cost model, Landlord is $O\left(\min\left(k, \frac{\log n}{k}\right) n^{\frac{1}{k+1}}\right)$-competitive for $k \leq \frac{1}{2}\log n$ caches, and $O(1)$-competitive for $k \geq \frac{1}{2}\log n$ caches.

Proof: The result follows from proposition 9 and theorem 10. ■

6 Oblivious Algorithms for the Cost Model

We show that Least Frequently Evicted (LFE) is essentially the best oblivious algorithm and that the algorithm for Site Search that uses depth first and LFE is strongly competitive if $k = O(1)$. Recall that an oblivious algorithm is one that ignores access times.

LFE Description: An eviction count $e(p_i)$ is maintained for each page p_i. Initially each $e(p_i) = 0$. Assume that the page p_r has just been requested at time u. If this is not the first time that p_r was requested, the page p_c requested at time $u-1$ is decached if p_c is in the cache (note that p_c is a child of p_r). As a consequence of this, LFE maintains the invariant that all the pages that are in the cache are on the path from the root to the last requested page. If the cache is not full just before p_r was requested then p_r is added to the cache. If the cache was full before p_r was requested, then LFE pretends that p_r is in the cache and selects a p_i in the cache that minimizes $e(p_i)$; in case of a tie, p_i is selected to the the page closest to the root. Note that it may be the case that $i = r$. The selected page p_i is then evicted and $e(p_i)$ is incremented.

For fixed n and k, let $\gamma = \gamma(n,k)$ be the smallest integer that satisfies $\gamma\binom{\gamma+k}{\gamma} = \gamma\binom{\gamma+k}{k} \geq n$. Observe that $(k+1)(\frac{\gamma}{k+1})^{k+1} \leq \gamma\binom{k+\gamma}{\gamma} \leq (k+1)(\frac{e(\gamma+k)}{k+1})^{k+1}$. For $k \leq \frac{1}{4}\log n$, we have $\gamma = \Theta(kn^{\frac{1}{k+1}})$. We will call a page *fat* if it has at least γ children, and otherwise we call the page *skinny*. Define $a(k,\ell)$ as the minimum over all trees of the number of distinct fat pages that must be requested before LFE, with a cache of size k, causes the eviction count of some page to reach $\gamma+\ell$. We now state some preliminary lemmas that are necessary for the analysis of LFE.

Lemma 12 If $1 \leq \ell \leq \gamma+1$ then $a(k,\ell) \geq \binom{k+\ell-1}{\ell-1}$.

Proof: We will prove the claim by induction on pairs (k,ℓ). For the base case we first show that the claim holds if $k=0$ or $\ell=1$. First consider the case that $\ell=1$. In this case we want to show that there is at least one fat node. If the eviction count of some page p_i reaches $\gamma+1$ then p_i must have at least γ children, and hence be fat. Now consider the case that $k=0$. Again we want to show that there is at least one fat node. If the eviction count of some page p_i reaches $\gamma+\ell$, then p_i has at least $\gamma+\ell-1$ children. Hence, p_i is fat since by the assumption that $\ell \geq 1$.

We show that if the claim holds for $a(k-1,\ell)$ and $a(k,\ell-1)$ then it holds for $a(k,\ell)$. Let p_r be the first page whose eviction count reaches $\gamma+\ell$. Consider the time u when p_r's eviction count was incremented to $(\gamma+\ell-1)$. This happened after some descendent of p_r (including possibly p_r) was requested. By the definition of $a(k,\ell-1)$, at least $a(k,\ell-1)$ distinct fat pages had been requested by time u.

Now we show that at time u it must be the case that no proper ancestor of p_r is in the cache. Assume to reach a contradiction that an ancestor p_g, $p_g \neq p_r$, of p_r is in the cache at time u. We break the argument into cases.

- In first case assume that $e(p_g) \geq \gamma+\ell$ at time u. This would contradict the assumption that p_r is the first page whose eviction count reaches $\gamma+\ell$.
- In the second case, assume that $e(p_g) = \gamma+\ell-1$ at time u. Let v be the time that $e(p_r)$ was incremented to $\gamma+\ell$. Then just before time v the page p_g must still be in the cache, or its eviction count would have reached $\gamma+\ell$ before p_r's count reached $\gamma+\ell$, contradicting the choice of p_r. But then, since LFE breaks ties by evicting the page closer to the root, p_g would be evicted instead of p_r at time v, which is a contradiction.
- In the final case, assume that $e(p_g) \leq \gamma+\ell-2$ at time u. But then since $e(p_r) = \gamma+\ell-2$ just before time u, and LFE breaks ties by evicting the page closer to the root, p_g would have been evicted at time u instead of p_r, which is a contradiction.

Now consider the time v when p_r is evicted and $e(p_r)$ is incremented to $\gamma+\ell$. Let T be the subtree of p_r that is being traversed at time v. Consider two successive evictions of p_r at times s and t. Then p_r must have been requested during the time interval $[s,t]$. As a result at time u the algorithm LFE could not yet have entered T. By the above argument p_r will be the lone page in cache when the root of T is first requested. Notice that from the time that T is first entered until time v, LFE behaves on T exactly as if T was the original tree and LFE had $k-1$ caches. Now consider the point in the algorithm LFE at time v when LFE is pretending to add the $(k+1)$st page to cache; call these $k+1$ pages $p_r, p_1, \ldots, p_k$, where the ordering is from the top of the tree to the bottom. We already know that $e(p_r) = \gamma + \ell - 1$ just before the eviction at time v occurs. Each of the other p_i, $1 \le i \le k$, must have $e(p_i) = \gamma + \ell - 1$ at this time because if the eviction count of p_i was larger it would contradict the choice of p_r, and if the eviction count of p_i was smaller it would contradict p_r getting evicted at time v. Hence, if LFE was given tree T as input it would evict p_1 at time u and increment $e(p_1)$ to $\gamma + \ell$. Hence, by induction T contains at least $a(k-1, \ell)$ fat nodes. Therefore,

$$a(k,\ell) \ \ge \ a(k,\ell-1) + a(k-1,\ell) \ \ge \ \binom{k+\ell-2}{\ell-2} + \binom{k+\ell-2}{\ell-1} \ = \ \binom{k+\ell-1}{\ell-1}$$

where the second inequality follows by induction. ∎

Lemma 13 For any $z \ge 1$, and for any rooted tree T with n nodes, there are at most $(n-1)/z$ nodes in T that have z or more children.

Proof: Note that the root is not a child of any node. Therefore there are at most $n-1$ children. Also note that each node is the child of at most one parent. Hence, there can be at most $\frac{n-1}{z}$ nodes with z or more children. ∎

Theorem 14 For Web Caching in the cost model, LFE is $(2\gamma+2)$-competitive, and hence, $\Theta(kn^{\frac{1}{k+1}})$-competitive.

Proof: We show that no page was missed more than $(2\gamma+2)$ times. If a page has x children then it is requested $x+1$ times. So a skinny page is requested at most γ times, and hence missed at most γ times.

Assume to reach a contradiction that there is a fat page p_i that was missed $2\gamma+2$ times. Then at some time u is must be the case that $e(p_i)$ is incremented to $2\gamma+1$. Let $F(u)$ be the number of fat pages seen by LFE up until time u. By the definition of $a(k,\ell)$, and by lemma 12,

$$F(u) \ge a(k, \gamma+1) \ge \binom{k+\gamma}{\gamma}$$

Hence,

$$\gamma F(u) \ge \gamma \binom{k+\gamma}{\gamma} \ge n$$

where the last inequality follows from the definition of γ. But by lemma 13, $F(u) \le (n-1)/\gamma$, or equivalently, $n-1 \ge \gamma F(u)$. Hence, we can conclude that $n-1 \ge F(u) \ge n$, which is a contradiction. ∎

Corollary 15 For Site Search in the cost model, the algorithm that uses LFE and depth first search is $O(kn^{\frac{1}{k+1}})$-competitive.

We now show that every oblivious online algorithm for Web Caching has competitive $\Omega(kn^{\frac{1}{k+1}})$-competitive. Since any page could have nonzero access time while all other pages have zero access time, an γ-competitive oblivious algorithm cannot miss any page more than γ times.

Theorem 16 For Web Caching in the cost model, every deterministic oblivious online algorithm is $\Theta(kn^{\frac{1}{k+1}})$-competitive.

Proof: Let A be an arbitrary oblivious online algorithm. We recursively define an adversarial strategy $\textsc{Bad}(k, \ell, A)$ to construct a tree T that has the property that:

- If A uses at most k caches during the traversal of T, then there must a node z in T that A misses on ℓ times.

Note that in this proof we do not consider first visit of a node to be a miss. Since the algorithm is oblivious, the result then follows if we set the access time of z to be very large and the access time of all other nodes to be 0.

We first consider the base cases. If $k = 0$ then the tree is a node with ℓ children. Hence, every algorithm with no cache will miss on the root ℓ times. If $\ell = 1$ then the tree is a line with $k + 2$ nodes. Then since A can only cache k of the top $k + 1$ nodes, some page will be missed at least once when the input sequence in traversing back up the tree.

We now consider the general inductive case. Let T_1 be the tree constructed by $\textsc{Bad}(k, \ell - 1, A)$. If A uses more than k cache locations to handle T_1 then T is set to T_1. Obviously T has the desired property since A uses more than k cache locations on $T = T_1$.

Otherwise, assume that A uses at most k cache locations while handling T_1. Then by induction there must be some page $p_r \in T_1$ that A misses on at least $\ell - 1$ times. Let T_2 be the tree constructed by $\textsc{Bad}(k - 1, \ell, A)$. The final tree T is then formed by making the root of T_2 the last child of p_r in T_1. We now argue that T has the desired property. If A used more than k cache locations to handle T then the desired property obviously holds. So assume that A used at most k cache locations to handle T. By induction we conclude that either A uses k caches to handle T_2, or there is a node in T_2 that A misses at least ℓ times. First consider the case that A uses k cache locations while traversing T_2. As a result, it must be the case that p_r is not in the cache when the depth first traversal returns from T_2 to p_r. Hence, p_r will experience its ℓth miss at this time, and the desired property holds. Now assume that A uses at most $k - 1$ cache locations to traverse T_2. Then by induction, A must miss some node ℓ times in T_2. Again the desired property for T holds.

The remainder of the proof is very similar to Lemma 12. Define $b(k, \ell)$ to be the number of distinct pages in the tree T constructed by $\textsc{Bad}(k, \ell, A)$. We will now show by induction that

$$b(k, \ell) \leq \binom{k + \ell + 1}{\ell}.$$

Note that by construction it is the case that $b(k, 1) = k + 2 = \binom{k+2}{1}$, and $b(0, \ell) = \ell + 1 = \binom{\ell+1}{\ell}$. Also by construction,

$$\begin{aligned} b(k, \ell) &\leq b(k, \ell - 1) + b(k - 1, \ell) \\ &\leq \binom{k + \ell}{\ell - 1} + \binom{k + \ell}{\ell} \\ &\leq \binom{k + \ell + 1}{\ell} \end{aligned}$$

We now want to find the maximum $\ell - 1$ such that $b(k, \ell - 1) \leq n$. Instead it suffices to find the minimum ℓ such that $b(k, \ell) > n$. In other words

$$n < \binom{k + \ell + 1}{\ell}$$

$$< \left(\frac{e(k+\ell+1)}{k+1} \right)^{k+1}$$

Observe that solving this inequality gives that $\ell = \Omega(kn^{1/(k+1)})$ for $k \leq \frac{\log n}{2}$. ■

References

[1] S. Albers, S. Arora, and S. Khanna, "Page replacement for general caching problems", ACM/SIAM Symposium on Discrete Algorithms, 31–40, 1999.

[2] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, "Competitive paging with locality of reference", *Journal of Computer and System Sciences* **50**, 244–258, 1995.

[3] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms", USENIX Symposium on Internet Technologies and Systems, 193–206, 1997.

[4] L. Catledge and J. Pitkow, "Characterizing browsing strategies in the world wide web", *Computer Networks and ISDN Systems* **27**, 1065-1073, 1995.

[5] E. Cohen, and H. Kaplan, "Caching documents with variable sizes and fetching costs: an LP based approach", ACM/SIAM Symposium on Discrete Algorithms, S879–S880, 1999.

[6] R.E. Ladner, J.D. Fix, and A. LaMarca, "Cache performance analysis of traversal and random accesses", ACM/SIAM Symposium on Discrete Algorithms, 613–622, 1999.

[7] B. Huberman, P. Pirolli, J. Pitkow, and R. Lukose, "Strong regularities in world wide web surfing", *Science* **280**, 95–97, 1998.

[8] S. Irani, "Page replacement with multi-size pages and applications to web caching", ACM Symposium on Theory of Computing, 701–710, 1997.

[9] B. Jiang, "DFS-traversing graphs in a paging environment, LRU or MRU", *Information Processing Letters* **40**, 193–196, 1991.

[10] B. Kalyanasundaram and K. Pruhs, "Constructing competitive tours from local information", *Theoretical Computer Science* **130**, 125–138, 1994.

[11] A. Karlin, S. Phillips, and P. Raghavan, "Markov paging", IEEE Symposium on Foundations of Computer Science, 208–217, 1992.

[12] L. Tauscher and S. Greenberg, "How people revisit web pages: empirical findings and implications for the design of history systems", *International Journal of Human-Computer Studies* **47**, 97–137, 1997.

[13] N. Young, "The k-server dual and loose competitiveness", *Algorithmica* **11**, 525–541, 1994.

[14] N. Young, "On-line file caching", ACM/SIAM Symposium on Discrete Algorithms, 82–86, 1998.