Title: Online paging and caching
Name: Neal E. Young[1]
Affil./Addr. University of California, Riverside
Keywords: paging; caching; least recently used; k-server problem; online algorithms; competitive analysis; competitive ratio
SumOriWork: 1985–2013; multiple authors

Online paging and caching

Neal E. Young[1]
University of California, Riverside

Years aud Authors of Summarized Original Work

1985–2013; multiple authors

Keywords

paging; caching; least recently used; k-server problem; online algorithms; competitive analysis; competitive ratio

SYNONYMS: paging, caching, weighted caching, weighted paging, file caching

Problem Definition

A *file-caching* problem instance specifies a cache size k (a positive integer) and a sequence of requests to files, each with a *size* (a positive integer) and a *retrieval cost* (a non-negative number). The goal is to maintain the cache to satisfy the requests while minimizing the retrieval cost. Specifically, for each request, if the file is not in the cache, one must retrieve it into the cache (paying the retrieval cost) and remove other files to bring the total size of files in the cache to k or less. *Weighted caching*, or *weighted paging* is the special case when each file size is 1. *Paging* is the special case when each file size and each retrieval cost is 1 (then the retrieval cost is the number of *cache misses*, and the *fault rate* is the average retrieval cost per request).

An algorithm is *online* if its response to each request is independent of later requests. In practice this is generally necessary. Standard worst-case analysis is not meaningful for online algorithms — any algorithm will have some input sequence that forces a retrieval for every request. Yet worst-case analysis can be done meaningfully as follows. An algorithm is $c(h,k)$*-competitive* if on *any* sequence σ the total (expected) retrieval cost incurred by the algorithm using a cache of size k is at most $c(h,k)$ times the *minimum* cost to handle σ with a cache of size h (plus a constant independent of σ). Then the algorithm has *competitive ratio* $c(h,k)$. The study of competitive ratios is called *competitive analysis.* (In the larger context of approximation algorithms for combinatorial optimization, this ratio is commonly called the *approximation ratio.*)

Algorithms. Here are definitions of a number of caching algorithms; first is LANDLORD. LANDLORD gives each file "credit" (equal to its cost) when the file is requested and not in cache. When necessary, LANDLORD reduces all cached file's credits proportionally to file size, then evicts files as they run out of credit.

```
File-caching algorithm LANDLORD
Maintain real value credit[$f$] with each file $f$ (credit[$f$] = 0 if $f$ is not in the cache).
When a file $g$ is requested:
1. if $g$ is not in the cache:
2.     until the cache has room for $g$:
3.         for each cached file $f$: decrease credit[$f$] by $\Delta \cdot$ size[$f$],
4.             where $\Delta = \min_{f \in \text{cache}} \text{credit}[f]/\text{size}[f]$.
5.         Evict from the cache any subset of the zero-credit files $f$.
6.     Retrieve $g$ into the cache; set credit[$g$] $\leftarrow$ cost($g$).
7. else Reset credit[$g$] anywhere between its current value and cost($g$).
```

For weighted caching, file sizes equal 1. GREEDY DUAL is LANDLORD for this special case. BALANCE is the further special case obtained by leaving credit unchanged in line 7.

For paging, files sizes and costs equal 1. FLUSH-WHEN-FULL is obtained by evicting *all* zero-credit files in line 5; FIRST-IN-FIRST-OUT is obtained by leaving credits unchanged in line 7 and evicting the file that entered the cache earliest in line 5; LEAST-RECENTLY-USED is obtained by raising credits to 1 in line 7 and evicting the least-recently requested file in line 5. The MARKING algorithm is obtained by raising credits to 1 in line 7 and evicting a *random* zero-credit file in line 5. (LANDLORD generalizes to arbitrary covering problems with submodular costs as described in [10].)

Key Results

This entry focuses on competitive analysis of paging and caching strategies as defined above. Competitive analysis has been applied to many problems other than paging and caching, and much is known about other methods of analysis (mainly empirical or average-case) of paging and caching strategies, but these are outside scope of this entry.

Paging. In a seminal paper, Sleator and Tarjan showed that LEAST-RECENTLY-USED, FIRST-IN-FIRST-OUT, and FLUSH-WHEN-FULL are $\frac{k}{k-h+1}$-competitive [13]. Sleator and Tarjan also showed that this competitive ratio is the best possible for any deterministic online algorithm. Fiat *et al.* showed that the MARKING algorithm is $2H_k$-competitive and that no randomized online algorithm is better than H_k-competitive [6]. Here $H_k = 1 + 1/2 + \cdots + 1/k \approx .58 + \ln k$. McGeoch and Sleator gave an optimal H_k-competitive randomized online paging algorithm [12].

Weighted caching. For weighted caching, Chrobak *et al.* showed that the deterministic online BALANCE algorithm is k-competitive [4]. Young showed that GREEDY DUAL is $\frac{k}{k-h+1}$-competitive, and that GREEDY DUAL is a primal-dual algorithm — it generates a solution to the linear-programming dual which proves the near-optimality of the primal solution [14]. Bansal et al., resolving a long-standing open problem, used the primal-dual framework to give an $O(\log k)$competitive randomized algorithm for weighted caching [2].

File caching. When each cost equals 1 (the goal is to minimize the *number* of retrievals), or when each file's cost equals the file's size (the goal is to minimize the total number of *bytes* retrieved), Irani gave $O(\log^2 k)$-competitive randomized online algorithms [7].

For general file caching, Irani and Cao showed that a restriction of LANDLORD is k-competitive [3]. Independently, Young showed that LANDLORD is $\frac{k}{k-h+1}$-competitive [15].

Other theoretical models. Practical performance can be better than the worst case studied in competitive analysis. Refinements of the model have been proposed to increase realism. Borodin *et al.* [1], to model locality of reference, proposed the *access-graph* model (see also [8; 9]). Koutsoupias and Papadimitriou proposed the *comparative ratio* (for comparing classes of online algorithms directly) and the *diffuse-adversary model* (where the adversary chooses requests probabilistically subject to restrictions) [11]. Young showed that any $\frac{k}{k-h+1}$-competitive algorithm is also *loosely* $O(1)$-competitive: for any fixed $\varepsilon, \delta > 0$, on any sequence, for all but a δ-fraction of cache sizes k, the algorithm either is $O(1)$-competitive or pays at most ε times the sum of the retrieval costs [15].

Analyses of deterministic algorithms. Here is a competitive analysis of GREEDY DUAL for weighted caching.

Theorem 1. GREEDY DUAL *is* $\frac{k}{k-h+1}$*-competitive for weighted caching.*

Proof. Here is an amortized analysis (in the spirit of Sleator and Tarjan, Chrobak *et al.*, and Young; see [14] for a different primal-dual analysis). Define potential

$$\Phi = (h-1)\cdot \sum_{f\in \mathrm{GD}} \mathrm{credit}[f] \;+\; k\cdot \sum_{f\in \mathrm{OPT}} \Big(\mathrm{cost}(f) - \mathrm{credit}[f]\Big),$$

where GD and OPT denote the current caches of GREEDY DUAL and OPT(the optimal off-line algorithm that manages the cache to minimize the total retrieval cost), respectively. After each request, GREEDY DUAL and OPT take (some subset of) the following steps in order.

OPT evicts a file f: Since $\mathrm{credit}[f] \le \mathrm{cost}(f)$, Φ cannot increase.

OPT retrieves requested file g: OPT pays $\mathrm{cost}(g)$; Φ increases by at most $k\,\mathrm{cost}(g)$.

GREEDY DUAL decreases $\mathrm{credit}[f]$ for all $f \in$ GD: The cache is full and the requested file is in OPT but not yet in GD. So $|\mathrm{GD}| = k$ and $|\mathrm{OPT}\cap\mathrm{GD}| \le h-1$. Thus, the total decrease in Φ is $\Delta[(h-1)|\mathrm{GD}| - k\,|\mathrm{OPT}\cap\mathrm{GD}|] \ge \Delta[(h-1)k - k(h-1)] = 0$.

GREEDY DUAL evicts a file f: Since $\mathrm{credit}[f] = 0$, Φ is unchanged.

GREEDY DUAL retrieves requested file g and sets $\mathrm{credit}[g]$ to $\mathrm{cost}(g)$: GREEDY DUAL pays $c = \mathrm{cost}(g)$. Since g was not in GD but is in OPT, $\mathrm{credit}[g] = 0$ and Φ decreases by $-(h-1)c + k\,c = (k-h+1)c$.

GREEDY DUAL resets $\mathrm{credit}[g]$ between its current value and $\mathrm{cost}(g)$: Since $g \in$ OPT and $\mathrm{credit}[g]$ only increases, Φ decreases.

So, with each request: (1) when OPT retrieves a file of cost c, Φ increases by at most kc; (2) at no other time does Φ increase; and (3) when GREEDY DUAL retrieves a file of cost c, Φ decreases by at least $(k-h+1)c$. Since initially $\Phi = 0$ and finally $\Phi \ge 0$, it follows that GREEDY DUAL's total cost times $k-h+1$ is at most OPT's cost times k.

Extension to file caching. Although the proof above easily extends to LANDLORD, it is more informative to analyze LANDLORD via a *general reduction* from file caching to weighted caching:

Corollary 1. LANDLORD *is* $\frac{k}{k-h+1}$*-competitive for file caching.*

Proof. Let W be any deterministic c-competitive weighted-caching algorithm. Define file-caching algorithm F_W as follows. Given request sequence σ, F_W simulates W on weighted-caching sequence σ' as follows. For each file f, break f into size(f) "pieces" $\{f_i\}$ each of size 1 and cost cost(f)/size(f). When f is requested, give a batch $(f_1, f_2, \ldots, f_s)^{N+1}$ of requests for pieces to W. Take N large enough so W has all pieces $\{f_i\}$ cached after the first sN requests of the batch.

Assume that W *respects equivalence*: after each batch, for every file f, all or none of f's pieces are in W's cache. After each batch, make F_W update its cache correspondingly to $\{f : f_i \in \text{cache}(W)\}$. F_W's retrieval cost for σ is at most W's retrieval cost for σ', which is at most $c\,\text{OPT}(\sigma')$, which is at most $c\,\text{OPT}(\sigma)$. Thus, F_W is c-competitive for file caching.

Now, observe that GREEDY DUAL can be made to respect equivalence. When GREEDY DUAL processes a batch of requests $(f_1, f_2, \ldots, f_s)^{N+1}$ resulting in retrievals, for the last s requests, make GREEDY DUAL set credit$[f_i]$ = cost(f_i) = cost$(f)/s$ in line 7. In general, restrict GREEDY DUAL to raise credits of equivalent pieces f_i equally in line 7. After each batch the credits on equivalent pieces f_i will be the same. When GREEDY DUAL evicts a piece f_i, make GREEDY DUAL evict all other equivalent pieces f_j (all will have zero credit).

With these restrictions, GREEDY DUAL respects equivalence. Finally, taking W to be GREEDY DUAL above, F_W is LANDLORD.

Analysis of the randomized* MARKING *algorithm. Here is a competitive analysis of the MARKING algorithm.

Theorem 2. *The* MARKING *algorithm is* $2H_k$*-competitive for paging.*

Proof. Given a paging request sequence σ, partition σ into contiguous *phases* as follows. Each phase starts with the request after the end of the previous phase and continues as long as possible subject to the constraint that it should contain requests to at most k distinct pages. (Each phase starts when the algorithm runs out of zero-credit files and reduces all credits to zero.)

Say a request in the phase is *new* if the item requested was not requested in the previous phase. Let m_i denote the number of new requests in the ith phase. During phases $i-1$ and i, $k+m_i$ distinct files are requested. OPT has at most k of these in cache at the start of the $i-1$st phase, so it will retrieve at least m_i of them before the end of the ith phase. So OPT's total cost is at least $\max\{\sum_i m_{2i}, \sum_i m_{2i+1}\} \geq \sum_i m_i/2$.

Say a non-new request is *redundant* if it is to a file with credit 1 and non-redundant otherwise. Each new request costs the MARKING algorithm 1. The jth non-redundant request costs the MARKING algorithm at most $m_i/(k-j+1)$ in expectation because, of the $k-j+1$ files that if requested would be non-redundant, at most m_i are not in the cache (and each is equally likely to be in the cache). Thus, in expectation MARKING pays at most $m_i + \sum_{j=1}^{k-m_i} m_i/(k-j+1) \leq m_i H_k$ for the phase, and at most $H_k \sum_i m_i$ total.

Applications

Variants of GREEDY DUAL and LANDLORD have been incorporated into file-caching software such as Squid [5].

Open Problems

None to report.

Experimental Results

For a study of competitive ratios on practical inputs, see for example [14; 3; 5].

Cross-References

Algorithm DC-Tree for k-Servers on Tree
Online List Update
Performance Measures in Online Algorithms
Price of Anarchy
Work-Function Algorithm for K-servers

Recommended Reading

1. Borodin A, Irani S, Raghavan P, Schieber B (1991) Competitive paging with locality of reference. In: Proc. 23rd Symp. Theory of Computing (STOC), ACM, pp 249–259
2. Buchbinder N, Naor J (2005) Online primal-dual algorithms for covering and packing problems. Lecture Notes in Computer Science 3669:689–701
3. Cao P, Irani S (1997) Cost-aware WWW proxy caching algorithms. In: USENIX Symposium on Internet Technologies and Systems, USENIX Association Berkeley, CA, USA
4. Chrobak M, Karloff H, Payne T, Vishwanathan S (1991) New results on server problems. SIAM J Discrete Math 4(2):172–181
5. Dilley J, Arlitt M, Perret S (1999) Enhancement and validation of Squid's cache replacement policy. Tech. Rep. HPL-1999-69, Hewlett-Packard Laboratories, also in 4th International Web Caching Workshop
6. Fiat A, Karp RM, Luby M, McGeoch LA, Sleator DD, Young NE (1991) Competitive paging algorithms. J Algorithms 12:685–699
7. Irani S (2002) Page replacement with multi-size pages and applications to web caching. Algorithmica 33(3):384–409
8. Irani S, Karlin A, Phillips S (1992) Strongly competitive algorithms for paging with locality of reference. In: Proc. 3rd Symp. on Discrete Algorithms (SODA), ACM/SIAM, pp 228–236
9. Karlin AR, Phillips SJ, Raghavan P (2000) Markov paging. SIAM Journal on Computing 30(3):906–922
10. Koufogiannakis C, Young NE (2013) Greedy Δ-approximation algorithm for covering with arbitrary constraints and submodular cost. Algorithmica 66(1):113–152
11. Koutsoupias E, Papadimitriou C (2000) Beyond competitive analysis. SIAM J Comput 30(1):300–317
12. McGeoch L, Sleator D (1991) A strongly competitive randomized paging algorithm. Algorithmica 6(6):816–825
13. Sleator D, Tarjan RE (1985) Amortized efficiency of list update and paging rules. Commun ACM 28:202–208
14. Young NE (1994) The k-server dual and loose competitiveness for paging. Algorithmica 11:525–541, preliminary version appeared in SODA'91 titled "On-Line Caching as Cache Size Varies"
15. Young NE (2002) On-line file caching. Algorithmica 33(3):371–383