Caching Documents with Variable Sizes and Fetching Costs: An LP-Based Approach*

Edith Cohen
AT&T Labs–Research
180 Park Avenue
Florham Park, NJ 07932 USA
edith@research.att.com

Haim Kaplan
Tel-Aviv University
Tel-Aviv 69978
Israel
haimk@math.tau.ac.il

September 14, 2001

Abstract

We give an integer programming formulation of the paging problem with varying sizes and fetching costs. We use this formulation to provide an alternative proof that a variant of the algorithm greedy-dual-size previously considered in [4, 15] is $\frac{k+1}{k-h+1}$-competitive against the optimal strategy with cache size $h \leq k$. Our proof provides further insights to greedy-dual-size. We also indicate how the same integer programming formulation has been recently used [11, 1] to obtain approximation algorithms to the NP-complete "offline" problem.

1 Introduction

We consider the paging problem where pages have varying sizes and fetching costs. This problem applies to caching at Web servers, proxies, and browsers. The goal is to maintain a cache of total size at most k bytes so as to minimize the total fetching costs over cache misses. We refer to this problem as the *generalized caching problem*. Following Sleator and Tarjan [12] we define an online algorithm to be ρ *competitive* if for any request sequence its cost is at most ρ times the cost of the optimal offline strategy.

Cao and Irani [4] suggested a simple algorithm for the generalized caching problem which they named *greedy-dual-size*. This algorithm is a generalization of an algorithm of Young [14] called *greedy-dual* that handles variable fetching costs but uniform sizes. Both greedy-dual and greedy-dual-size generalize LRU, FIFO, and other well studied caching algorithms when sizes and fetching costs are uniform. Young proved [14] that *greedy-dual* is $k/(k-h+1)$ competitive against an optimal offline strategy that uses a cache of size $h \leq k$. Cao and Irani proved that *greedy-dual-size* is k competitive against an optimal strategy that also uses a cache of size k. Independently, Young

*An abstract of this paper appeared in Proc. 10th ACM-SIAM Symposium on Discrete Algorithms, 1999 [5].

[15] studied a slightly different variant of greedy-dual-size that he called the *landlord* algorithm. Young proved [15] that landlord is $k/(k-h+1)$ competitive against an optimal offline strategy that uses a cache of size h for every $h \leq k$.

Young's proof that greedy-dual is $k/(k-h+1)$ competitive uses a primal-dual linear programming approach [14]. The linear programming formulation used by Young is not easily extendible to handle varying sizes, and both Cao and Irani [4] and Young [15] prove the performance guarantee of greedy-dual-size using direct counting arguments.

In this paper we give an integer programming formulation of the caching problem with variable sizes and fetching costs. This formulation is different from the one used by Young in [14]. Using our LP formulation, we give an alternative proof to the performance guarantee of greedy-dual-size that provides additional insights into the algorithm.

The structure of this paper is as follows. Section 2 gives basic definitions and presents the variant of greedy-dual-size which we analyze. Section 3 presents an integer programming formulation of the generalized caching problem and use it to prove that our variant of greedy-dual-size is $(k+1)/k-h+1$ competitive against an optimal offline strategy with cache of size $h \leq k$. Section 4 indicates how our linear programming formulation can also be used to construct approximation algorithms for the offline generalized caching problem, where the request sequence is known in advance.

2 Preliminaries

We denote the size of the cache in bytes by k, and the size of page p in bytes by $s(p)$. We also let $f(p)$ be the cost of fetching p at the time of its next request. (In other words, $f(p)$ is the cost of a cache miss resulting from evicting p.) Note that $f(p)$ is a function of time and may change as a result of a request for page p. In particular, if p is never requested again, we assume that $f(p) = 0$. At any time the cache contains a set of pages of total size no greater than k. When a page is requested, if it is cached then the request is a *hit*. Otherwise it is a *miss* and the caching algorithm may evict a subset of pages in order to make room for caching the currently requested page. Similar to Irani [7], we charge the algorithm by the respective fetching cost $f(p)$ upon each decision to evict or not to cache a page p. This cost measure differs by an additive constant[1] (incurred by all algorithms) from charging for the fetching cost upon each miss. The goal of a caching algorithm is to minimize the total fetching cost over cache misses to pages that were previously requested and evicted from the cache.

We consider the following variant of greedy-dual-size. The algorithm maintains a nonnegative number, credit(p), with each page p in the cache. Given a request for p, if p is in the cache the algorithm resets credit(p) to $f(p)$. Otherwise, the algorithm sets credit$(p) = f(p)$ and "pretends" p is in the cache. Then it repeats the following eviction step until the total size of the items in the

[1] which equals to the sum of the costs of cache misses incurred on the first request of each page.

cache is no greater than k.

Eviction step: Let $p_{\min}$ be an item in the cache which minimizes the ratio $\text{credit}(p)/s(p)$ and let $\Delta = \text{credit}(p_{min})/s(p_{\min})$. For every p in the cache the algorithm decreases $\text{credit}(p)$ by $s(p)\Delta$, and then evicts $p_{\min}$.

When a request arrives for a page p which is not in the cache, our variant of greedy-dual-size does not necessarily force p into the cache. Forcing a page into the cache is a realistic model of operating system caches but the actual scenario in most web-caches is that the page does not have to enter the cache in order to serve the request. This assumption is consistent with that of Irani [7] and is the only detail in which our variant differs from the one considered by Young [15]. We prove the following:

Theorem 1 *The cost of greedy-dual-size is at most $\frac{k+1}{k-h+1}$ times the cost of the optimal strategy that uses a cache of size h for every $h \leq k$.*

We prove this theorem in the next section. Our algorithm is $\frac{k+1}{k-h+1}$ competitive rather than $\frac{k}{k-h+1}$ competitive because it does not force the requested page into the cache. Under this assumption LRU is also $\frac{k+1}{k-h+1}$ competitive and slight modification in the proof of Sleator and Tarjan [12] shows that this is a lower bound on the performance of any deterministic strategy.

As other variants of greedy-dual-size one can implement the variant we consider here such that the time per eviction is at most logarithmic in the number of pages in the cache that is, $O(\log k)$ per eviction. We achieve that by maintaining a running score L that we increase when a page is evicted rather than decreasing credits of every cached page. When a page p enters the cache we give it a score of $L + f(p)/s(p)$. We pick a page to evict whose score L' is minimum among the scores of cached pages and after evicting it we increase L to be L'. It is not hard to see that this is indeed an implementation of greedy-dual-size as we described it before.

3 Performance of greedy-dual-size

In this section we formulate the generalized caching problem as an integer program and use a linear programming relaxation of this formulation to prove Theorem 1.

We index the requests $1, 2, \ldots, n$ and denote by p_j the page requested by the jth request. For a request j we denote by j' the index of the previous request to the same page. More precisely j' is the maximum index l such that $l < j$ and $p_l = p_j$. If j is the first request to a page then we define j' to be zero. We also denote by $s_j = s(p_j)$ the size of the page requested by the jth request and denote by $f_j = f(p_j)$ the fetching cost of this page at the time of its next request.

We formulate our caching problem as an integer program, IP_k, as follows. For every request j such that $j' \neq 0$ we introduce a variable x_j taking either the value 0 or the value 1. Variable x_j takes the value 1 if and only if the page of the jth request is not in the cache when the request is issued. In other words x_j takes the value 1 if and only if the jth request is a miss. For every request i, $1 \leq i \leq n$ we have a *packing constraint* in IP_k which requires that the sum of the sizes of pages that are cached right after request i is issued does not exceed k.

$$\text{minimize} \sum_{j | j' \neq 0} x_j f_j \ ,$$

subject to $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathrm{IP}_\mathbf{k}$

$$\begin{aligned} \sum_{j | 0 < j' \leq i < j} x_j s_j \quad &\geq \quad \left(\sum_{j | 0 < j' \leq i < j} s_j \right) - k, \quad \forall i = 1, \ldots, n, \\ x_j \quad &\in \quad \{0, 1\} \quad \forall j = 1, \ldots, n \text{ such that } j' \neq 0. \end{aligned}$$

We get a linear program LP_k by relaxing the constraints $x_j \in \{0,1\}$ to $x_j \in [0,1]$.

The dual linear program DP_k uses a variable $r_j \geq 0$ for each constraint $x_j \leq 1$, and a variable $q_i \geq 0$ associated with each packing constraint. The program DP_k is defined as follows.

The objective is to

$$\text{maximize} \sum_{i=1}^{n} \left(\sum_{j | 0 < j' \leq i < j} s_j - k \right) q_i - \sum_{j | j' \neq 0} r_j$$

subject to $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathrm{DP}_\mathbf{k}$

$$s_j \left(\sum_{i | 0 < j' \leq i < j} q_i \right) - r_j \leq f_j \quad \forall j \text{ such that } j' \neq 0.$$

We shall show that after each step of the algorithm, there is a feasible dual solution (r, q), where $r = (r_{j | j' \neq 0})$ and $q = (q_1, \ldots, q_n)$ such that the cost[2] of greedy-dual-size is at most

$$\mathrm{cost}(\mathrm{GDS}) \leq \Phi(r, q) \equiv \frac{k+1}{k-h+1} \mathrm{DP}_h(r, q) - \sum_{p \in C} (f(p) - \mathrm{credit}(p)) \ , \tag{1}$$

where $\mathrm{DP}_h(r, q)$ is the value of the dual objective at (r, q), and C is the set of currently cached pages.

By linear programming duality and the fact that LP_h is a relaxation of IP_h we have that,

$$\mathrm{DP}_h(r, q) \leq \mathrm{OPT}(\mathrm{DP}_h) = \mathrm{OPT}(\mathrm{LP}_h) \leq \mathrm{OPT}(\mathrm{IP}_h) \ . \tag{2}$$

Combining Invariant (1) with (2) and the fact that at the end of the sequence, $f(p) = \mathrm{credit}(p) = 0$ for all pages we obtain that the total cost of greedy-dual-size is at most

$$\frac{k+1}{k-h+1} \mathrm{OPT}(\mathrm{IP}_h)$$

[2] Sum of fetching costs of items evicted so far.

and Theorem 1 follows.

We prove Invariant (1) by induction on the steps of the algorithm. Furthermore we shall prove that the feasible dual solution (r, q) which certifies Invariant (1) after request i also satisfies Invariants (3) and (4) defined as follows. Invariant (3) requires that

$$s_j \left(\sum_{0<j' \leq i < j} q_i \right) - r_j + \text{credit}(p_j) \leq f_j \ , \tag{3}$$

for every j such that $0 < j' \leq i < j$, and p_j is cached by greedy-dual-size. Invariant (4) requires that

$$q_\ell = 0 \ \text{ for every } \ \ell > i \ \text{ and } \ r_j = 0 \ \text{ if } \ j' > i. \tag{4}$$

Invariants (1), (3), and (4) are true before serving the first request by taking $r = 0$ and $q = 0$. Now assume that the invariant is true after serving request $i - 1$, for some $1 \leq i \leq n$, and we would like to prove that it is true also after serving request i. If request i is a hit then the cost of greedy-dual-size does not increase. The potential $\Phi(r, q)$ cannot decrease and may even increase as a result of resetting credit(p_i) to $f(p_i)$. Therefore Invariant (1) continues to hold with the same dual solution (r, q). It is easy to see that Invariant (4) also holds. To see that (3) holds note that the set of constraints included in Invariant (3) contains one constraint that was not included by this invariant before serving request i. This is the constraint associated with the first request m to p_i following request i. Since $q_l = 0$ for $l \geq i$ and $r_m = 0$ the new constraint holds. The other constraints in Invariant (3) clearly continue to hold.

Assume request i is a miss. The first stage in serving the ith request is "pretending" that p_i is in the cache. This adds one additional constraint to Invariant (3) that corresponds to the next request m for p_i. This constraint is satisfied since $q_l = 0$ for $l \geq i$ and $r_m = 0$. Thus the same dual solution still satisfy the invariants after this first stage of serving a miss.

Let $p_1, \ldots, p_f$ be the pages which were evicted while processing the ith request. For each p_ℓ we show that Invariants (1), (3), and (4) continue to hold after the eviction of p_ℓ. Let Δ_ℓ be the value of Δ when p_ℓ is evicted. Let (r^1, q^1) be the dual solution which certifies the invariant before evicting p_ℓ. The dual solution (r^2, q^2) which certifies the invariant after evicting p_ℓ is obtained from (r^1, q^1) by increasing q_i^1 by Δ_ℓ and increasing r_j^1 by $s_j \Delta_\ell$ for every j such that $j' \leq i < j$ and the item requested by the jth request is not in the cache at the time of processing request i. The new solution (r^2, q^2) is feasible since (r^1, q^1) was feasible and satisfied Invariant (3). Furthermore, since greedy-dual-size decreases the credits of each cached page by $s_j \Delta_\ell$ the new solution (r^2, q^2) also satisfies Invariant (3).

We show that Invariant (1) continues to hold in two stages. First we show that it continues to hold after greedy-dual-size decreases credits but before it actually evicts p_ℓ and then we show that it still holds after greedy-dual-size evicts p_ℓ. Note that greedy-dual-size incurs no cost at the first stage when it decreases credits. Let C be the set of pages in the cache before evicting p_ℓ. The difference between the dual objective value at (r^2, q^2) and the dual objective value at (r^1, q^1) is

$$\left(\sum_{p \in C} s(p) - h\right) \Delta_\ell \ .$$

Let Φ_1 be the value of $\Phi(r^1, q^1)$ using the credits of the cached pages before the decrease that results in the eviction of page p_ℓ, and let Φ_2 be $\Phi(r^2, q^2)$ evaluated using the credits of pages after the decrease which preceded the eviction of p_ℓ. Then we have

$$\Phi_2 - \Phi_1 = \frac{k+1}{k-h+1} \left(\sum_{p \in C} s(p) - h\right) \Delta_\ell - \sum_{p \in C} s(p) \Delta_\ell \geq 0$$

where the last inequality holds since the sum of the sizes of cached items when we execute an eviction step is at least $k+1$. Therefore (r^2, q^2) satisfies Invariant (1) after the decrease in credits.

The eviction of p_ℓ itself increases Φ_2 by exactly $f(p_l)$, the fetching cost associated with p_ℓ, because credit$(p_\ell) = 0$ before evicting p_ℓ. The cost of greedy-dual-size also increases by $f(p_\ell)$ so $\Phi(r^2, q^2)$ bounds the cost of greedy-dual-size after evicting p_ℓ and the invariant holds.

4 Approximating the offline problem

The offline generalized caching problem, where the request sequence is known in advance, is known to be NP-hard [7]. In this section we focus on designing approximation algorithms for the offline generalized caching problem. First we indicate that one can apply the randomized rounding technique of Ragahavan and Thompson [10] to the solution of the linear programming relaxation of the integer program IP_k to obtain a relatively good approximate solution. Furthermore, independently of our work, Albers et. al. [11] and Bar Noy et. al. [1] used the same integer programming formulation to get better approximation algorithms. We shall review these applications of the integer programming formulation IP_k in this section. Finally we show how to formulate the generalized caching problem as a minimum cost unsplittable flow problem on a very simple graph. Whether this formulation can be used to obtain better approximation algorithms for offline generalized caching is an open question, recent general results on unsplittable flow problems [8, 9, 13] when applied to our formulation do not provide better approximation algorithms.

Our integer programming formulation of the generalized caching problem is just a special case of the more general class of *integer covering programs*. Given a n by m matrix A, n dimensional vector b, and m dimensional vector c, all non-negative, an *integer covering program* seeks to minimize $c^t x$ subject to $Ax \geq b$ over the non-negative integer vectors of length m. A key technique to approximate this class of integer programming problems is a rounding method called randomized rounding that has been proposed in [10]. According to this technique we first obtain a solution x^* to the linear programming relaxation of the integer program. Then we compute $x' = \alpha x^*$

for some $\alpha > 1$ and independently for each i, set z_i to be $\lfloor x'_i \rfloor + 1$ with probability $x'_i - \lfloor x'_i \rfloor$, and $\lfloor x'_i \rfloor$ with probability $1 - (x'_i - \lfloor x'_i \rfloor)$. A straightforward application of Chernoff-Hoeffding bounds shows that if $\alpha = 1 + O(\max\{(\ln n/B, \sqrt{(\ln n)/B})\})$ where $B = \min_i\{b_i/\max_j\{A_{ij}\}\}$ then the solution obtained by randomized rounding is within factor of α from optimal with high probability (i.e. with probability $1 - 1/\text{polynomial}(n)$) (see [10, 2]). In our case B is the ratio between the size of the cache and the size of the largest page. Srinivasan [2] refined the analysis of randomized rounding and showed that in fact it produces a solution that is within a factor of $1 + O(\max\{(\ln(nBC/y^*)/B, \sqrt{(\ln 2\lceil nBC/y^* \rceil)/B})\})$ from optimum, where B is defined as before, $C = \max_i c_i$, and y^* is the value of the optimum solution. In our case C is the maximum fetching cost of a page and y^* is the minimum cost of serving the sequence.

Albers et. al. [11] recently used a linear programming formulation similar to IP_k to approximate the generalized caching problem. As in randomized rounding Albers et. al. solve the linear program and suggest a rounding technique to round the fractional solution. Their algorithm finds an integral solution whose cost is within a constant factor of the optimal cost. Their solution, however, is not feasible and needs a cache of size larger by a constant factor times the largest page size. When no additional memory is allowed Albers et. al. obtain an integral solution whose cost is $O(\log(k+C))$ where C is the largest fetching cost.

Recently Bar Noy et. al. [1] obtained a combinatorial approximation algorithm for the generalized caching problem (i.e. an algorithm that does not solve the linear programming relaxation). Their algorithm is based on the local ratio technique [3] which is related to the primal-dual technique [6] for obtaining approximation algorithms to combinatorial optimization problems. They also use the integer programming formulation IP_k, and find a solution that is within a factor 4 of optimum.

Next we suggest a formulation of the generalized caching problem as a min-cost unsplittable multicommodity flow problem on a directed network with $O(n)$ edges where n is the length of the request sequence. For each request we create a node v_i representing the time interval between requests i and $i+1$. We connect these nodes via a simple directed path $v_1, \ldots, v_n$. Each edge (v_i, v_{i+1}) has capacity k and zero cost. In the sequel we use the same notation from Section 3. In particular, for a request j we denote by j' the index of the previous request to the same page. For every request j which is not the first request for a page we introduce two nodes u_j and w_j. Nodes u_j and w_j are a source-sink pair with demand s_j, i.e. s_j units of flow have to be shipped from u_j to w_j. We put an arc from u_j to $v_{j'}$, and an arc from v_j to w_j both with capacity s_j and zero cost. We also put an arc from u_j to v_j with capacity s_j and cost f_j/s_j. It is easy to see that a multicommodity flow that satisfies all the demands corresponds to a fractional solution to the corresponding instance of the generalized caching problem. Furthermore, the cost of the flow equals to the cost of the corresponding fractional solution of the generalized caching problem. An unsplittable muliticommodity flow that satisfies all the demands corresponds to an integral solution to the generalized caching problem with the same cost. Whether this formulation can be used to obtain better approximation algorithms for the generalized caching problem is an open problem.

References

[1] Bar-Noy A., R. Bar-Yehuda, Freund A., Naor S., and B. Schieber. A unified approach to approximating resource allocation and scheduling. In *Proc. 32nd Annual ACM Symposium on Theory of Computing*. ACM, 2000.

[2] Srinivasan A. Improved approximation guarantees for packing and covering integer programs. *SIAM J. Comput.*, 29:648–670, 1999.

[3] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–46, 1985.

[4] Pei Cao and Sandy Irani. Cost-aware www proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, California, December 1997.
`http://www.usenix.org/events/usits97`.

[5] E. Cohen and H. Kaplan. LP-based analysis of greedy-dual-size. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 879–880. ACM-SIAM, 1999.

[6] M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In D. S. Hochbaum, editor, *Approximation algorithms for NP-hard problems.* PWS publishing company, Boston, MA, 1995.

[7] S. Irani. Page replacement with multi-size pages and applications to web caching. In *Proc. 29th Annual ACM Symposium on Theory of Computing*. ACM, 1997.

[8] Jon M. Kleinberg. Single-source unsplittable flow. In *37th Annual Symposium on Foundations of Computer Science*, pages 68–77, Burlington, Vermont, 14–16 October 1996. IEEE.

[9] S. G. Kolliopoulos and Stein C. Improved approximation algorithms for unsplittable flow problems. In *Proc. 38th IEEE Annual Symposium on Foundations of Computer Science*. IEEE, 1997.

[10] P. Raghavan and C. D. Thompson. Provably good routing in graphs: regular arrays. In *Proc. 17th Annual ACM Symposium on Theory of Computing*, pages 79–87. ACM, 1985.

[11] Albers S., Arora S., and Khanna S. Page replacement for general caching problems. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*. ACM-SIAM, 1999.

[12] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[13] Dinitz Y., Garg N., and Goemans M. G. On the single-source unsplittable flow problem. *Combinatorica*, 19:17–41, 1999.

[14] N. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11:525–541, 1994.

[15] N. Young. On line file caching. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*. ACM-SIAM, 1998.