Competitive Caching of Query Results in Search Engines

Ronny Lempel *

IBM Research Labs, Haifa 31905, Israel

Shlomo Moran

Department of Computer Science, The Technion, Haifa 32000, Israel

Abstract

We study the problem of caching query result pages in Web search engines. Popular search engines receive millions of queries per day, and for each query, return a result page to the user who submitted the query. The user may request additional result pages for the same query, submit a new query, or quit searching altogether. An efficient scheme for caching query result pages may enable search engines to lower their response time and reduce their hardware requirements.

This work studies query result caching within the framework of the competitive analysis of algorithms. We define a discrete time stochastic model for the manner in which queries are submitted to search engines by multiple user sessions. We then present an adaptation of a known online paging scheme to this model. The expected number of cache misses of the resulting algorithm is no greater than 4 times the expected number of misses that any online caching algorithm will experience under our specific model of query generation.

1 Introduction

1.1 The need and promise of caching search results

Popular search engines receive millions of queries per day on any and every walk of life. While these queries are submitted by millions of unrelated users,

* corresponding author

Email addresses: rlempel@il.ibm.com (Ronny Lempel), moran@cs.technion.ac.il (Shlomo Moran).

studies have shown that a small set of popular queries accounts for a significant fraction of the query stream. These statistical properties of the query stream seem to call for the caching of search results. Indeed, such caching was noted in Brin and Page's description of the prototype of the search engine *Google* [1] as an important optimization technique of search engines [1]. An engine that answers many queries from a cache instead of processing them through its index, can lower its response time and reduce its hardware requirements.

Several works report on experiments with caching query results. In [2], Markatos used a log containing a million queries submitted to the Excite [2] search engine to drive simulations of query result caches. He used four replacement policies - LRU (Least Recently Used) and three variations, demonstrating that warm, large caches of search results can attain hit ratios of close to 30%. Saraiva et al. [3] proposed a two-level caching scheme that combines caching query results and inverted lists. The replacement strategy they adopted for the query result cache was LRU. They experimented with logged query streams, testing their approach against a system with no caches. Overall, their combined caching strategy increased the throughput of the system by a factor of three, while preserving the response time per query.

In addition to storing results of submitted queries in the cache, search engines may also *prefetch* results that they predict to be requested shortly. An immediate example is prefetching the second page of results whenever a new query is submitted by a user. In [4], a log containing over seven million queries submitted to the search engine AltaVista [3] was used to test integrated schemes for the caching and prefetching of search results. Hit ratios exceeding 50% were achieved. Prefetching of results proved to be of major importance, doubling the hit ratios of small caches and increasing those of larger caches by more than 50%. The prefetching of search results was also examined in [5], albeit from a different angle: the objective was to minimize the computational cost of serving search sessions rather than to maximize the hit ratio of the results cache.

1.2 Online paging and caching algorithms

Paging is a classic problem that has been studied extensively in the context of *competitive analysis* of algorithms. The classic formulation of the problem defines a system with two levels of memory: a fast memory (a cache) of size k pages, and a slow memory of $N \gg k$ pages that represents the entire address space. A paging algorithm (cache replacement policy) ALG is presented with

[1] http://www.google.com/
[2] http://www.excite.com
[3] http://www.altavista.com

a sequence of requests for memory pages, and must service the requests *online* by having each requested page in the cache prior to seeing (and serving) the following requests. When a request for an uncached page arrives, a *page fault* (or *cache miss*) occurs, and ALG must evict a cached page in order to vacate space for serving the request. Paging algorithms differ from each other by the policy they apply when choosing which cached page to evict upon a fault. The cost of serving a request sequence σ with a (deterministic) paging algorithm ALG, denoted by $ALG(\sigma)$, is the number of page faults that ALG encounters while serving σ.

The difficulty of paging stems from the online nature of the problem. Had the entire sequence of requests been known in advance, an optimal policy is to always evict the cached page whose next request is the farthest away. This optimal *offline* algorithm, proposed by Belady [6], will be denoted by OPT. Following Sleator and Tarjan [7], paging algorithms are commonly evaluated using competitive analysis. A paging algorithm ALG is said to be *c-competitive* if there exists a constant α, such that

$$\forall\sigma, \quad ALG(\sigma) < c \cdot OPT(\sigma) + \alpha \ .$$

Sleator and Tarjan showed that the competitive ratio of any deterministic online paging algorithm is at least the cache size, k. They also showed that the FIFO and LRU (least recently used) paging policies both achieve a competitive ratio of k, and so attain the optimal competitive ratio for the problem. However, LRU usually outperforms FIFO in many practical settings [8]. This stems from the fact that the request streams that are generated by many real-life applications exhibit a large degree of *locality of reference*: after a page p is requested, p and a small set of related pages are likely to be requested in the near future. In order to model reality more closely, many researchers have examined the paging problem for non-arbitrary request sequences that exhibit some degree of locality of reference. Another expansion of the classic formulation that has attracted researchers deals with multi-user settings, where the system is presented with several request streams that are generated concurrently. These efforts are detailed in Section 3. See [9] for a comprehensive study of competitive analysis and online algorithms, and [10] for a survey of the paging problem.

1.3 This work

This work examines the caching of search results within the theoretical framework of competitive analysis. We model search engine query streams that are created by concurrent sessions of multiple clients. We then adapt a known

online paging algorithm to this setting in a manner that preserves its competitiveness.

We start in Section 2 by surveying some statistical properties that have been observed in search engine query streams. Section 3 reviews online paging algorithms which handle multiple request streams and streams that exhibit some form of locality of reference. We recount in special detail the paging algorithm of Lund et. al [11], that deals with request streams that are generated by a distribution.

Section 4 presents a spectrum of theoretical models for the problem of caching search result. We discuss several issues that are encountered when transforming the complex reality of the problem into concrete, abstract models. Section 5 narrows the discussion to one concrete model, which is subsequently analyzed by an adaptation of the algorithm of [11]. The analysis shows that the algorithm is both efficient and competitive for that model. Concluding remarks and suggestions for future research are brought in Section 6.

2 Locality of Reference in Search Engine Query Logs

Several studies have analyzed the manner in which users query search engines and view result pages [2,4,12,13]. These reports shed light on the locality of reference that is found in search engine query logs, such as the distribution of query popularities, and the number of result pages that users view per *search session*.

Search sessions start when users submit *initial* queries to search engines, by typing some search phrase which describes their topic of interest. From the user's point of view, an engine answers each initial query with a linked set of ranked result pages, typically with 10 results per page. All users browse the first page of results, and some users scan additional result pages of the list, usually in the natural order in which those pages are presented. A search session implicitly terminates when the user decides not to browse additional result pages on the topic which initiated the session. From the engines' point of view, users that request additional result pages, beyond the first page of results, are actually submitting *follow-up* queries. Such queries retain the search phrase of the original query, and specify the number of result page that is requested. Throughout the remainder of this paper, we will use the term *topic* to refer to a search phrase, while the term *query* will refer to a request for a specific result page of a specific topic.

Topic and query popularities

The number of distinct information needs of users is very large. Silverstein et al. [13] analyzed a log of a billion queries submitted to AltaVista over a period of six weeks, and reported that almost two thirds of the queries appeared only once in the log. However, popular queries were repeated many times: the 25 most popular queries found in their log accounted for 1.5% of the total submissions. The findings of Markatos [2], who analyzed a log of about a million queries submitted to Excite, are consistent with the later figure - the 25 most popular queries in the Excite log accounted for 1.23% of the submissions. Markatos also found that many successive submissions of queries on the same topic appear in close proximity (are separated by a small number of other queries in the query log).

In [4], a log of over 7 million queries submitted to AltaVista was examined. 67% of all topics were only requested once, with the corresponding figure for queries being 48%. However, the 50 most popular queries accounted for almost 2% of the log. In general, both topic and query popularities were found to obey power-law distributions: the number of topics[queries] that were requested n times in the log was proportional to n^{-c} (c was about 2.8 for queries, 2.4 for topics) [4].

Structure of search sessions

The studies cited above report that users browse through very few result pages during search sessions, and that the "deeper" the result page, the less users view it. The exact distributions are different in each study, but all four reports agree that at least 58% of the users view only the first page (the top-10 results), at least 15% of the users view more than one page of results, and that no more than 12% of users browse through more than 3 result pages. Although not many users venture deeply into the set of result pages, statistical data in the order of 10^{-2} are powerful predictors for the purpose of caching result pages.

3 Beyond the Classic Model of Paging

We discussed the classic formulation of the paging problem and its competitive ratio in the Introduction. This section surveys additional results on variations of the problem.

[4] The number of extremely popular topics and queries exceeded the value predicted by the power law distributions.

We first note that randomized paging algorithms can achieve a better competitive ratio for the classic problem. A randomized paging algorithm ALG is said to have a competitive ratio of c if, for every request stream σ,

$$E[\ ALG(\sigma)\ ]\ <\ c \cdot OPT(\sigma) + \alpha$$

where the expectancy is with respect to the random choices of ALG. Fiat et al. [14] showed that all randomized paging algorithms have a competitive ratio of at least $\mathcal{H}_k$ (the k'th harmonic number), while McGeoch and Sleator presented an algorithm that attains the lower bound [15].

Modeling locality of reference

The *access graph* model was proposed by Borodin et al. [16] to capture the locality of reference in streams of page requests. Given a graph G, whose nodes correspond to the possible N pages of memory, request sequences in the access graph model describe paths in G. Every graph G and paging algorithm A implicitly define a competitive ratio $c_A(G)$. Given G, let $c(G)[c^R(G)]$ denote the infimum, over all deterministic[randomized] paging algorithms of $c_A(G)$. Thus, $c(G)[c^R(G)]$ is the best competitive ratio that any deterministic[randomized] algorithm can hope to achieve on G. Borodin et al. presented a universal deterministic algorithm that was later shown to be $\mathcal{O}(c(G))$-competitive on any undirected access graph G. Fiat and Karlin presented the randomized counterpart in [8].

Locality of reference may also be modeled by request sequences that are generated by distributions. Karlin et al. [17] suggested a model based on Markov chains, in which request sequences correspond to random walks on the chain. Each state of the chain corresponds to one of the N pages of memory, and the transition probability from state i to state j corresponds to the probability of requesting page j immediately after requesting page i. Given a Markov chain M, the goal is to define an algorithm whose expected cost on randomly generated request sequences is minimal. The authors use results from Markov decision theory to deduce that, for every chain M, there exists an optimal online paging policy. Furthermore, this policy is deterministic, time-invariant and memoryless, meaning that eviction decisions are determined by the k cached pages and requested page at each step. However, computing the optimal policy requires solving a linear program in $N\binom{N}{k}$ variables, which, in general, is exponential in k. They then present an algorithm whose expected cost (for any Markov chain M) is no more than $c \times f(M)$, where c is a constant and $f(M)$ is the expected cost of the optimal policy on sequences generated by M. This algorithm requires a preprocessing step which is polynomial in N, and further performs $\mathcal{O}(k)$ calculations per fault. Lund et al. [11] considered request sequences that are generated by a broader set of distributions.

These distributions, and a suitable uniform randomized paging algorithm, are covered in Section 3.1.

Multiple, interleaving request streams

In many realistic settings, and certainly in the context of search engines, paging and caching systems are faced with multiple, independent streams of requests. Alborzi et al. [18] defined and analyzed a model where the paging algorithm is faced with requests originating from u clients. At every moment, the algorithm sees the next pending request of each client; in other words, it sees a "front" of u pending requests. The algorithm may serve any of those u requests at each step. This model did not address locality of reference in the requests generated by each client. Fiat and Karlin [8] addressed multiple request streams in the access graph model by considering u *pointers*, each pointing to a node of the graph. A request is generated by moving any of the pointers from its node to a neighboring node. They presented a universal deterministic paging algorithm for undirected access graphs with u pointers, and proved that its competitive ratio is $\mathcal{O}(c_u(G))$, where $c_u(G)$ is the straightforward adaptation of $c(G)$ (defined in the earlier discussion of the basic access graph model) to the u-pointer scenario. Multiple, interleaving request streams can also be integrated into the Markov paging model. When u streams are generated independently by the same chain, and when each request of the combined stream is chosen independently and uniformly from one of the streams, the resulting behavior can be described by a Markov chain with $\binom{u+N-1}{u}$ states.

Lookahead, and reordering of requests

High-volume Web servers such as search engines receive requests from many users concurrently. Unlike requests that are generated by a serial program, where request $i+1$ cannot be issued before request i is served, requests of different clients to Web servers arrive independently of each other. This enables the caching algorithm to examine some future portion of the request sequence while replacing entries, and even to serve requests not necessarily in FIFO order (provided that the delay experienced by every request is sufficiently small). As noted above, in [18] the paging algorithm has knowledge of the pending request of each client. Breslauer [19] defined the notion of paging algorithms with ℓ-*natural lookahead*. Such algorithms may see a prefix of future requests until $\ell+1$ uncached page requests are encountered (whenever they serve an uncached page, they see future requests that extend until ℓ other uncached pages have appeared). He presented an optimal deterministic paging algorithm for this variant, whose competitive ratio is $\frac{k+\ell}{\ell+1}$. Albers [20] suggested a different lookahead model, called *strong lookahead*. Whenever an algorithm with a strong lookahead of ℓ serves an uncached page p, it sees future requests

that extend until ℓ distinct pages (that are also different from p) appear. Note that the lookahead is independent of the algorithm in this model. For $\ell \leq k-2$, Albers presented a deterministic algorithm that is $k-\ell$ competitive, and proved it to be optimal. She then presented a nearly-optimal algorithm, whose competitive ratio is $k-\ell+1$, which does not exploit the full power of the allowed lookahead; rather, it serves the requests in non-overlapping *blocks*, where each block (except, perhaps, the last) contains exactly $\ell+1$ distinct requests. The algorithm has the property that after serving the last request of each block, all of that block's requests are cached. Thus, the internal order of the requests in each block is of no consequence, and the algorithm may be thought of as serving an entire block of requests "at once". Recently, Feder et al. [21] applied competitive analysis to paging algorithms that allow reordering of requests, and may enjoy additional lookahead capabilities. Note that reordering itself implicitly involves lookahead. Their results cover both deterministic and randomized algorithms.

3.1 Paging against a distribution

This section recounts the definitions and algorithm presented in [11] for paging against a distribution. The algorithm assumes that a stream of page requests is generated by some distribution $\mathcal{D}$. The only assumption about $\mathcal{D}$ is that whenever a page must be evicted from fast memory, it is possible to compute for every two pages a and b in fast memory, the probability $\mathcal{P}(a,b)$ that the next request for a occurs before the next request for b. Since the algorithm handles requests sequentially, $\mathcal{P}(a,b)+\mathcal{P}(b,a)=1$ for all $a \neq b$.

Let $\mathcal{C}$ denote the set of pages in fast memory when an uncached page is requested (and a page in $\mathcal{C}$ must be evicted). The above probabilities define a *weighted tournament* on $\mathcal{C}$.

Definition 1 *A weighted tournament is a set $\mathcal{C}$ and a function $\mathcal{P}: \mathcal{C} \times \mathcal{C} \to [0,1]$ so that for every two distinct elements $a, b \in \mathcal{C}$, $\mathcal{P}(a,a)=0$ and $\mathcal{P}(a,b)+\mathcal{P}(b,a)=1$.*

After calculating the weighted tournament for the pages in $\mathcal{C}$, the algorithm proceeds to calculate a *dominating distribution* for the tournament.

Definition 2 *A dominating distribution in a weighted tournament $(\mathcal{C},\mathcal{P})$ is a probability distribution $\mathcal{Q}$ on $\mathcal{C}$ such that for every $a \in \mathcal{C}$, the expectation*

$$\sum_{b \in \mathcal{C}} \mathcal{P}(b,a)\mathcal{Q}(b) \leq \frac{1}{2} \ .$$

Lund et al. prove in [11] that a dominating distribution exists for every weighted tournament, and show how to calculate the distribution $\mathcal{Q}$ in a time that is polynomial in $|\mathcal{C}|$ (the size of the fast memory). Their algorithm then chooses a page according to $\mathcal{Q}$, and evicts it from fast memory.

Let $\mathcal{O}N$ denote any online paging algorithm against the distribution $\mathcal{D}$. The algorithm described above is proved to be 4-competitive against $\mathcal{O}N$: the expected number of page faults of the proposed algorithm is at most 4 times the expected number of faults of $\mathcal{O}N$. In Section 5.4 we bring a modification of the algorithm and its analysis, adapted to our needs.

4 From the Practical Problem to a Theoretic Model

The problem of caching query result pages is a natural problem that arises when designing a large scale search engine that is expected to accommodate millions of queries per day. We now present some of the issues that must be tackled when modeling this problem. We begin by formalizing the notions of queries and topics, introduced in Section 2: a *query* will refer to an ordered pair (t, k) where t is the *topic* of the query (the search phrase that initiated the session), and $k \geq 1$ is the number of result page requested. For example, a $(t, 2)$ query will denote the second page of results (which typically contains results $11 - 20$) for the topic t.

Structure of search sessions As discussed in Section 2, a search session begins when a user issues an initial $(t, 1)$ query. The user then browses several result pages, usually in the natural order of the pages. Indeed, it has been observed that the percentage of users that view result page (t, k) diminishes as k grows [13,12,4]. Search engines, however, allow users to browse through result pages in a less strict manner. Upon viewing a page (t, k), all engines allow users to submit a query for result page $(t, k+1)$ [5] . Whenever $k > 1$, the engines also allow users to retract and request page $(t, k - 1)$. Some engines allow users to request pages in a more random fashion, jumping quite arbitrarily from one result page to another.

One simple model of search sessions, which will be adopted in Section 5.1, is the *forward viewing* model: the series of result pages of every topic $t \in T$ form the states of a Markov chain. A user viewing result page (t, k) will either proceed to view $(t, k + 1)$ with a certain predetermined probability, or will quit viewing result pages (with the remaining probability). This geometric modeling of user behavior has also been applied in [4,5,22].

[5] provided that the engine has more results to offer for the query

Activation and termination of search sessions In Section 3 we surveyed previous work on paging in the presence of multiple, interleaving request sequences [18,8,17]. Those works have all assumed that the number of interleaving sequences is constant and known to the algorithm. Search engines, however, face an ever changing number of *active* search sessions. Users activate sessions by submitting $(t, 1)$ queries at random. They then view some number of result pages for topic t at their leisure, and terminate the session (which becomes inactive) once they are satisfied (or hopelessly dissatisfied) with the results. While the engines can track the session activations (they receive the $(t, 1)$ queries), the engines cannot know when users decide to terminate the sessions. Thus, not only does the number of sessions generating requests vary in a chaotic manner, the engines cannot even keep track of the number of active sessions at each moment.

A model of session activation should decide the rate in which users will initiate sessions, and the topics that the users will query on. Not all query topics are equally popular, and thus it is reasonable that each topic should have its own session activation rate. Furthermore, the model should decide whether the topic-specific activation rates are stationary or dynamic: the popularity (or lack thereof) of topic t may be constant, but t may be a trendy topic whose popularity fluctuates with respect to current events or other temporal issues.

Session termination is governed by the modeling of the search session. However, since users do not "notify" the engine when terminating their sessions, the engine should assume that sessions which have been inactive (did not generate queries) over long periods of time, have been terminated. Implicitly, this requires some modeling of the timespan between successive query generations in each session. Must active sessions generate a query once per time period? Once in every window of queries? According to some probability distribution of inter-generation times? A concrete model should resolve this.

Synchronization and ordering of user requests With many active sessions generating queries concurrently, the model must consider the order in which the various queries arrive to the system.

Pure online models will have queries arrive sequentially, and have the caching algorithm serve each query prior to receiving the next one. There are a couple of natural alternatives for merging multiple request streams into a fully ordered request sequence: the requests might be ordered arbitrarily, as if by an adversary, or they may be ordered by some underlying stochastic process. However, as noted in Section 3, internet servers that handle multiple, concurrent client sessions may have the privilege of serving requests not in the order of their arrival, and may thus enjoy some lookahead capabilities. Specifically, the model may allow for the arrival of queries in *batches*, where groups of

queries are presented to the system simultaneously. Batched arrivals are particularly applicable when the model assumes discrete time steps, since many sessions may generate queries at the same time unit τ [6]. When requests arrive in batches, a natural requirement is that all pages requested in a batch must reside simultaneously in the cache prior to the arrival of the next batch [7]. Implicitly, treatment of batched arrivals involves aspects of both lookahead and reordering of requests, since the system is allowed to see many unordered requests at once. However, this luxury is partially offset by the requirement that all the requests in a batch be cached prior to the arrival of the next batch. This requirement is more stringent than in sequential paging settings, where each request must only be cached momentarily upon its treatment.

Association of sessions with requests The discussion so far assumed that upon receiving a request, the server can identify the session which generated the request. This assumption enables the server to keep track of the set of active sessions and of the specific page that each session is currently viewing. While the assumption is technologically sound in the context of search engines (engines may provide each session with a session id, or use the cookie mechanism to couple users with actions), it is also possible to consider the problem of treating *anonymous* streams, where requests are not associated with the generating sessions.

An interesting model that addresses many of these issues has been proposed by Kraiss and Weikum [23]. Their model is based on continuous-time Markov chains, whose states represent both the pages of the system and potential user sessions. The model supports general transition patterns between states, and can thus model arbitrary search session structures. It also allows for dynamic session arrivals and departures. As the model is time-continuous, it implies that the requests arrive sequentially, as governed by the stochastic process.

Kraiss and Weikum suggested a caching heuristic based on their model. Roughly speaking, they proposed to prioritize cached pages according to the expected number of requests that each page will have within a certain time horizon. Their work did not involve competitive analysis; rather, they proceeded to experimentally evaluate several flavors of that heuristic.

[6] Batched-arrival models may limit each session to one query per batch, thus maintaining a strict FIFO order between the requests of each session.

[7] This requirement is similar to the property maintained in the block-based algorithm of [20], see Section 3.

5 A Theoretical Model with a Provably Competitive Algorithm

This section considers a concrete (theoretical) model for the manner in which queries are presented to search engines. For this model we prove that the result-page cache can be managed by a competitive online algorithm.

Section 5.1 explains the model of query generation. In Section 5.2, we show how to efficiently compute, for a given time τ and every two distinct pages a, b, the probability $\mathcal{P}_\tau(a, b)$ that the first post-τ request of page a will occur no later than the first post-τ request of page b. Therefore, we can adapt the online algorithm of [11] (see Section 3.1) and achieve a caching strategy whose expected page fault rate is at most 4 times that of any online caching strategy. Section 5.3 presents the adaptation of the algorithm, and Section 5.4 brings its analysis.

5.1 The model

We propose a discrete time probabilistic model, in which anonymous queries arrive in batches. The batch of queries arriving at time τ will be denoted by B_τ. B_τ is a multiset of queries, since certain queries may be submitted by more than one search session. We thus denote by B_τ^U the set of *distinct* queries in B_τ. The caching algorithm examines each batch of queries in full, and then updates the cache. Let k be the size of the cache, and let $\mathcal{C}_\tau$ denote the set of cached result pages right before the arrival of B_τ. The requirement is that just before the arrival of $B_{\tau+1}$, all pages requested in B_τ are cached. Formally, using the above notations, we require that $B_\tau^U \subseteq \mathcal{C}_{\tau+1}$. Implicitly, this requires that for all $\tau, |B_\tau^U| \leq k$.

B_τ is composed of follow-up queries submitted by users that are in the midst of search sessions, and by initial queries submitted by newly activated sessions. All sessions who contribute to B_τ will be called τ-*active* sessions. We first expand on the activation of new sessions.

Let T denote the (possibly infinite) set of all query topics. New sessions are activated at independent, topic-specific Poissonic rates as follows. With every topic $t \in T$, we associate a positive real number λ_t. The number of initiations of t-sessions (requests of page $(t, 1)$) in batch B_τ will be denoted by the random variable $U_{t,\tau} \sim \text{Pois}(\lambda_t)$.

The submission of follow-up queries is modeled using the forward-viewing behavior. For simplicity, and as is often done in search engines, we limit the number of result pages that users may browse per session to s. Thus, for each $t \in T$, users may request result pages $(t, 1), \ldots, (t, s)$. A user that requested

$(t, m), m < s$ at time τ, will follow-up and request $(t, m+1)$ at time $\tau+1$ with probability p, or terminate the session with probability $1-p$. Terminating users will not notify the system of their decision, and it will be up to the algorithm to keep track of the set of active sessions. The actions in different sessions are independent of each other.

Note that this probabilistic model may generate batches of requests which violate the restriction that $|B_\tau^U| \leq k$. We assume that k is large enough so that the probability of such events is sufficiently low. The rest of the discussion addresses request sequences that respect the restriction.

Implications of the model

The model implies that each session contributes at most one query to each batch, and that the paging algorithm, when serving B_τ, sees the requests of all τ-active sessions. The latter is similar to the lookahead allowed in the model of Alborzi et al. [18].

The identical (and independent) properties of different same-topic sessions imply that the paging algorithm needs only to track the *number* of active sessions that are viewing every result page. Since the model requires active sessions to generate a follow-up query in every batch of their active life, the system can track that number for each page at all times. Thus, the fact that the queries in this model are anonymous does not hinder the paging algorithm in any way. Furthermore, the memory overhead that is incured by tracking these numbers is linear in k, the size of the cache. Recall that the model requires all the queries of B_τ^U to be cached prior to time $\tau+1$. As there are no active sessions that are viewing pages other than those requested in B_τ^U, the algorithm needs only to associate a natural number $u_\tau(t, m)$ with each page $(t, m) \in \mathcal{C}_{\tau+1}$, noting the number of requests for page (t, m) in B_τ. This is also the number of sessions that may potentially request page $(t, m+1)$ during $B_{\tau+1}$.

5.2 Calculating $\mathcal{P}_\tau(a, b)$ for all pairs of pages

This section shows that at any time τ when the cache must be refreshed ($B_\tau^U \not\subseteq \mathcal{C}_\tau$), it is possible to compute $\mathcal{P}_\tau(a, b)$ for every two cached pages a, b, in time which is polynomial in $ks \cdot \log |B_\tau|$. The computation depends solely on the contents of the cache at time τ, as $\mathcal{C}_\tau$ implies the status of the active sessions, while the model for the activation of new sessions is stationary. Note that the sum $\mathcal{P}_\tau(a, b) + \mathcal{P}_\tau(b, a)$ may exceed 1, since both pages may be requested simultaneously (in the same future batch). Throughout this section, let τ denote the current time.

We first observe that for two independent geometric random variables $X \sim \mathcal{G}(p),\ Y \sim \mathcal{G}(q)$,

$$Pr[X < Y] = p(1-q) + (1-p)(1-q)Pr[X < Y]$$

and so

$$Pr[X < Y] = \frac{p(1-q)}{p+q-pq} \tag{1}$$

We now state two simple propositions regarding the probability $p_i(x)$, of a result page $x = (t_x, n_x)$ being requested in batch $\tau + i$ by at least one session. Formally,

$$p_i(x) = Pr[\ x \in B^u_{\tau+i}\ ]$$

Proposition 1 *Let x be the page (t_x, n_x). For all $i \geq n_x$,*

$$p_i(x) = 1 - e^{-\lambda \cdot p^{n_x - 1}}$$

where $\lambda = \lambda_{t_x}$ is the Poissonic activation rate of new sessions on topic t_x, and p is the probability for proceeding to view the next result page in a session.

Proposition 1 follows from basic properties of Poisson processes [24]. The next proposition is immediate.

Proposition 2 *Let x be the page (t_x, n_x). For all $1 \leq i < n_x$,,*

$$p_i(x) = 1 - (1-p^i)^{u_\tau(t_x, n_x - i)}$$

where $u_\tau(t, n)$ denotes the number of sessions that have requested page (t, n) at time τ (in batch B_τ).

Let $a, b \in \mathcal{C}_\tau$ be the pages (t_a, n_a) and (t_b, n_b), respectively. Define the following two events for all $i > 0$:

(1) $E_i^{a<b}$: When considering only requests of batches $\tau + i$ and beyond, the next request for page a precedes the next request for page b.
(2) $E_i^{a=b}$: When considering only requests of batches $\tau + i$ and beyond, the next request for page a and the next request for page b occur at the same batch.

Note that calculating $Pr[E_1^{b<a}]$ will allow us to derive $\mathcal{P}_\tau(a, b)$:

$$\mathcal{P}_\tau(a,b) = Pr[(E_1^{a<b} \cup E_1^{a=b})] = Pr[E_1^{a<b}] + Pr[E_1^{a=b}] = 1 - Pr[E_1^{b<a}] \tag{2}$$

The case where $t_a \neq t_b$

The activations of search sessions on different topics are independent of each other. We rely on this independence in the following proposition, which presents a closed-form expression for $Pr[E_n^{a<b}]$ where $n = \max\{n_a, n_b\}$.

Proposition 3 *Let $\alpha \triangleq \lambda_{t_a}$, the activation rate of new sessions on topic t_a. Similarly, let $\beta \triangleq \lambda_{t_b}$. For $n \geq \max\{n_a, n_b\}$,*

$$Pr[E_n^{a<b}] = \frac{p_n(a)(1-p_n(b))}{p_n(a)+p_n(b)-p_n(a)p_n(b)} = \frac{(1-e^{-\alpha p^{n_a-1}})e^{-\beta p^{n_b-1}}}{1-e^{-\alpha p^{n_a-1}-\beta p^{n_b-1}}}$$

Proof: Let $n \geq \max\{n_a, n_b\}$. By Proposition 1, for all $i \geq n$, $p_i(a) = p_n(a)$ and $p_i(b) = p_n(b)$. Let $X[Y]$ be the smallest non-negative integer j for which at least one session requests page $a[b]$ at time $n+j$. It is easy to see that $X[Y]$ is a geometric random variable with parameter $p_n(a)[p_n(b)]$. Therefore,

$$Pr[E_n^{a<b}] = Pr[X < Y] \ .$$

Substituting the expressions of $p_n(a), p_n(b)$ from Proposition 1 in Equation 1 yields the result. □

Having computed $Pr[E_n^{a<b}]$, we can proceed to recursively compute $Pr[E_1^{a<b}]$ by the following equation:

$$\begin{aligned} Pr[E_i^{a<b}] &= p_i(a)[1-p_i(b)] \ + \ [1-p_i(a)][1-p_i(b)]Pr[E_{i+1}^{a<b}] \\ &= [1-p_i(b)]\left(p_i(a) + [1-p_i(a)]Pr[E_{i+1}^{a<b}]\right) \end{aligned} \tag{3}$$

Propositions 1-3 and Equations 2-3 enable the calculation of $\mathcal{P}_\tau(a,b)$ for any pair of different-topic pages. The complexity of each such computation is $\mathcal{O}(s \log |B_\tau|)$.

The case where $t_a = t_b$

Assume w.l.o.g. that $n_a > n_b$. The forward-viewing model implies that only active sessions already at or beyond page b at time τ, may generate post-τ requests for page a without first generating post-τ requests for page b. Therefore, if page a is not requested in $(\tau+1), \ldots, (\tau+n_a-n_b)$, page b will surely

be requested prior to the next request of a. Denoting the difference $n_a - n_b$ by d, this argument yields that substituting

$$Pr[E_{d+1}^{a<b}] \leftarrow 0,\ Pr[E_{d+1}^{b<a}] \leftarrow 1$$

in the calculation described for different-topic pages (while leaving the values of $E_i^{a<b}$ and $E_i^{b<a}$ for all $i \leq d$ as described there) will enable the calculation of $\mathcal{P}(a,b)$ and $\mathcal{P}(b,a)$.

Combining this with the discussion above, we conclude that calculating $\mathcal{P}_\tau(a,b)$ for all cached pairs can be achieved in $\mathcal{O}(k^2 s \log |B_\tau|)$.

5.3 The caching algorithm

Recall that $\mathcal{C}_\tau$ is the set of cached result pages just before the arrival of batch B_τ. Let $\{s_1, \ldots, s_m\} \stackrel{\triangle}{=} B_\tau^U \setminus \mathcal{C}_\tau$ be the set of cache misses (or faults) at time τ, which force the caching algorithm to evict m pages of $\mathcal{C}_\tau \setminus B_\tau^U$. These pages are chosen as follows:

(1) $\mathcal{P}_\tau(a,b)$ is calculated for every two pages $a, b \in \mathcal{C}_\tau \setminus B_\tau^U$. These probabilities are then normalized to form the set of probabilities $\bar{\mathcal{P}}_\tau(a,b)$ as follows:

$$\bar{\mathcal{P}}_\tau(a,b) = \begin{cases} \mathcal{P}_\tau(a,b) - \frac{\mathcal{P}_\tau(a,b)+\mathcal{P}_\tau(b,a)-1}{2} & a \neq b \\ 0 & a = b \end{cases}$$

Consider two distinct pages $a \neq b$. The normalization ensures that $\bar{\mathcal{P}}_\tau(a,b) + \bar{\mathcal{P}}_\tau(b,a) = 1$. Furthermore, Equation 2 implies that $\bar{\mathcal{P}}_\tau(a,b) = \mathcal{P}_\tau(a,b) - \frac{1}{2} Pr[E_1^{a=b}]$. Thus, $Pr[E_1^{a<b}] \leq \bar{\mathcal{P}}_\tau(a,b) \leq \mathcal{P}_\tau(a,b)$.

(2) For $i = 1, \ldots, m$:
   (a) Calculate the dominating distribution (see Section 3.1) over the set of pages that are candidates to be evicted from the cache (cached pages that are not in B_τ^U), using the probabilities $\bar{\mathcal{P}}_\tau(\cdot,\cdot)$.
   (b) Store s_i while evicting a page chosen randomly according to the dominating distribution.

As argued earlier in this section, step 1 can be achieved in $\mathcal{O}(k^2 s \log |B_\tau|)$. Each iteration of step 2 involves preparing (and solving) a linear program whose size is $\mathcal{O}(k^2)$[11]. The complexity of each iteration is thus polynomial in k, and since $m \leq k$, the caching algorithm requires a time polynomial in $ks \log |B_\tau|$ to handle the batch of requests B_τ. Thus, the amortized complexity of handling each request is polynomial in ks.

5.4 Analysis of the algorithm

Let $\mathcal{A}$ denote the algorithm presented in Section 5.3, and let $\mathcal{O}N$ denote any online algorithm for caching search engine result pages. The cache size in both algorithms is k, and the analysis assumes that both $\mathcal{A}$ and $\mathcal{O}N$ start with the same initial set of k cached pages, $\mathcal{C}_1$. The following is an adaption of the analysis of [11] to our needs.

Let $\mathcal{D}$ denote the set of all possible result pages. The analysis uses a *charging function* $c : \mathcal{D} \longrightarrow \mathcal{D} \cup \{\text{nil}\}$, that associates with each page d that $\mathcal{A}$ evicts, a page $c(d)$ that $\mathcal{O}N$ does not currently cache. The intuition behind these charges is that with high probability, $\mathcal{O}N$ will have to reload $c(d)$ no later than $\mathcal{A}$ reloads d. We will thus charge the cache miss of $\mathcal{A}$ on d on the cache miss of $\mathcal{O}N$ on $c(d)$. Initially, $c(d) = \text{nil}$ for all $d \in \mathcal{D}$.

Let $\{s_1, s_2, \ldots, s_m\} = B_\tau^U \setminus \mathcal{C}_\tau$ be the pages that $\mathcal{A}$ must bring into its cache following B_τ. Let $d_1, \ldots, d_m \in \mathcal{C}_\tau \setminus \mathcal{C}_{\tau+1}$ be the list of pages that $\mathcal{A}$ decides to evict from its cache, where d_i is the page evicted in the i'th iteration of step 2. Also, define $\mathcal{C}_\tau^0 = \mathcal{C}_\tau$, and for $0 < i \leq m$:

$$\mathcal{C}_\tau^i \stackrel{\triangle}{=} [\mathcal{C}_\tau \setminus \{d_1, \ldots, d_i\}] \cup \{s_1, \ldots, s_i\} = [\mathcal{C}_\tau^{i-1} \setminus \{d_i\}] \cup \{s_i\}.$$

Observe that $\mathcal{C}_\tau^m = \mathcal{C}_{\tau+1}$.

Denote by $\mathcal{O}N_\tau^+$ the set of pages in $\mathcal{O}N$'s cache after serving B_τ. Note that $\{s_1, \ldots, s_m\} \subseteq \mathcal{O}N_\tau^+ \setminus \mathcal{C}_\tau$. Also, $|\mathcal{O}N_\tau^+| = |\mathcal{C}_\tau^i| = k$. Thus,

$$|\mathcal{C}_\tau^i \setminus \mathcal{O}N_\tau^+| = |\mathcal{O}N_\tau^+ \setminus \mathcal{C}_\tau^i|. \tag{4}$$

The charging function maintains the following two invariants for all values of τ and i and for all pages d, d':

Invariant IV1 If $d \in \mathcal{C}_\tau^i$ then $c(d) = \text{nil}$.
Invariant IV2 If $c(d) = d'$ and $d' \in \mathcal{C}_\tau^i \setminus \mathcal{O}N_\tau^+$, then $d \in \mathcal{O}N_\tau^+ \setminus \mathcal{C}_\tau^i$.

Invariant IV2 and Equation 4 imply that whenever there is a page $d \in \mathcal{O}N_\tau^+ \setminus \mathcal{C}_\tau^i$ for which $c(d) = \text{nil}$, there is also a page $q \in \mathcal{C}_\tau^i \setminus \mathcal{O}N_\tau^+$ which carries no charge (i.e., there is no d' for which $c(d') = q$).

Both invariants trivially hold when $\tau = 0$. For each $\tau > 0$, the charging function is updated as follows. Initially, for every result page d,

- If $d \in B_\tau^U$ (d was requested at time τ) then $c(d) \leftarrow \text{nil}$.
- If $c(d) \neq \text{nil}$ and $d \notin \mathcal{O}N_\tau^+$, then $c(d) \leftarrow d$.

Note that the operations above preserve invariants IV1 and IV2.

Next, we update the charges of the m evicted pages. For all $i = 1, \ldots, m$, $d_i \in \mathcal{C}_\tau^{i-1}$, and so invariant IV1 implies that $c(d_i) = \text{nil}$. Now, if $d_i \notin \mathcal{O}N_\tau^+$ we set $c(d_i) \leftarrow d_i$. Otherwise (i.e., $d_i \in \mathcal{O}N_\tau^+ \setminus \mathcal{C}_\tau^i$), $c(d_i)$ is set to an arbitrary page $q \in C_\tau^i \setminus \mathcal{O}N_\tau^+$ which carries no charge (as noted above, Invariant IV2 and Equation 4 imply that such a page q must exist). We call this page as the *default charge* of d_i. It is easy to see that setting $c(d_i)$ in this manner preserves both invariants.

Lemma 1 *At any time τ and for any page q, there is at most a single page $d \neq q$ such that $c(d) = q$.*

Proof: An easy induction on τ, using the fact we set $c(d) \leftarrow q$ for a page $q \neq d$ only if q carries no charge. □

Corollary 4 *For any page q, at any point in time,*

$$|\{d \in \mathcal{D} \ : \ c(d) = q \ \}| \leq 2$$

and equality holds only if $q = c(q)$.

Lemma 2 *Let d_i be the i'th page that was evicted by $\mathcal{A}$ while serving B_τ, and let $q = c(d_i)$ at that time. With probability $\geq \frac{1}{2}$, the first post-τ request of q will occur no later than the first post-τ request of d_i.*

Proof: The Lemma trivially holds if $q = d_i$, so assume that this is not the case. Thus, q is the default charge of d_i. Define $V \stackrel{\triangle}{=} \mathcal{C}_\tau^{i-1} \setminus B_\tau^U$; by the method of assigning charges, both q and d_i are in V. Let $\mathcal{Q}$ denote the dominating distribution on the set V by which d_i was selected for eviction, and let $\bar{\mathcal{P}}_\tau(\cdot, \cdot)$ be the probabilities of the weighted tournament that gave rise to $\mathcal{Q}$. Then, by the way d_i is chosen, the first post-τ request of q will occur no later than that of d_i with probability $\sum_{d \in V} \mathcal{Q}(d)\mathcal{P}_\tau(q, d)$. By the properties of the dominating distribution,

$$\begin{aligned}
\frac{1}{2} &\geq \sum_{d \in V} \mathcal{Q}(d)\bar{\mathcal{P}}_\tau(d, q) \\
&= \sum_{d \in V \setminus \{q\}} \mathcal{Q}(d)[1 - \bar{\mathcal{P}}_\tau(q, d)] \\
&= 1 - \mathcal{Q}(q) - \sum_{d \in V \setminus \{q\}} \mathcal{Q}(d)\bar{\mathcal{P}}_\tau(q, d) \\
&\geq 1 - \mathcal{Q}(q) - \sum_{d \in V \setminus \{q\}} \mathcal{Q}(d)\mathcal{P}_\tau(q, d)
\end{aligned}$$

$$= 1 - \sum_{d \in V} \mathcal{Q}(d)\mathcal{P}_\tau(q,d)$$

and so $\sum_{d \in V} \mathcal{Q}(d)\mathcal{P}_\tau(q,d) \geq \frac{1}{2}$. □

Let $M(ALG)$ denote the number of cache misses of algorithm ALG in a given time frame $1, \ldots, \tau_f$ (including the service of B_{τ_f}).

Theorem 1 *$E[M(\mathcal{A})] \leq 4 \cdot E[M(\mathcal{O}N)]$, where the expectation is taken over both the random model of the requests and the random actions of $\mathcal{A}$.*

Proof: As in the proof of [11], we use the following indicator variables:

- $\chi(\tau, d)$ is 1 iff $\mathcal{A}$ evicts page d at time τ.
- $\gamma(\tau, d)$ is 1 iff $\mathcal{A}$ evicts page d at time τ, and $c(d)$ is requested no later than the first post-τ request of d (causing a cache miss of $\mathcal{O}N$).

Then we have:

$$M(\mathcal{A}) = \sum_{\tau=1}^{\tau_f} \sum_{d \in \mathcal{D}} \chi(\tau, d) \tag{5}$$

To estimate $M(\mathcal{O}N)$ we use two additional notations: $\mathcal{I}_\tau$ is the set of distinct pages requested in $B_1, \ldots, B_\tau$ that were not in $\mathcal{C}_1$, and $\mathcal{O}_\tau$ is the set of pages that $\mathcal{A}$ has evicted at least once by time τ and are not cached by $\mathcal{A}$ at time τ. Initially $\mathcal{I}_0 = \mathcal{O}_0 = \emptyset$. For $\tau > 0$, consider the changes in the contents of these sets when a page s is brought into $\mathcal{C}$, replacing an evicted page d: d is always added to $\mathcal{O}_\tau$. If s has not been previously cached by $\mathcal{A}$, it is added to $\mathcal{I}_\tau$; otherwise, it is removed from $\mathcal{O}_\tau$. In the former case the cardinalities of both sets increase by one, and in the latter case both cardinalities remain unchanged. We conclude that for all τ, $|\mathcal{I}_\tau| = |\mathcal{O}_\tau|$.

Now, let $\mathcal{I} \stackrel{\triangle}{=} \mathcal{I}_{\tau_f}$ and $\mathcal{O} \stackrel{\triangle}{=} \mathcal{O}_{\tau_f}$. When a page $q \in \mathcal{I}$ is requested for the first time, q causes a cache miss to $\mathcal{O}N$ (in fact, also to $\mathcal{A}$), but there is no page d for which $c(d) = q$ (since q has yet to be cached by $\mathcal{A}$). Therefore, $\mathcal{O}N$ has $|\mathcal{I}|$ cache misses which are unaccounted for by the γ-variables. To count the cache misses of $\mathcal{O}N$ which are accounted for by the γ-variables, observe that whenever $\gamma(\tau, d) = 1$, either (1) there is a post-τ request of d by time τ_f, and hence there is also a post-τ request of $c(d)$ (and a cache miss of $\mathcal{O}N$) by time τ_f, or (2) page d is not cached at time τ_f, and so $d \in \mathcal{O}$. Since for each page q there are at most two pages d for which $c(d) = q$, the number of cache misses of $\mathcal{O}N$ which are accounted for by the γ-variables is at least $\frac{1}{2}\left(\sum_{\tau=1}^{\tau_f} \sum_{d \in \mathcal{D}} \gamma(\tau, d) - |\mathcal{O}|\right)$. Summing $\mathcal{O}N$'s cache misses of both types we

have

$$M(\mathcal{O}N) \geq |\mathcal{I}| + \frac{1}{2}\left(\sum_{\tau=1}^{\tau_f}\sum_{d\in\mathcal{D}}\gamma(\tau,d) \ - |\mathcal{O}|\right) \geq \frac{1}{2}\sum_{\tau=1}^{\tau_f}\sum_{d\in\mathcal{D}}\gamma(\tau,d) \tag{6}$$

(The second inequality follows by the observation above that $|\mathcal{I}| = |\mathcal{O}|$.)

By lemma 2, for every page d and time τ,

$$E[\gamma(\tau,d) \mid \chi(\tau,d)=1] \geq \frac{1}{2} \Longrightarrow E[\chi(\tau,d)] \leq 2E[\gamma(\tau,d)] \tag{7}$$

Combining equations 5, 6 and 7, we have

$$\begin{aligned} E[M(\mathcal{A})] &= E[\sum_{\tau=1}^{\tau_f}\sum_{d\in\mathcal{D}}\chi(\tau,d)] = \sum_{\tau=1}^{\tau_f}\sum_{d\in\mathcal{D}}E[\chi(\tau,d)] \\ &\leq \sum_{\tau=1}^{\tau_f}\sum_{d\in\mathcal{D}}2E[\gamma(\tau,d)] = 2E[\sum_{\tau=1}^{\tau_f}\sum_{d\in\mathcal{D}}\gamma(\tau,d)] \leq 4E[M(\mathcal{O}N)] \ , \end{aligned}$$

which concludes the proof. □

6 Conclusions and Future Research

This paper considered the online problem of caching search result pages in Web search engines. We discussed several issues which should be addressed when transforming the real-life problem into an abstract model. We then presented a specific discrete-time model of the manner in which queries are submitted to search engines by multiple client sessions, and showed an adaptation of a known probabilistic online paging algorithm to this model. The expected cost of paging that this algorithm incurs is no worst than 4 times the expected cost of any online scheme.

Possible variations of our model The model and algorithm of Section 5 have assumed that the probability p of users requesting page $(t, m+1)$ after viewing page (t, m) is independent of both t and m. However, replacing the single probability p by $s-1$ probabilities $p_1, \ldots, p_{s-1}$ (after viewing (t, m), $(t, m+1)$ is requested with probability p_m) will require only minor adjustments in the algorithm, and will not change its complexity. Furthermore, each topic t may have its own set of probabilities associated with it, provided that the

algorithm is familiar with all these probabilities (as it is familiar with λ_t, the arrival rate of new search sessions on topic t).

Another possible change in the model is to lift the limit of s result pages per topic, thereby considering each topic to have an infinite list of result pages. The algorithm itself need not undergo any changes; only the analysis of its complexity is affected, as the complexity of calculating $\mathcal{P}(a, b)$ depends on the depth of a and b in their corresponding topics' lists. However, since the probability of browsing increasingly deep result pages declines exponentially in our model, the expected complexity of such calculations remains low.

Future research One of the drawbacks of our current model is that it requires all active sessions to generate queries at successive discrete time steps. This implies that all users digest search results and conduct their sessions at the same pace, which is not a faithful representation of reality. It would be interesting to extend the theoretical analysis to more realistic models of query streams. This may involve the adoption of continuous time models, perhaps in the spirit of [23], or further work on discrete models where users will be allowed to skip several batches between successive submissions of a search session.

Regarding the algorithm itself, it currently requires solving a linear program that is quadratic in the size of the cache upon every cache replacement. This is clearly impractical in the domain of Web search engines, which serve millions of queries per day, and which use caches whose capacity exceeds hundreds of thousands of queries. We identify two separate research directions that may address this problem:

- The algorithm, as described in Section 5.3, requires the computation of a sequence of m closely-related dominating distributions (where m is the number of cache misses in a batch). It would be helpful to (1) calculate (or approximate) a dominating distribution without solving the quadratic LP, or (2) find a way to quickly derive dominating distribution i based on dominating distribution $i - 1$ (for $i = 2, \ldots, m$).
- In [4], a probability-driven cache replacement algorithm called *PDC* was proposed, requiring time that is logarithmic in the size of the cache per operation. PDC's replacement procedure, which approximates the probability that every cached page will be requested in the near future by at least one query, shares much of the same intuition behind the probabilistic model presented in Section 5.1. Through trace-driven simulations it was shown that PDC outperformed LRU-based schemes, and achieved hit rates close to those that were theoretically possible for the traces used. It would be interesting to pursue other caching schemes that bridge the gap between theoretically guaranteed cache performance on one hand, and practical im-

plementations on the other.

Another issue for future research is the integration of search result prefetching into the competitive analysis framework. In an experimental study of search result caching [4], it was shown that search engines can significantly improve the hit ratios of their result page caches by prefetching search results. The idea behind such prefetching is to anticipate follow-up queries by users, and to cache result pages $(t, m), \ldots, (t, m + r)$ for some $r \geq 1$ for every query (t, m) that causes a cache miss. While the classic paging model would count such an action as $r + 1$ replacements, the actual cost in many search engine architectures of preparing bulks of result pages on the same topic is *sublinear* in the size of the bulk [5]. Accordingly, competitive analysis of algorithms that are charged sublinearly for bulk prefetch operations may analytically show the merits of prefetching.

Acknowledgments

We thank Adi Rosén for valuable discussions on the problems covered in this paper.

References

[1] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, in: Proc. 7th International WWW Conference, 1998, pp. 107–117.

[2] E. P. Markatos, On caching search engine query results, in: Proceedings of the 5th International Web Caching and Content Delivery Workshop, 2000.

[3] P. Saraiva, E. Moura, N. Ziviani, W. Meira, R. Fonseca, B. Ribeiro-Neto, Rank-preserving two-level caching for scalable search engines, in: Proc. 24rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New Orleans, Louisiana, USA, 2001, pp. 51–58.

[4] R. Lempel, S. Moran, Predictive caching and prefetching of query results in search engines, in: Proc. 12th World Wide Web Conference (WWW2003), Budapest, Hungary, 2003.

[5] R. Lempel, S. Moran, Optimizing result prefetching in web search engines with segmented indices, in: Proc. 28th International Conference on Very Large Data Bases, Hong Kong, China, 2002.

[6] L. A. Belady, A study of replacement algorithms for a virtual-storage computer, IBM Systems Journal 5 (2) (1966) 78–101.

[7] D. D. Sleator, R. E. Tarjan, Amortized efficiency of list update and paging rules, Communications of the ACM 28 (1985) 202–208.

[8] A. Fiat, A. R. Karlin, Randomized and multipointer paging with locality of reference, in: Proc. 27th Annual ACM Symposium on Theory of Computing, 1995, pp. 626–634.

[9] A. Borodin, R. El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press, 1998.

[10] S. Irani, Competitive analysis of paging, in: A. Fiat, G. J. Woeginger (Eds.), Online Algorithms, The State of the Art, Lecture Notes in Computer Science, Vol. 1442, Springer, 1998, pp. 52–73.

[11] C. Lund, S. Phillips, N. Reingold, Paging against a distribution and IP networking, Journal of Computer and System Sciences 58 (1) (1999) 222–232. URL `citeseer.nj.nec.com/94934.html`

[12] B. J. Jansen, A. Spink, T. Saracevic, Real life, real users, and real needs: A study and analysis of user queries on the web, Information Processing and Management 36 (2) (2000) 207–227.

[13] C. Silverstein, M. Henzinger, H. Marais, M. Moricz, Analysis of a very large altavista query log, Tech. Rep. 1998-014, Compaq Systems Research Center (October 1998).

[14] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, N. E. Young, Competitive paging algorithms, Journal of Algorithms 12 (4) (1991) 685–699.

[15] L. A. McGeoch, D. D. Sleator, A strongly competitive randomized paging algorithm, Algorithmica 6 (1991) 816–825.

[16] A. Borodin, S. Irani, P. Raghavan, B. Schieber, Competitive paging with locality of reference, in: Proc. 23rd ACM Symposium on Theory of Computing (STOC 1991), 1991, pp. 249–259.

[17] A. R. Karlin, S. J. Phillips, P. Raghavan, Markov paging, SIAM Journal on Computing 30 (3) (2000) 906–922.

[18] H. Alborzi, E. Torng, P. Uthaisombut, S. Wagner, The k-client problem, Journal of Algorithms 41 (2) (2001) 115–173.

[19] D. Breslauer, On competitive on-line paging with lookahead, Theoretical Computer Science 209:2 (1998) 365–375.

[20] S. Albers, On the influence of lookahead in competitive paging algorithms, Algorithmica 18 (1997) 283–305.

[21] T. Feder, R. Motwani, R. Panigrahy, A. Zhu, Web caching with request reordering, in: Proc. 13th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2002.

[22] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, L. Ozsen, Optimal crawling strategies for web search engines, in: Proc. 11th International World Wide Web Conference (WWW2002), 2002.

[23] A. Kraiss, G. Weikum, Integrated document caching and prefetching in storage hierarchies based on markov-chain predictions, VLDB 7 (3) (1998) 141–162.

[24] R. Goodman, Introduction to Stochastic Models, Benjamin/Cummings, 1988.