

SIGACT News Online Algorithms Column 2

Wolfgang W. Bein and Lawrence L. Larmore
School of Computer Science
University of Nevada
Las Vegas, NV 89154
bein@cs.unlv.edu, larmore@cs.unlv.edu

1 The Paging Problem and the Bookmarks Issue

In this column, we examine the issue of competitiveness-complexity tradeoff. The “goodness” of an online algorithm is typically measured by its competitiveness (the lower the better), but (without ignoring competitiveness) we address other resource issues here, such as memory. We will focus on randomized online algorithms for the well-known *paging problem*, where an operating system tries to make optimal use of a small fast memory and a large slow memory, but our discussion extends naturally to the classic *server problem* and to other online problems as well.

We will assume that an operating system has a *cache* which can hold exactly k *pages*, that there are arbitrarily many pages in some more distant *slow memory*, and that a running job requests a sequence of pages which cannot be predicted in advance. If a requested page is already in the cache, that is called a *hit* and costs zero, while if it is not, that is called a *fault* and the requested page must be moved into the cache and an existing cache page ejected, at cost 1. One optimal offline algorithm for the paging problem is FFU (**F**arthest **F**orward **U**se) which always ejects the cache page whose future use is farthest away. In case there are multiple pages in the cache which will never be used again, FFU ejects one arbitrarily. An online algorithm for the cache problem is said to be C -*competitive* if its cost (expected cost in the case of a randomized algorithm) does not exceed C times the cost of FFU for the given request sequence, plus a constant. We view competitive analysis as a money game between us and the adversary. When we (using an online algorithm) incur a cost, we must pay in one currency (say dollars). When the adversary (who presumably uses an optimal offline algorithm) incurs a cost, it owes us that much in another currency (say euros). We can exchange each euro for C dollars. We are allowed a fixed initial balance, and, if our algorithm is C -competitive, we will never run out of money. (In the randomized case, this means instead, that our *expected* balance will never be negative.)

In the paging problem, the memory¹ of an online algorithm consists of “bookmarks,” which are names of pages in slow memory that have been ejected and might (possibly) be requested later, and what we’ll call “regular memory,” which is everything else, such as the algorithm’s state, marks on cache positions, or whatever. In particular, we will examine *trackless* online algorithms for the paging problem, *i.e.*, algorithms which do not keep any bookmarks, and also online algorithms which are allowed to keep only a limited number of bookmarks. It has been argued [7] that keeping bookmarks is impractical. Yet giving them up prevents us from achieving the best possible competitiveness. Without a bookmark, an online algorithm, in the case of a fault, cannot distinguish between a request to a previously used but ejected page and a request

¹Not to be confused with the cache or the slow memory.

to an entirely new page. But how can we get optimal competitiveness with as few bookmarks as possible?

2 Keeping Track of Everything

We will assume that the reader is familiar with the distributional model of randomized online algorithms. Unlike the behavioral model, which gives the probability that an algorithm makes a certain choice at step t given its state at step $t - 1$ and the input at step t , the distributional model gives the probability that it is at any given state at step t , given inputs for the first t steps. It is long established that the two models are equivalent. (See, for example, [6]).

No deterministic online algorithm for the k -paging problem can have competitiveness less than k , and no randomized online algorithm for the k -paging problem can have competitiveness less than H_k , where $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} = \Theta(\log k)$, the so-called k^{th} harmonic number. However, these lower bounds do not address resource questions.

The first randomized online algorithm for the paging problem to achieve the optimal competitiveness of H_k was PARTITION [16]. Suppose that the adversary uses FFU, which is optimal for every prefix of the request sequence. For a given request sequence r^1, \dots, r^n , and for any $t \leq n$, there is always a finite set of k -tuples of pages (called the *support at time t*) which has property that, without loss of generality, the adversary's cache at time t will be some member of that support. For example, suppose that $k = 2$ and the initial cache is $\{a, b\}$, and suppose that the first two requests are ca . After the first request, since the adversary can see the future, its cache is actually $\{a, c\}$ for sure, but, not knowing the future, we only know that the adversary's cache is either $\{a, c\}$ or $\{b, c\}$. The support after 2 steps is $\{a, c\}$, since, after we read the second request of a , we know that the adversary must be there.

The distribution definition of PARTITION is that our cache is equal to one of the support configurations with some probability. Consider the above example, *i.e.*, request sequence ca . Our cache is initially $\{a, b\}$ and is $\{a, c\}$ after two steps, but after one step is $\{a, c\}$ with probability $\frac{1}{2}$ and $\{b, c\}$ with probability $\frac{1}{2}$. The expected cost for us to serve the request sequence is thus $\frac{3}{2}$, compared to the optimal cost of 1.

In general for $k = 2$, PARTITION calls for a uniform distribution over all configurations in the support.² Since the cardinality of the support could be as great as the number of requests, the number of bookmarks that we must maintain is unbounded. For example, if the request sequence is $cdefgh$, and if our cache is $\{e, h\}$, we have bookmarks for pages a, b, c, d, f, g .

3 Forgive and Forget

But, do we really need to remember all that? The answer is no. In fact, Achlioptas *et al.* [1, 2] showed that, for fixed k , the optimal competitiveness H_k can be achieved while maintaining a constant number of bookmarks (although that constant is a rapidly growing function of k). Their algorithm is called EQUITABLE.

We can suspect that remembering the entire support is not necessary, by looking at a simple example. Suppose $k = 2$, the initial cache is $\{a, b\}$, and the first three requests are cde . Then

- *support at time 0* = $\{\{a, b\}\}$
- *support at time 1* = $\{\{a, c\}, \{b, c\}\}$

²We refer the reader to the original paper for PARTITION's distribution if $k \geq 3$.

- *support at time 2* = $\{\{a, d\}, \{b, d\}, \{c, d\}\}$
- *support at time 3* = $\{\{a, e\}, \{b, e\}, \{c, e\}, \{d, e\}\}$

and, if we use PARTITION, after step 3 we have three bookmarks. Now suppose that the fourth and last request is either a, b, c , or d . The optimal cost is 3, while our expected cost is $3 + \frac{3}{4} = 3.75$, since the last request is a hit with probability $\frac{1}{4}$. The ratio of costs is $\frac{3.75}{3} = \frac{5}{4}$, which is considerably less than $\frac{3}{2}$. The adversary has been “wasteful” in choosing that request sequence. Why can’t we be wasteful ourselves, and simply forget (lose track of) some of those pages? We can, and still maintain a competitive ratio of $\frac{3}{2}$. After 3 steps, assuming we use PARTITION, we have been paid 3 euros, *i.e.*, 4.5 dollars³ but have paid only 3 dollars, which means that we’re ahead, in fact unnecessarily far ahead, in the game. We exploit this fact to avoid remembering more than two bookmarks for the paging problem for $k = 2$, using *forgiveness*. As long as the number of bookmarks does not exceed two, we will follow the rules of PARTITION. If the number of bookmarks would rise to three, we “forgive and forget,” arriving again at a situation where there are no bookmarks.

Continuing our analysis of the above example, after two steps, we have paid two dollars, the adversary has paid us two euros, and we know that the adversary is at $\{a, d\}$, $\{b, d\}$, or $\{c, d\}$. Using PARTITION, our cache is each of the three possible support configurations with probability $\frac{1}{3}$, and the other two pages are bookmarked.

Now consider the third request, e . PARTITION would then demand one more euro, and keep track of the new support, which has three configurations. Instead, we deterministically eject the least recently used page, erase all bookmarks, and demand no payment. Our cache is now $\{d, e\}$, and we pretend that the support set is $\{\{d, e\}\}$. This move costs us 1. In this new situation, we have paid 3 dollars and the adversary has paid us 2 euros.

We are assuming that the adversary’s cache is $\{d, e\}$, although it might not be. What is really happening is that, despite the fact that the adversary owes us at least one more euro, we partially **forgive** the debt, saying that it owes us nothing if its cache is $\{d, e\}$, and that it owes us one euro if it is at $\{e, x\}$, where x is any page other than d . We then proceed to **forget** all pages other than d and e . In fact, the algorithm forgets the true support $\{\{a, e\}, \{b, e\}, \{c, e\}, \{d, e\}\}$ and remembers instead the “estimate” $\{\{d, e\}\}$.

	true	estimate
<i>support at time 0</i>	$\{\{a, b\}\}$	$\{\{a, b\}\}$
<i>support at time 1</i>	$\{\{a, c\}, \{b, c\}\}$	$\{\{a, c\}, \{b, c\}\}$
<i>support at time 2</i>	$\{\{a, d\}, \{b, d\}, \{c, d\}\}$	$\{\{a, d\}, \{b, d\}, \{c, d\}\}$
<i>support at time 3</i>	$\{\{a, e\}, \{b, e\}, \{c, e\}, \{d, e\}\}$	$\{\{d, e\}\}$
<i>support at time 4</i>	$\{\{a, e\}\}$	$\{\{a, d\}, \{a, e\}\}$
<i>support at time 5</i>	$\{\{a, e\}\}$	$\{\{a, e\}\}$

Continuing this process for the entire request sequence, we can maintain a competitive ratio of $\frac{3}{2}$, and never need more than two bookmarks. Why is this method valid? The answer is that if we have made a wrong guess, the one euro we did not demand, but which the adversary still owes, will be enough to pay for our error. Continuing the example, if the request sequence is $cdeae$, the optimal cost is 3. If we use PARTITION, our expected cost is 3.75. If we use forgiveness, our expected cost is 4.5, thus maintaining the desired competitive ratio of $\frac{3}{2}$.

³Note that the exchange rate of euros to dollars is actually less than $\frac{3}{2}$ at the time this column was written.

EQUITABLE uses this technique for arbitrary k . When the set of support configurations reaches a certain complexity, EQUITABLE deterministically sets its cache and then assumes that the support set consists of this single configuration, even though this assumption is probably wrong. EQUITABLE then erases all bookmarks. The invariant that EQUITABLE must maintain is that, if the minimum distance between the adversary’s true configuration and the nearest member of our estimated support is d , then the adversary owes us at least d euros. EQUITABLE needs $O(k^2 \log k)$ bookmarks, and is H_k -competitive [1, 2].

4 The Las Vegas Method

We have recently introduced a generalization of the concept of forgiveness, where our estimate is randomly chosen.⁴ Of course such a random choice cannot be completely arbitrary, and the mechanism by which such a choice is valid, is through a “bookie,” whom the algorithm has to pay whenever a random estimate is made. For the $k = 2$ case, our Las Vegas technique requires at most one bookmark, while still maintaining optimal competitiveness.

We illustrate the technique with an example. As before, let $\{a, b\}$ be the initial cache. Suppose the request sequence is $cdad$. The figure below shows the true support, and our estimate of the support. Unlike before, that estimate is randomly chosen at step 2. That is, at step 2, we pick one of the three singleton sets with equal probability and assume that the support consists of just that set, and set our cache to that configuration. Whichever choice we make, we forget the others.

To analyze the competitiveness of this Las Vegas approach, we invoke the concept of a *fair bookie*⁵, whom we alluded to a moment ago. The bookie allows us to place any wager, provided he knows that his expected return on the wager is non-negative. What we do at step 2 is to wager that the adversary is not at the configuration we choose to go to. We first pay the bookie one dollar. With probability $\frac{1}{3}$, we have made the correct guess, and the bookie pays us back nothing. With probability $\frac{2}{3}$, we have made the wrong guess, we win the wager, and the bookie pays us back \$1.50. (It is important to realize that the wager is actually *hedging*, like an insurance policy, *i.e.*, we win the wager only when we make a bad choice.) Note that the wager is fair regardless of how the adversary makes its choice. We note that the table assumes that the adversary chooses the configuration $\{a, d\}$, since it knows that the third request is a . But as far we are concerned, the third request could be b, c , or some new page. Continuing with the example, note that while servicing the given request sequence, we pay three dollars no matter what choice we make. The optimal cost is 2 euros, so the competitive ratio is maintained.

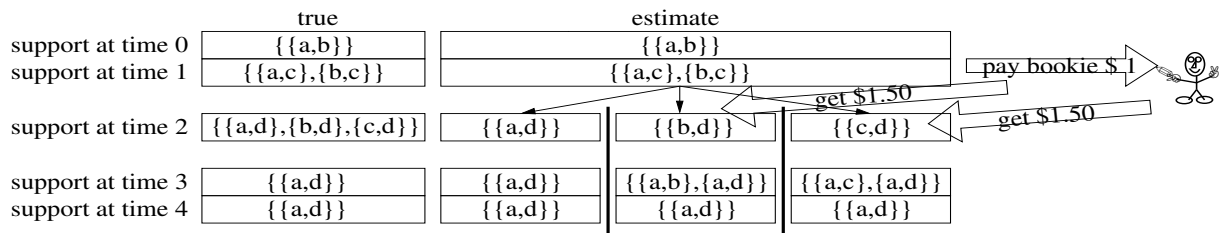


Figure 1: The Las Vegas Step

⁴This paper is to appear shortly: Bein, Larmore, Reischuk: Knowledge States Algorithms: Randomization with Limited Information.

⁵This column could only have been written in Las Vegas.

Although in the example, we eventually estimate the support to be the true support, that need not ever happen. Using Las Vegas analysis, we can prove that a competitiveness of $\frac{3}{2}$ can be maintained for any request sequence. We can view this as if we had made arrangements for the adversary and bookie to deposit payments (and the bookie sometimes to withdraw) to or from a blind account⁶ at the casino’s players’ bank, which we can draw from to pay for our bets and our moves as long as it has enough money, but where we can never query the balance.

In general, whenever our distribution is on two configurations and a new page is requested, we place the wager described above. If it so happens that none of the three configurations that we might choose is the adversary’s true configuration (which we cannot, in general know), the bookie discreetly refuses the bet, *i.e.*, returns our dollar to the blind account. No matter what the sequence of events, competitiveness of $\frac{3}{2}$ is maintained.

Knowledge States

The above randomized algorithm has two *knowledge states*, one where our estimate of the support is one configuration, which is our cache, the other where our distribution is uniform over the two configurations of our estimate. The knowledge state approach can be used to describe a randomized H_3 -competitive online algorithm for the paging problem where $k = 3$, using six knowledge states and no more than 2 bookmarks at any time. Can this approach be generalized to all k , so that $O(k)$ bookmarks are needed for an H_k -competitive algorithm?

5 Trackless Online Algorithms

We remind the reader that an online algorithm \mathcal{A} for the paging problem is trackless if, whenever there is a fault, \mathcal{A} cannot know whether the requested page has ever been requested before, or if so, when it was requested. A memoryless online algorithm is necessarily trackless, but not vice-versa, for example, the well-known deterministic online algorithm LRU (which ejects the **Least Recently Used** page in the cache if there is a fault) is trackless, and is (optimally) k -competitive for cache size k , but is not memoryless, as it must remember the ordering of its cache pages by recency of use.

Considering now randomization, we start with the simplest possible idea: in case of a fault, just eject a page at random, uniformly chosen among the k pages in the cache. We call this (memoryless) algorithm RAND. Its competitiveness is k for the k -cache problem, which is not very good.

RMARK, by Fiat, Karp, Luby, McGeoch, Sleator, and Young [14], is based on the same idea as RAND, namely eject a page at random, but is somewhat more intelligent, since it marks each page as it is used, and always ejects an unmarked page when there is a fault. Of course, eventually, all pages in the cache will become marked; when this happens, RMARK erases all the marks and starts over. It is relatively easy to prove that RMARK is $2H_k$ -competitive but its true competitiveness is $2H_k - 1$, as shown by Achlioptas, Chrobak, and Noga [1].

Lower and Upper Bounds

Little is known about the optimal competitiveness of a trackless randomized online algorithm for the paging problem, other than the obvious lower bound of H_k and the upper bound of $2H_k - 1$ given by RMARK. Even for $k = 2$, the optimal competitiveness is not known,

⁶Known conventionally as the *potential*.

but a lower bound of $\frac{37}{24} \approx 1.5416 > \frac{3}{2}$ has been proved [4], thus leading to the (almost certainly true) conjecture that, for any $k \geq 2$, no trackless H_k -competitive algorithm exists.⁷

An upper bound of $\frac{3+\sqrt{13}}{4} \approx 1.6154$ on this competitiveness of this problem is given by TL2, a clever finite-state algorithm by Chrobak, Koutsoupias, and Noga [7]. In TL2, each page in the cache is *unmarked*, *marked*, or *double marked*. If a is a page, we write a , \dot{a} , or \ddot{a} to indicate that a is unmarked, marked, or double marked. The most recently used page is always unmarked. If the cache is $\{a, b\}$ where a is most recently used, we denote the state of TL2 by the notation ab , $a\dot{b}$, or $a\ddot{b}$. If both pages are unmarked, it doesn't matter which was most recently used, *e.g.*, ab and ba denote the same state. If there is a request to an unmarked page, nothing happens. If the state is $a\dot{b}$ or $a\ddot{b}$ and the request is to b , the new state is ab . We now consider the case that the request is to a new page, say c . If the state is ab , then TL2 ejects a page chosen uniformly at random, and its new state is either $c\dot{a}$ or $c\dot{b}$. If the state is $a\dot{b}$, then TL2 ejects b and goes to state ca with probability $p = \frac{5-\sqrt{13}}{2} \approx 0.6972$, and otherwise ejects a and goes to state $c\ddot{b}$. If the state is $a\ddot{b}$, then TL2 ejects b and goes to state ca .

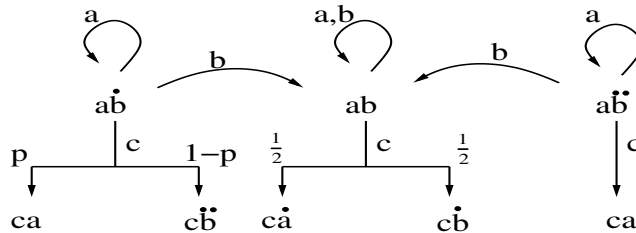


Figure 2: A Trackless Paging Algorithm for $k = 2$, with Competitiveness < 2

6 Related Issues for the Server Problem and Other Problems

We can consider the issue of memory limitation for other online problems. One online problem that has attracted a great deal of attention is the server problem, first proposed by Manasse, McGeoch and Sleator [18]. In this problem, there are k mobile identical servers in a metric space \mathcal{M} . At any time, a point $r \in \mathcal{M}$ can be “requested,” and must be “served” by moving one of the k servers to the point r . The cost of that service is defined to be the distance the server is moved. An online algorithm for the server problem decides, at each request, which server to move, but does not know the sequence of future requests. In fact, the paging problem is essentially a special case of the server problem, where the metric space is uniform, *i.e.* the distance between any two points in the metric space is 1.

We say that an online algorithm for the server problem is *trackless* if it is not permitted to know the distances between points unless both are currently “in play.” If s_1, \dots, s_k are the positions of our servers, and we are using a trackless online algorithm, we know $d(s_i, s_j)$ for all i, j , but cannot query distances between any other pairs of points. If r is a new request, we are told the distances $d(s_i, r)$ for each i , and nothing more. We must then choose some i , and move the server from s_i to r , at which point we lose “track” of s_i . This means that we cannot, in the future, query $d(s_i, x)$ where x is some future request point, but we can save the numerical values of distances $d(s_i, s_j)$ and $d(s_i, r)$ in our memory for future reference, just not the identities of the

⁷For $k = 2$, a lower bound of $\frac{3453}{2233} \approx 1.548736$ is indicated by a computer program, but has never been hand-verified.

points themselves. (This definition of tracklessness for the server problem is consistent with the definition of tracklessness for the paging problem and its standard reduction to the server problem in a uniform metric space.)

The Irani-Rubinfeld algorithm BALANCE [15] is trackless, and is known to be 10-competitive for $k = 2$. An improvement, BALANCE_SLACK, defined only for $k = 2$, has competitiveness 4, the lowest known for any trackless deterministic algorithm for the 2-server problem [8]. HARMONIC is a memoryless and trackless competitive randomized online algorithm for the k -server problem. Given servers located at points s_1, \dots, s_k and a request at a point r , the probability that the server at s_i serves the request is inversely proportional to the distance from s_i to r . The competitiveness of HARMONIC is at least $\binom{k+1}{2}$ [19], and it is conjectured that that is the upper bound as well. For $k = 2$, the competitiveness of 3 is settled [9, 11]. For $k \geq 3$, Bartal and Grove’s result that $C = O(2^k \log k)$ is so far the best upper bound known [3].

We also mention that recently Epstein *et al.* [13] have considered a generalized 2-server problem on a uniform space in which servers have different cost. For ratios below 2.2, they present an optimal algorithm which is trackless, and they give lower bounds which depend on the cost ratio.

There are at least two “notorious,” (in the words of some researchers) open problems in the area of server research. The first one is the k -server conjecture [18], which states that there is a k -competitive deterministic online algorithm for the k -server problem. WFA (the **W**ork **F**unction **A**lgorithm) has the lowest competitiveness known for a deterministic online algorithm, being $(2k - 1)$ -competitive in general [17], and 2-competitive for $k = 2$. It requires keeping track of (in the worst case) all previous requests, hence is as far from being trackless as it could be. There is no memoryless competitive deterministic online algorithm for the k -server problem, for any $k \geq 2$, and it is known that no deterministic trackless online algorithm for the 2-server problem can be 2-competitive for all metric spaces. A lower bound of $\frac{23}{11} \approx 2.09$ is proved in [5].

The second notorious problem has to do with the randomized 2-server problem. Despite much effort, no randomized algorithm for general metric spaces with competitiveness strictly lower than 2 has been found, yet the best known lower bounds are substantially less than 2. In fact, $1 + e^{-\frac{1}{2}} \approx 1.6065$, is the greatest lower bound with a published proof (see [10]) on the competitiveness of any randomized online algorithm.⁸ Breaking through this “2-competitive barrier” was chosen by the participants of the 2002 Dagstuhl Workshop on Online Algorithms to be one of the three most important outstanding online problems. In short, this problem has stymied researchers for many years – and the randomized online algorithm with best known competitive ratio for arbitrary metric spaces is the very simple (in fact, trackless and memoryless) algorithm RANDOM_SLACK [12], now known for over a decade.

Knowledge State Algorithms for the Server Problem

What is interesting is that RANDOM_SLACK can, in fact, be viewed as a simple knowledge state algorithm for the 2-server problem. A k -DIP algorithm is defined to be a knowledge state algorithm for the randomized 2-server problem in a particular metric space M where each knowledge state is supported by at most k configurations. RANDOM_SLACK is a 1-DIP algorithm where the distribution is always concentrated at a single configuration and the estimate of the support contains just that one configuration. It is important to note that a 2-DIP approach yields optimal competitiveness, *i.e.*, $\frac{3}{2}$, for the uniform case, described earlier.

⁸A lower bound very slightly larger than $1 + e^{-\frac{1}{2}}$ is given in [10], but without proof.

We conjecture that the knowledge state approach will yield the sought-after breakthrough, an algorithm which is less than 2 competitive for all metric spaces. At the Dagstuhl workshop, we presented a preliminary result, which many colleagues viewed as an important step: we used optimization techniques to construct a non-trivial knowledge state algorithm for the 2-server problem on the line with competitive ratio $\frac{71}{36} \approx 1.972$, where the distribution is never defined on more than two configurations. A hand proof derived from that program seems to be quite daunting, and may eventually be published, but may not, since it appears not to be the best result that can be obtained by that method.

Acknowledgements

The authors of this column acknowledge generous support from the National Science Foundation for their work summarized here. The earlier work on trackless algorithms was supported by NSF grant CCR-9821009. The current work on knowledge state algorithms is supported by NSF Information Technology Research grant CCR-0132093.

References

- [1] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. In *Proc. 4th European Symp. on Algorithms (ESA)*, volume 1136 of *Lecture Notes in Computer Science*, pages 419–430. Springer, 1996.
- [2] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234:203–218, 2000.
- [3] Yair Bartal and Edward Grove. The harmonic k -server algorithm is competitive. *Journal of the ACM*, 47(1):1–15, 2000.
- [4] Wolfgang Bein, Rudolph Fleischer, and Lawrence L. Larmore. Limited bookmark randomized online algorithms for the paging problem. *Information Processing Letters*, 76:155–162, 2000.
- [5] Wolfgang Bein and Lawrence L. Larmore. Trackless online algorithms for the server problem. *Information Processing Letters*, 74:73–79, 2000.
- [6] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- [7] Marek Chrobak, Elias Koutsoupias, and John Noga. More on randomized on-line algorithms for caching. *Theoretical Computer Science*, 290:1997–2008, 2003.
- [8] Marek Chrobak and Lawrence L. Larmore. On fast algorithms for two servers. *Journal of Algorithms*, 12:607–614, 1991.
- [9] Marek Chrobak and Lawrence L. Larmore. HARMONIC is three-competitive for two servers. *Theoretical Computer Science*, 98:339–346, 1992.
- [10] Marek Chrobak, Lawrence L. Larmore, Carsten Lund, and Nick Reingold. A better lower bound on the competitive ratio of the randomized 2-server problem. *Information Processing Letters*, 63:79–83, 1997.

- [11] Marek Chrobak and Jiří Sgall. A simple analysis of the harmonic algorithm for two servers. *Information Processing Letters*, 75:75–77, 2000.
- [12] Don Coppersmith, Peter G. Doyle, Prabhakar Raghavan, and Marc Snir. Random walks on weighted graphs and applications to on-line algorithms. *Journal of the ACM*, 40:421–453, 1993.
- [13] Leah Epstein, Csánad Imreh, and Rob van Stee. More on weighted servers or FIFO is better than LRU. In *Proc. 27th Symp. on Mathematical Foundations of Computer Science (MFCS)*, volume 2420 of *Lecture Notes in Computer Science*, pages 257–268. Springer, 2002.
- [14] Amos Fiat, Richard Karp, Michael Luby, Lyle A. McGeoch, Daniel Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.
- [15] Sandy Irani and R. Rubinfeld. A competitive 2-server algorithm. *Information Processing Letters*, 39:85–91, 1991.
- [16] Elias Koutsoupias and Christos Papadimitriou. Beyond competitive analysis. In *Proc. 35th Symp. Foundations of Computer Science (FOCS)*, pages 394–400. IEEE, 1994.
- [17] Elias Koutsoupias and Christos Papadimitriou. On the k -server conjecture. *Journal of the ACM*, 42:971–983, 1995.
- [18] Mark Manasse, Lyle A. McGeoch, and Daniel Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.
- [19] Prabhakar Raghavan and Marc Snir. Memory versus randomization in on-line algorithms. *IBM Journal on Research and Development*, 38, 1994.