Buying a Constant Competitive Ratio for Paging

János Csirik[1], Csanád Imreh[1]*, John Noga[2]**, Steve S. Seiden[3]***, and Gerhard J. Woeginger[2]**

[1] University of Szeged, Department of Computer Science, Árpád tér 2, H-6720 Szeged, Hungary. {csirik,cimreh}@inf.u-szeged.hu
[2] Institut für Mathematik, Technische Universität Graz, Steyrergasse 30, A-8010 Graz, Austria. {noga,gwoegi}@opt.math.tu-graz.ac.at
[3] Department of Computer Science, 298 Coates Hall, Louisiana State University, Baton Rouge, LA 70803, USA. sseiden@acm.org

Abstract. We consider a variant of the online paging problem where the online algorithm may buy additional cache slots at a certain cost. The overall cost incurred equals the total cost for the cache plus the number of page faults. This problem and our results are a generalization of both, the classical paging problem and the ski rental problem.
We derive the following three tight results: (1) For the case where the cache cost depends linearly on the cache size, we give a λ-competitive online algorithm where $\lambda \approx 3.14619$ is a solution of $\lambda = 2 + \ln \lambda$. This competitive ratio λ is best possible. (2) For the case where the cache cost grows like a polynomial of degree d in the cache size, we give an online algorithm whose competitive ratio behaves like $d/\ln d + o(d/\ln d)$. No online algorithm can reach a competitive ratio better than $d/\ln d$. (3) We exactly characterize the class of cache cost functions for which there exist online algorithms with finite competitive ratios.

1 Introduction

The classical problem. The paging problem considers a two level memory system where the first level (the *cache*) can hold k pages, and where the second level (the *slow memory*) can store n pages. The n pages ($n \gg k$) in slow memory represent virtual memory pages. A *paging algorithm* is confronted with a sequence of requests to virtual memory pages. If the page requested is in the cache (a page *hit*), no cost is incurred; but if the page is not in the cache (a page *fault*), then the algorithm must bring it into the cache at unit cost. Moreover, the algorithm must decide which of the k pages currently in cache to evict in order to make room for the newly requested page.

The paging problem has inspired several decades of theoretical and applied research and has now become a classical problem in computer science. This is

* Supported by the Hungarian National Foundation for Scientific Research, Grant T030074
** Supported by the START program Y43-MAT of the Austrian Ministry of Science.
*** Supported by the Research Competitiveness Subprogram of the Louisiana Board of Regents.

due to the fact that managing a two level store of memory has long been, and continues to be, a fundamentally important problem in computing systems. The paging problem has also been one of the cornerstones in the development of the area of online algorithms. Starting with the seminal work of Sleator & Tarjan [7] which initiated the recent interest in the competitive analysis of online algorithms, the paging problem has motivated the development of many important innovations in this area.

In the *offline* version of the paging problem the request sequence is a priori known to the algorithm. Belady [1] gives a polynomial time optimal offline algorithm for paging. In the *online* version of the paging problem each request must be served without any knowledge of future requests. An online algorithm is *R-competitive* if on all possible request sequences the ratio of the algorithm's cost to the optimal offline cost is bounded by the constant R. The *competitive ratio* of an online paging algorithm is the smallest such constant R. Sleator & Tarjan [7] show that for cache size k, the online algorithm *LRU* (which always evicts the *least recently used* page) has a competitive ratio of k and that no better ratio is possible. In fact, they prove the following more general result that we will use many times in this paper.

Proposition 1. *(Sleator & Tarjan [7])*
If LRU with cache size k is compared against an optimal offline algorithm with cache size ℓ, then LRU has a competitive ratio of $k/(k-\ell+1)$.

No better competitive ratio is possible: For every online algorithm A with cache size k, there exist arbitrarily long request sequences σ such that A faults on every page of σ whereas the optimal offline algorithm with cache size ℓ only faults on a fraction $(k-\ell+1)/k$ of σ. □

For more information on online paging we refer the reader to the survey chapter of Irani [5] and to the book of Borodin & El Yaniv [2].

The problem considered in this paper. In all previous work on online paging a basic assumption was that the cache size k is fixed a priori and cannot be changed. In this paper, we consider the situation where at any moment in time the algorithm may increase its cache size by purchasing additional cache slots. If its final cache size is x, then it is charged a cost $c(x)$ for it. Here $c : \mathbb{N} \to \mathbb{R}^+$ is a non-decreasing, non-negative cost function that is a priori known to the algorithm. An equivalent way of stating this problem is the following: The algorithm starts with no cache. At each request, the algorithm may increase its cache from its current size x_1 to a larger size x_2 at a cost of $c(x_2) - c(x_1)$.

We have three basic motivations for looking at this problem. First, and most importantly, the classical online analysis of paging is criticized for being overly pessimistic. For instance, *LRU* (and all other deterministic algorithms) cannot have a competitive ratio smaller than the size of the cache, which can be arbitrarily large. However, in practice *LRU* typically performs within a small constant ratio of optimal for all cache sizes. For any particular instance of our problem any algorithm will either have a constant competitive ratio or not be competitive. Second, we see this version as a first approximation to the problem of deciding when and how to upgrade memory systems. Though our version ignores many practical concerns, it incorporates enough to illustrate ideas like amortizing cost over time and balancing cost against performance. Third, we can model a system where a portion of memory is reserved for the use of one high priority process (or set of processes) and balance the faults incurred by this process against the decreasing system performance for all other processes.

The offline version of this problem is quite easy to solve: For a given cost function $c(k)$ and a given request sequence σ, there are only a polynomial number of possible optimal cache sizes (from 0 to the length of the request sequence). All of them can be checked in polynomial time by computing the cost of Belady's algorithm [1] on the sequence σ with this cache size, and by adding the cache cost to the result. The best such solution yields the global optimum. Note that for many special cost functions $c(k)$ (e.g., convex functions) there are even faster offline methods for finding the optimal cache size.

Now let us turn to the online version of this problem. If the function $c(x)$ has the form $c(x) = 0$ for $x \leq k$ and $c(x) = \infty$ for $x > k$, then we are back at classical paging with cache size k. Moreover, our problem captures some features of classical renting-versus-buying problems (the most simple instance of which is the well known ski rental problem (see Karlin, Manasse, McGeoch & Owicki [6])). Imreh & Noga [4] analyze machine scheduling problems where the online algorithm may adjust its resources at an additional cost.

Notation. For a fixed algorithm A and a given request sequence σ, we denote by $\text{fault}_A(\sigma)$ the number of page faults that A incurs on σ, and we denote by $\text{cache}_A(\sigma)$ the cost $c(x)$ where x is the final cache size of A on σ. Moreover, we define $\text{cost}_A(\sigma) = \text{fault}_A(\sigma) + \text{cache}_A(\sigma)$. An online algorithm A is called *R-competitive* if there is a fixed constant b such that for all request sequences σ we have

$$\text{cost}_A(\sigma) \leq R \cdot \text{cost}_{OPT}(\sigma) + b. \tag{1}$$

The smallest R for which an online algorithm is R-competitive is its competitive ratio. The optimal offline algorithm will be denoted by OPT.

Organization of the paper. In Section 2 we summarize and explain our three main results. The proof of the result on linear cost functions can be found in Sections 3 (proof of the positive result) and 4 (proof of the negative result), respectively. The results on polynomial cost functions are proved in Section 5. The characterization of cost functions that allow finite competitive online algorithms is proved in Section 6. Finally, we end with a short conclusion in Section 7.

2 Our Results

We start by discussing several 'natural' cost functions $c(x)$. In the simplest case the cost of the cache is proportional to its size and each cache slot costs much more than a single page fault.

Theorem 1. *Assume that the cache cost is $c(x) = \alpha x$ where α is some positive real. Let $\lambda \approx 3.14619$ be the largest real solution of*

$$\lambda = 2 + \ln \lambda. \tag{2}$$

Then there exists an online algorithm for the cache purchase problem with competitive ratio λ. For any $r < \lambda$ there exists an α such that no online algorithm can be r-competitive.

The equation (2) has two real solutions, where the smaller one is approximately 0.15859 and where the larger one is $\lambda \approx 3.14619$. For users of MAPLE we remark that $\lambda = -W(-e^{-2})$ where $W(\cdot)$ is the well known Lambert W function [3].

We also note that in the (easier, but somewhat unreasonable) case where cache slots are not much more expensive than a single fault (i.e., where α is close to 1), an online algorithm can be $(1 + \alpha)$-competitive by simply purchasing a cache location for each new item requested and never evicting any item. Besides linear cost functions, another fairly natural special case considers polynomial cost functions.

Theorem 2. *Assume that the cache cost is $c(x) = x^d$ with $d \geq 2$. Then there exists an online algorithm for the cache purchase problem whose competitive ratio grows like $d/\ln d + o(d/\ln d)$. Moreover, no online algorithm can reach a competitive ratio that is better than $d/\ln d$.*

Finally, we will exactly characterize the class of cost functions c for which there exist online algorithms with finite competitive ratios.

Theorem 3. *There exists an online algorithm with finite competitive ratio for online paging with cache purchasing cost $c(x)$ if and only if*

$$\exists q > 1 \ \exists p > 0 \ \exists s \geq 0 \ \exists X > 0 \ \forall x \geq X : \quad c(qx) \leq p \cdot c(x) + s. \tag{3}$$

One way to interpret the condition described above is that the cost function $c(x)$ must be polynomially bounded.

3 The Positive Result for Linear Cost Functions

In this section we prove the positive result claimed in Theorem 1 for linear cost functions of the form $c(x) = \alpha x$ with $\alpha > 0$. We consider the online algorithm *BETA* that uses *LRU* as its paging strategy, and that increases its cache size to ℓ as soon as $\text{fault}_{BETA} \geq \alpha\beta(\ell - 1)$ holds. Here the critical parameter β equals $1/\ln \lambda$ where λ was defined in equation (2). Then $\beta \approx 0.872455$, and it can be checked that β is the unique positive root of $2 + 1/\beta = e^{1/\beta}$.

Lemma 1. *If λ is defined as in (2) and $\beta = 1/\ln \lambda$, then any real $y \geq 0$ satisfies the inequality*

$$(\beta + 1)y \leq \lambda(1 + \beta y - \beta - \beta \ln y).$$

Proof. First note that by the definition of β we have $\ln(\frac{\beta\lambda}{\beta\lambda-\beta-1}) = 1/\beta$. Next we will apply the well known inequality $z+1 \le e^z$ for real numbers z. Setting $z \doteq y \cdot \frac{\beta\lambda-\beta-1}{\beta\lambda} - 1$ in this inequality, taking logarithms, and applying some algebra yields

$$\ln y \;\le\; \ln\left(\frac{\beta\lambda}{\beta\lambda-\beta-1}\right) + y \cdot \frac{\beta\lambda-\beta-1}{\beta\lambda} - 1 \;=\; \frac{1}{\beta} + y \cdot \frac{\beta\lambda-\beta-1}{\beta\lambda} - 1.$$

It is easily verified that this inequality leads to the claimed inequality. □

Now consider an arbitrary request sequence σ. Let k denote the final cache size of *BETA* when fed with σ, and let ℓ denote the optimal offline cache size for σ. We denote by f_i the total number of page faults that *BETA* incurs up to the moment when it purchases the ith cache slot. Then between buying the $(i+1)$th and the ith slot, *BETA* has $f_{i+1} - f_i = \alpha\beta$ faults.

Lemma 2. *For a sequence σ on which BETA purchases a cache of size k and OPT uses a cache of size ℓ*

$$\text{fault}_{OPT}(\sigma) \ge \sum_{i=\ell}^{k-1} (f_{i+1} - f_i)(i - \ell + 1)\frac{1}{i}.$$

Proof. If $\ell \ge k$ then the lemma is trivial.

For $\ell < k$, we prove the lemma by modifying the standard technique of dividing the request sequence into phases. The first phase will begin with the first request. If at some point i is the current size of *BETA*'s cache, i distinct items have been requested during the current phase, and an item distinct from the i items requested during this phase is the next request then end the current phase and begin the next phase with the next request.

Consider a given phase which ends with *BETA*'s cache size equal to i. Since *BETA* uses *LRU* as its paging strategy and because of the way a phase is ended, during this phase *BETA* will fault on any item at most once. On the other hand between the second request in this phase and the first request of the next phase the optimal algorithm must fault at least $i - \ell + 1$ times. If the size of *BETA*'s cache was also i at the beginning of the phase then this phase contributes exactly what is needed to the sum. If however *BETA*'s cache size was smaller then this phase contributes (slightly) more than is necessary to the sum. □

We now prove the positive result in Theorem 1.

Proof. First assume that $k \le \ell$. Since offline purchases ℓ cache slots the offline cost is at least $\alpha\ell$. Since *BETA* did not purchase the $(k+1)$-th cache slot, at the end $\text{fault}_{BETA}(\sigma) < \alpha\beta k$ and $\text{cache}_{BETA}(\sigma) = \alpha k$. Since $\beta + 1 \le \lambda$, the cost of *BETA* is at most a factor of λ above the optimal offline cost.

Now assume that $k > \ell$. Using the previous lemma:

$$\begin{aligned}\mathrm{cost}_{OPT}(\sigma) &= \mathrm{cache}_{OPT}(\sigma) + \mathrm{fault}_{OPT}(\sigma) \\ &\geq \alpha\ell + \sum_{i=\ell}^{k-1}(f_{i+1} - f_i)(i - \ell + 1)\frac{1}{i} \\ &\geq \alpha\ell + \alpha\beta\sum_{i=\ell}^{k-1}(i-\ell)\frac{1}{i} \\ &\geq \alpha\ell + \alpha\beta\left(k - \ell - \ell\ln\frac{k}{\ell}\right).\end{aligned}$$

Since the online cache cost is αk and the online fault cost is at most $\alpha\beta k$, by substituting $y \doteq k/\ell > 1$ we have

$$\begin{aligned}\mathrm{cost}_{BETA}(\sigma)/\mathrm{cost}_{OPT}(\sigma) &\leq \alpha(\beta+1)k \;/\; \left(\alpha\ell + \alpha\beta\left(k - \ell - \ell\ln\frac{k}{\ell}\right)\right) \\ &= (\beta+1)y \;/\; (1 + \beta y - \beta - \beta\ln y) \quad \leq \quad \lambda.\end{aligned}$$

Here we used Lemma 1 to get the final inequality. Therefore, algorithm *BETA* indeed is a λ-competitive online algorithm for paging with cache purchasing cost $c(x) = \alpha x$. This completes the proof of the positive statement in Theorem 1. □

4 The Negative Result for Linear Cost Functions

In this section we prove the negative result claimed in Theorem 1 for linear cost functions of the form $c(x) = \alpha x$ with $\alpha \gg 1$. The proof is done by an adversary argument.

Assume that there is an online algorithm $\mathcal{A}$ which is r-competitive for some $1 < r < \lambda$ and that α is very large. Further, assume that the pages are numbered 1,2,3,.... The adversary always requests the smallest numbered page which is not in the online cache. Thus, the online algorithm faults on every request. Let σ be the infinite sequence of requests generated by this procedure.

In order to be finitely competitive, the online algorithm cannot have any fixed upper bound on the size of its cache; hence, the number of purchased slots is unbounded. Let f_i be the number of requests (respectively, the number of page faults) which precede the purchase of the ith cache slot. Note that $f_1 = 0$, and that $f_1, f_2, \ldots$ is a monotone non-decreasing integer sequence. The cost to $\mathcal{A}$ for the requests $1, \ldots, f_i$ equals $i\alpha + f_i$.

We will now consider the adversary's cost for requests $\sigma_i = 1, \ldots, f_i$. Guided by the results of Sleator & Tarjan [7] (see Proposition 1) we upper bound the adversary's cost in the following lemma:

Lemma 3. *If $\mathcal{A}$ purchases the jth cache slot after f_j faults and OPT uses a cache of size ℓ then*

$$\mathrm{fault}_{OPT}(\sigma_i) \;\leq\; i + \sum_{j=\ell}^{i-1}(f_{j+1} - f_j)\frac{j - \ell + 1}{j}.$$

Proof. Any algorithm will fault the first time a particular page is requested. These i faults correspond to the first term in the right hand side of the above inequality. Below we will explicitly exclude these faults from consideration.

As in Lemma 2 we divide the request sequence into phases. However, this time we do this in a slightly different way. The first phase begins with the first request. If j is the size of $\mathcal{A}$'s cache and j requests have been made during the current phase then the current phase ends and the next request begins a new phase. Note that this differs slightly from Lemma 2, since the first request in a phase need not be distinct from all requests in the previous phase.

Consider a phase which begins with the size of $\mathcal{A}$'s cache equal to j and ends with the size of $\mathcal{A}$'s cache equal to j'. During this phase there will be j' requests from items $1, 2, \ldots, j'+1$. Note that none of the items labeled $j+2, \ldots, j'+1$ will have been requested in any previous phase, since the size of $\mathcal{A}$'s cache prior to this phase was j. Recall that the first fault on each of these $j'-j$ items has already been counted.

From this point we proceed in a fairly standard manner. We claim that an optimal offline algorithm will incur at most $j'-\ell+1$ faults. Note that this is equivalent to stating that the optimal offline algorithm will *not* fault on $\ell-1$ requests.

If possible, whenever the offline algorithm faults it evicts an item which will not be requested in the remainder of this phase. If at some point it is not possible for the offline algorithm to evict such an item then all ℓ items in its cache will be requested later in this phase. In this case, it is easy to see that there will be at least $\ell-1$ requests on which the offline algorithm will not fault. On the other hand, if the offline algorithm is able to evict $j'-\ell+1$ items which will not be requested later in this phase then its cache contains all of the (at most ℓ distinct) items which will be requested during the remainder of the phase.

Of the $j'-\ell+1$ faults which the offline algorithm will incur during this phase, $j'-j$ faults correspond to the first time an item is requested (these $j'-j$ faults have already been counted). So this phase will contribute $j-\ell+1$ faults to the sum. If $j'=j$ then this phase contributes precisely what is claimed. If instead $j'>j$ this phase contributes (slightly) less. □

We now prove the lower bound of λ on any online algorithm.

Proof. Fix an online algorithm $\mathcal{A}$. For a given α, if f_i/i is not bounded above then $\mathcal{A}$ cannot have a constant competitive ratio. Clearly, f_i/i is bounded below by 0 (for $i \geq 1$). So $L = \liminf_i \frac{f_i}{i}$ exists. Suppose that the adversary initially purchases i/λ cache locations and serves σ_i with only these locations. From the definition of L, we know that for any ϵ there are arbitrarily large M such that $f_i/i \geq L-\epsilon$ for all $i \geq M/\lambda$. Further for sufficiently large M, $\left|\sum_{j=M/\lambda}^{M-1} 1/j - \ln(\lambda)\right| \leq \epsilon$. Using the previous lemma we get

$$\frac{\mathrm{cost}_{\mathcal{A}} - b}{\mathrm{cost}_{OPT}} \geq \frac{\alpha M + f_M - b}{\alpha M/\lambda + M + \sum_{j=M/\lambda}^{M-1}(f_{j+1}-f_j)\frac{j-M/\lambda+1}{j}}$$

$$= \frac{\alpha M + f_M - b}{\alpha M/\lambda + M + f_M \frac{M-M/\lambda}{M-1} + \sum_{j=M/\lambda+1}^{M-1} f_j \frac{1-M/\lambda}{j(j-1)} - f_{M/\lambda} \frac{\lambda}{M}}$$

$$\geq \frac{\alpha + L - b/M - \epsilon}{\alpha/\lambda + 1 + (L+2\epsilon)(1 - 1/\lambda - 1/\lambda(\ln(\lambda) - \epsilon) + (\ln(\lambda)+\epsilon)/M)}$$

$$\geq \frac{\alpha + L - b/M - \epsilon}{\alpha/\lambda + 1 + (L+2\epsilon)/\lambda + (L+2\epsilon)(\ln(\lambda)+\epsilon)/M}.$$

In the final inequality we have used that $1/\lambda = 1 - 1/\lambda - 1/\lambda \ln(\lambda)$. For α and M sufficiently large, the final value can be made arbitrarily close to λ. □

5 The Results for Polynomial Cost Functions

In this section we prove the two results (one positive and one negative) that are claimed in Theorem 2 for cost functions of the form $c(x) = x^d$ with $d \geq 2$.

We start with the proof of the positive result.

Proof. Let $\varepsilon > 0$ be a small real number. Similar to Section 3 we consider an online algorithm *BETA2* that uses *LRU* as its paging strategy. This time the online algorithm increases its cache size to k as soon as it has incurred at least $d^\varepsilon (k-1)^d$ page faults. We will show that this algorithm has a competitive ratio of at most $d(1+d^{-\varepsilon})/[(1-\varepsilon)\ln d]$.

Consider an arbitrary request sequence σ. Let k denote the final cache size of *BETA2*, and let ℓ denote the optimal offline cache size for σ. If $\ell \geq k+1$, the offline cost is at least $(k+1)^d$ and the online cost is at most $k^d + d^\varepsilon (k+1)^d$. Then the online cost is at most a factor of $1 + d^\varepsilon$ above the offline cost. From now on we assume that $\ell \leq k$. For $i = \ell, \ldots, k$ we denote by f_i the total number of page faults that *BETA2* incurs until the moment when it purchases the ith cache slot. Using an argument similar to that of Lemma 2, we get that the optimal offline algorithm incurs at least $\sum_{i=\ell}^{k-1} (f_{i+1} - f_i)(i - \ell + 1)\frac{1}{i}$ page faults during this time. Therefore,

$$\begin{aligned}
\mathrm{cost}_{OPT}(\sigma) &\geq \ell^d + \sum_{i=\ell}^{k-1} (f_{i+1} - f_i) \frac{i - \ell + 1}{i} \\
&\geq \ell^d + d^\varepsilon \sum_{i=\ell}^{k-1} ((i+1)^d - i^d) \frac{i - \ell}{i} \\
&= \ell^d + d^\varepsilon (k^d - \ell^d) - d^\varepsilon \ell \sum_{i=\ell}^{k-1} ((i+1)^d - i^d) \frac{1}{i} \\
&\geq \ell^d + d^\varepsilon (k^d - \ell^d) - d^\varepsilon \ell \int_\ell^k \frac{d \cdot x^{d-1}}{x}\, dx
\end{aligned}$$

$$= \ell^d + d^\varepsilon(k^d - \ell^d) - \ell\frac{d^{1+\varepsilon}}{d-1}(k^{d-1} - \ell^{d-1})$$
$$\approx \ell^d + d^\varepsilon(k-\ell)k^{d-1}.$$

The online cache cost is approximately $\mathrm{cost}_{BETA2}(\sigma) \approx (1+d^\varepsilon)k^d$. Substituting $y := \ell/k \le 1$ and putting things together we get

$$\mathrm{cost}_{BETA2}(\sigma)/\mathrm{cost}_{OPT}(\sigma) \;\le\; \frac{(1+d^\varepsilon)k^d}{\ell^d + d^\varepsilon(k-\ell)k^{d-1}} \;=\; \frac{1+d^\varepsilon}{y^d + d^\varepsilon(1-y)}. \qquad (4)$$

The denominator in the right hand side of (4) is minimized for $y = d^{(\varepsilon-1)/(d-1)}$ and hence is at least

$$d^{(\varepsilon-1)d/(d-1)} + d^\varepsilon\left(1 - d^{(\varepsilon-1)/(d-1)}\right) \;\ge\; d^\varepsilon\left(1 - d^{(\varepsilon-1)/(d-1)}\right).$$

By applying some (tedious) calculus we get that as d tends to infinity, the function $1 - d^{(\varepsilon-1)/(d-1)}$ grows like $(1-\varepsilon)\frac{\ln d}{d}$. By combining these observations with the inequality in (4), we conclude that the competitive ratio R of *BETA2* is bounded by

$$R \;\le\; \frac{(1+d^{-\varepsilon})d}{(1-\varepsilon)\ln d}.$$

This completes the proof of the positive statement in Theorem 2. $\square$

We turn to the proof of the negative statement in Theorem 2 which is done by an adversary argument.

Proof. Consider an r-competitive online algorithm for cost functions of the form $c(x) = x^d$. The pages are numbered 1,2,3,..., and the adversary always requests the smallest numbered page which is not in the online cache. Thus, the online algorithm faults on every request. In order to have a finite competitive ratio, the online algorithm cannot run forever with the same number of slots. Hence, the number of purchased slots must eventually exceed gr where g is a huge integer. Suppose that after i_g requests the online cache is extended for the first time to a size $k \ge gr$. Then the corresponding online cost at this moment is at least $i_g + (gr)^d$.

Now consider the optimal offline algorithm for cache size $gr - g + 1$. Since the online algorithm was serving the first i_g requests with a cache size of at most $gr - 1$, the results of Sleator & Tarjan [7] (see Proposition 1) yield that the number of offline faults is at most

$$i_g \cdot \frac{(gr-1) - (gr-g+1) + 1}{gr-1} \;=\; \frac{i_g(g-1)}{gr-1} \;\le\; \frac{i_g}{r}.$$

The offline cache cost is $(gr-g+1)^d$. Since the online algorithm is r-competitive, there exists a constant b such that the following inequality is fulfilled for all integers g; cf. equation (1):

$$i_g + (gr)^d \;\le\; r \cdot \left(\frac{i_g}{r} + (gr-g+1)^d\right) + b.$$

Since g can be arbitrarily large, this implies $r^d \le r(r-1)^d$ which is equivalent to

$$\frac{1}{r} \le \left(1 - \frac{1}{r}\right)^d . \tag{5}$$

Now suppose that $r < d/\ln d$. Then the left hand side in (5) is at least $\ln d/d$, whereas the right hand side is at most $1/d$. This contradiction completes the proof of the negative statement in Theorem 2. □

6 The Results for the General Case

In this section we prove the two results (one positive and one negative) that are claimed in Theorem 3.

We start with the proof of the positive result.

Proof. So we assume that the cost function $c(x)$ satisfies condition (3). Fix a request sequence σ. We may assume that the optimal offline algorithm for σ uses a cache of size $x_{OPT} \ge X$. The case when $x_{OPT} < X$ can be disregarded by making b in the definition of competitiveness sufficiently large; in fact, any b greater than $c(X)$ will do.

Consider the algorithm BAL which uses LRU as its paging strategy and which tries to balance its cache cost and its fault cost. In other words, BAL increases its cache size to x as soon as $\text{fault}_{BAL} \ge c(x)$. Until the time where BAL purchases a cache of size $q \cdot x_{OPT}$, the cost ratio of online to offline is at most $2p$: At this time $\text{cache}_{BAL} = c(qx_{OPT}) \le p \cdot c(x_{OPT})$ and $\text{fault}_{BAL} \approx \text{cache}_{BAL}$, whereas the offline cost is at least $c(x_{OPT})$. From the time where BAL purchases a cache of size $q \cdot x_{OPT}$ onwards, the ratio is at most $2q/(q-1)$: By using the result of Sleator & Tarjan as stated in Proposition 1 with $\ell = x_{OPT}$ and $k = q \cdot x_{OPT}$, we get that $\text{fault}_{BAL}/\text{fault}_{OPT} \le qx_{OPT}/(qx_{OPT} - x_{OPT} + 1)$. Therefore,

$$\text{cost}_{BAL} \approx 2\,\text{fault}_{BAL} \le 2\,\frac{q \cdot x_{OPT} \cdot \text{fault}_{OPT}}{qx_{OPT} - x_{OPT} + 1} < \frac{2q}{q-1}\,\text{cost}_{OPT}.$$

To summarize, we have shown that BAL is $\max\{2p, 2q/(q-1)\}$-competitive. This completes the argument for the positive result. □

Now let us turn to the negative result.

Proof. So we assume that the cost function $c(x)$ does not satisfy the condition (3), and that therefore

$$\forall q > 1\ \forall p > 0\ \forall s \ge 0\ \forall X > 0\ \exists x \ge X : \quad c(qx) > p \cdot c(x) + s. \tag{6}$$

The idea is quite simple. If OPT uses a cache of size x then an online algorithm which wants to be R competitive must eventually purchase a cache of size px for some $p \approx R/(R-1)$. The result of Sleator and Tarjan as stated in Proposition 1 requires that $R \cdot \mathrm{fault}_{OPT} - \mathrm{fault}_A$ cannot be too large. On the other hand, $Rc(x) - c(px)$ can be made arbitrarily large, since c is not polynomially bounded.

Now for the details. We will proceed by contradiction. Assume that there is an algorithm A which is R-competitive for some $R > 1$ and fix b as in the definition of competitiveness. By using (6) we can choose x to satisfy

$$c\left(x\,\frac{2R-1}{2R-2}\right) \;>\; R \cdot c(x) + x(2R-1)/2 + R + b.$$

If we use the lower bound sequence from Proposition 1 for $k = x(2R-1)/(2R-2) - 1$ and $\ell = x$ until A purchases a cache of size $x(2R-1)/(2R-2)$, then $R \cdot \mathrm{fault}_{BEL}(\sigma) - \mathrm{fault}_A(\sigma) \le x(2R-1)/2$. Note that A must eventually purchase a cache of this size, since otherwise cost_A will tend to ∞ while $\mathrm{cost}_{OPT} \le c((2R-1)x/(2R-2)) + (2R-1)x/(2R-2)$. Therefore,

$$\begin{aligned}\mathrm{cost}_A(\sigma) &= \mathrm{cache}_A(\sigma) + \mathrm{fault}_A(\sigma)\\ &> R \cdot \mathrm{cache}_{BEL}(\sigma) + R \cdot \mathrm{fault}_{BEL}(\sigma) + b\\ &= R \cdot \mathrm{cost}_{OPT}(\sigma) + b.\end{aligned}$$

This contradiction completes the proof of the negative result in Theorem 3. □

7 Conclusion

We have considered a simple model of caching which integrates the ability to add additional cache locations. A number of results for linear, polynomial, and arbitrary cost functions have been found. One possible direction for further study of this problem is to consider the degree to which randomization can further reduce the competitive ratios found in this paper. The primary difficulty when attacking the randomized version of the problem is finding relationships between the costs of the online and offline algorithms when the cache sizes are unequal.

References

1. L.A. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal 5*, 78–101, 1966.
2. A. Borodin and R. El Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.
3. R.M. Corless, G.H. Gonnet, D.E.G. Hare, D.J. Jeffrey, and D.E. Knuth. On The Lambert W Function. Maple Share Library.
4. Cs. Imreh and J. Noga. Scheduling with Machine Cost. Proceedings 2nd International Workshop on Approximation Algorithms, Springer LNCS 1671, 168–176, 1999.
5. S. Irani. Competitive analysis of paging. In A. Fiat and G.J. Woeginger (eds.) *Online Algorithms – The State of the Art.* Springer LNCS 1442, 52–73, 1998.
6. A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica 11*, 542–571, 1994.
7. D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM 28*, 202–208, 1985.