

Knowledge States for the Caching Problem in Shared Memory Multiprocessor Systems

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

Rüdiger Reischuk
Institut für Theoretische Informatik
Universität zu Lübeck, Germany
reischuk@tcs.uni-luebeck.de

Abstract

Multiprocessor systems with a global shared memory provide logically uniform data access. To hide latencies when accessing global memory each processor makes use of a private cache. Several copies of a data item may exist concurrently in the system. To guarantee consistency when updating an item a processor must invalidate copies of the item in other private caches. To exclude the effect of classical paging faults, one assumes that each processor knows its own data access sequence, but does not know the sequence of future invalidations requested by other processors. Performance of a processor with this restriction can be measured against the optimal behavior of a theoretical omniscient processor, using competitive analysis. A $\frac{4}{3}$ -competitive randomized online algorithm for this problem for cache size 2 is presented. This algorithm is derived with the help of a new concept we call knowledge states. We also prove a matching lower bound, thus this online algorithm is best possible. Finally, a lower bound of $\frac{3}{2}$ on the competitiveness for larger cache sizes is shown.

1. Introduction

High performance computing, even in large parallel systems with a shared global memory, still requires that each processor has an individual local cache and uses this limited memory efficiently. In the following, let k denote the size of this cache. Each processor P manages its cache itself, but whenever another processor P' has written to a page that has a copy in the cache of P , this copy has to be invalidated (and effectively ejected from the cache of P .) Therefore, in a multiprocessor system, there can be two reasons for a page miss: Either, P can experience a “miss” in the usual way, *i.e.*, because a data item needed by the processor was not among any of its k cached pages, or the page has been

in the cache, but an invalidation caused by some other processor has occurred before P accessing this page. This we call an *invalidation miss*.

This leads to an optimization model with a tantalizing information restriction: Given the sequence R of requested pages for processor P one considers R to be known in its entirety to P 's caching strategy; only invalidations are given online. We call this the *OFI* (Oblivious to Future Invalidations) restriction. Therefore, for the competitive ratio of this problem, P 's strategy knows R but not the invalidations. We measure the cost of the strategy against the cost of an optimal offline algorithm that knows both the entire sequence as well as all invalidations in advance. Other models which involve partial information about the future include *lookahead* for the paging problem [1], [2], [6], [11], and competitive implementation of parallel programs [8].

Genther and Reischuk [9, 10] have recently studied different paging strategies in such an environment, primarily focusing on *realistic* request sequences to analyze the average case behavior by simulation. Extending this research along more theoretical lines, they ask how much the effect of invalidations might increase page misses in the worst case, as compared with the uniprocessor situation. In [10] bounds on the competitive ratio for certain deterministic strategies are established. However, the randomized case has remained largely open.

In this paper, we use a novel technique, the *knowledge state approach*, to derive results in the randomized case. We will briefly describe this technique in Section 2. In Section 3, we give a knowledge state algorithm with a competitive ratio of $\frac{4}{3}$ for $k = 2$. Section 4 and 5 give lower bounds. For $k = 2$ we show that the ratio $\frac{4}{3}$ is best possible, thus establishing that the knowledge state algorithm is optimally competitive. For $k \geq 3$, we show a lower bound of $\frac{3}{2}$. We conjecture that for $k = 3$ there exist knowledge state algorithms that match this lower bound.

*Research of these authors supported by NSF grant CCR-0132093.

2. Knowledge States

Competitiveness makes sense as a concept when an algorithm lacks timely access to all input data. If \mathcal{A} is an algorithm for a given problem which attempts to minimize cost for a given input instance, we say that \mathcal{A} is ρ -competitive if there is some constant λ such that, for any instance, the cost paid by \mathcal{A} does not exceed λ plus ρ times the optimal cost for that instance. Central to our discussion is a novel technique in competitive analysis: the knowledge state approach. We summarize that technique briefly in this section. We refer the reader to [3] for a detailed description of the technique.

One tool that is useful in the analysis of online algorithms, and which plays a role in our problem, is the concept of a *work function*. Work functions provide information about the optimal cost of serving the past request sequence. For a request sequence R , by $\omega^R(x)$, we denote the minimum cost of serving R and ending in configuration x . A *configuration* is simply an unordered k -tuple of pages, and represents a possible cache configuration. We denote the set of all configurations as \mathcal{X} . (To be more precise, at the beginning or after several invalidation misses the cache may hold fewer than k pages, but it suffices to consider only complete configurations.) The optimal cost to service a request sequence R is given by $\text{opt}(R) := \min_x \omega^R(x)$. It can be computed by dynamic programming. As time increases, the work function grows without bound. It is convenient to consider $\omega^R - \text{opt}(R) : \mathcal{X} \rightarrow \mathbb{N}$, which is called the *offset function*, and which is non-negative.

We introduce a convenient notation, taken from [12], for offset functions for the k -cache problem, which we call the *bar notation*. Let w be a string consisting of at least k page names and exactly k bars, with the condition that at least i page names are to the left of the i^{th} bar. Then w defines an offset function ω as follows: If x is any configuration of pages such that, for each $i = 1, \dots, k$, the names of at least i members of x are written to the left of the i^{th} bar, then $\omega(x) = 0$. We call configurations which satisfy this condition the *support set* of ω . If $y \in \mathcal{X}$ does not satisfy this condition, then $\omega(y)$ is the number of page replacements necessary to change y to a configuration which is a member of the support set. For example, if $k = 2$, $ab||$ denotes the offset function whose support set consists of just the configuration $\{a, b\}$, while if $k = 4$, $ab||cd|ef|$ denotes the offset function whose support consists of the configurations $\{a, b, c, d\}$, $\{a, b, c, e\}$, $\{a, b, c, f\}$, $\{a, b, d, e\}$, and $\{a, b, d, f\}$. From [12], we have:

Lemma 1 *A function ω is an offset function for the k -cache problem if and only if it can be expressed using the bar notation.*

We will use a variation of the *distribution model* to de-

scribe a randomized algorithm. That is, at each step the state of the algorithm will be described by a probability distribution on the set of all possible [deterministic] states at that step. The distribution model is equivalent to the behavioral model for randomized online algorithms against an oblivious adversary. (See, for example, [5].) In the standard distribution model, the algorithm deterministically chooses a distribution at each step, but in this paper we allow the algorithm to use randomization to choose the distribution. This variation, called the *mixed model* of randomized algorithms is a generalization of both the behavioral model and the distributional model. A *knowledge state algorithm* [3, 4] is a mixed online algorithm that computes an *adjustment* and an *estimator* at each step, and uses the current estimator as its memory state. The estimator is a real-valued function on configurations that is updated at every step, and which estimates the cost of the optimal offline algorithm, while the adjustment is a real number that is computed at every step. More formally, if \mathcal{A} is a knowledge-state algorithm, then:

1. At any given step, the full state of \mathcal{A} is a pair (ω, π) , where π is a finite distribution on \mathcal{X} , and $\omega : \mathcal{X} \rightarrow \mathbb{R}$ is the current estimator. We call that pair the *current knowledge state*.
2. If $S = (\omega, \pi)$ is the knowledge state and the next request is r , then \mathcal{A} computes an adjustment, a number which we call $\text{adjust}_{\mathcal{A}}(S, r)$, and uses randomization to pick a new knowledge state $S' = (\omega', \pi')$. More precisely, there are subsequent knowledge states $S_i = (\omega_i, \pi_i)$ and subsequent positive weights λ_i for $i = 1, \dots, m$, $\sum_{i=1}^m \lambda_i = 1$, such that
 - (a) $(\omega \wedge r)(x) \geq \text{adjust}_{\mathcal{A}}(S, r) + \sum_{i=1}^m \lambda_i \omega_i(x)$ for each $x \in \mathcal{X}$, where we define function $\omega \wedge r$ as $(\omega \wedge r)(y) = \min_{x \in \mathcal{X}} \{\omega(x) + \text{cost}(x, r, y)\}$ and $\text{cost}(x, r, y)$ denotes the cost of serving r given cache configuration x while resulting in configuration y .
 - (b) For each i , \mathcal{A} chooses S' to be S_i with probability λ_i .

In our application, we use offset functions as estimators. We will also associate a finite amount of information about the future private request sequence with each offset function ω .¹ When there is a request, the algorithm computes a new offset function and selects a new distribution, but may also need to “query the future” to decide the new knowledge state. We note that the algorithm does not obtain new information in this query. The algorithm has the entire private request sequence in its memory (or otherwise available), and “querying the future” means simply looking at this memory.

¹In this sense our application requires a generalized model: Our knowledge states may include an extra piece of information, and we note that our method is also valid in this case (see [4].)

3. A $\frac{4}{3}$ -Competitive Algorithm for $k = 2$

Without loss of generality, each invalidation of a page a takes place immediately before a is requested since this is the worst case scenario for OFI-page faults in multiprocessor systems. The request of a preceded by an invalidation is denoted by \hat{a} , while a request of a with no invalidation before we simply write as a . For example, if $\hat{R} = a\hat{b}c\hat{b}a\hat{b}a$ is the sequence of future requests, then only the sequence of its private future requests, $R = abcba\hat{b}a$, is known to the OFI-online-algorithm. In the first step serving a , the algorithm must decide on evicting a page without knowing whether the next request is of the form b or \hat{b} . If it is \hat{b} it would not make sense to keep b . Thus if we have to make room for a , and b is in the cache at the moment an optimal offline algorithm knowing that \hat{b} comes next will evict b . We require that the request sequence \hat{R} must be *consistent* with the private request sequence R , meaning that R can be obtained from \hat{R} by replacing every \hat{a} by a , for any page a .

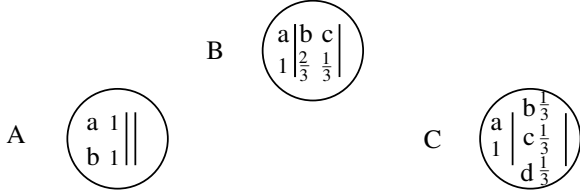


Figure 1. The Three Knowledge States A, B, C

Let us now define the four parameterized knowledge states, which we name A_{ab} , B_{abc} , C_{abcd} , and D_{abcde} , where a, b, c, d, e are arbitrary pages. A_{ab} , B_{abc} , and C_{abcd} are illustrated in Figure 1. The knowledge state D_{abcde} , which is only used as a transitory state, appears in Figure 4.

If a and b are pages, the notation “ $a < b$ ” means that, if a and b both appear in the future private request sequence (which is known to the algorithm), then the next instance of a precedes the next instance of b , or that b never appears in the future.

1. $A_{ab} = (a|b|, \pi_{ab}^A)$, where π_{ab}^A is defined to be 1 on the pair $\{a, b\}$. The knowledge state A_{ab} is the same as the knowledge state A_{ba} .

Our full notation for the knowledge state A_{ab} is $\begin{array}{c} a \\ b \end{array} \begin{array}{c} 1 \\ 1 \end{array} \Big|$, where we write the page names vertically instead of horizontally to indicate that we do not care whether $a < b$.

2. $B_{abc} = (a|bc| \text{ and } b < c, \pi_{abc}^B)$, where π_{abc}^B is defined to be $\frac{2}{3}$ on $\{a, b\}$ and $\frac{1}{3}$ on $\{a, c\}$. Besides the offset function $a|bc|$, the knowledge state contains the information that $b < c$.

Thus, $B_{abc} \neq B_{acb}$, despite the fact that they have the same offset function.

Our full notation for this knowledge state is $B_{abc} = \begin{array}{c} a \\ 1 \end{array} \begin{array}{c} b \\ \frac{2}{3} \\ c \\ \frac{1}{3} \end{array} \Big|$, where the placing of c horizontally after b indicates that $b < c$.

3. $C_{abcd} = (a|bcd|, \pi_{abcd}^C)$, where π_{abcd}^C is $\frac{1}{3}$ on each of the three states, $\{a, b\}$, $\{a, c\}$, and $\{a, d\}$. Note that $C_{abcd} = C_{abdc} = C_{acbd} = C_{acdb} = C_{adcb} = C_{adbc}$, since this knowledge state contains no information about the future order of requests.

The full notation for this knowledge state is $C_{abcd} = \begin{array}{c} a \\ 1 \end{array} \begin{array}{c} b \\ \frac{1}{3} \\ c \\ \frac{1}{3} \\ d \\ \frac{1}{3} \end{array} \Big|$

4. $D_{abcde} = (a|bcde|, \pi_{abcde}^D)$, where π_{abcde}^D is $\frac{1}{4}$ on each of the states $\{a, b\}$, $\{a, c\}$, $\{a, d\}$, and $\{a, e\}$.

The full notation for this knowledge state is $D_{abcde} = \begin{array}{c} a \\ 1 \end{array} \begin{array}{c} b \\ \frac{1}{4} \\ c \\ \frac{1}{4} \\ d \\ \frac{1}{4} \\ e \\ \frac{1}{4} \end{array} \Big|$

The knowledge state D is transitory, meaning that when our algorithm enters it it immediately leaves it, as shown in Figure 4.

Theorem 1 For cache size 2, there exists an online algorithm \mathcal{A} achieving a competitiveness bounded by $\frac{4}{3}$.

Proof: We will use a standard potential argument to prove competitiveness, and thus we will need to associate a potential Φ with each knowledge state. We now define the *update condition* for a given step. Fix $\rho > 1$. Let S^{t-1} be the knowledge state after $t - 1$ steps, let $\{U_i\}$ be the subsequents for step t , and λ_i be the probability that U_i will be chosen to be S^t . We define *adjust* to be the expected adjustment of this step, the minimum value of the difference between the updated work function and the expected work function after the Las Vegas step, and $cost_{\mathcal{A}}$ to be the expected cost of the algorithm \mathcal{A} . Then the update condition is that

$$\Phi(S^{t-1}) \geq cost_{\mathcal{A}} - \rho \cdot adjust + \sum_i \lambda_i \Phi(U_i).$$

We will make use of the following lemma from [4]:

Lemma 2 If the update condition holds at every step of an online algorithm then it is ρ -competitive.

Fix now $\rho = \frac{4}{3}$. We specify the ρ -competitive algorithm \mathcal{A} , simultaneously verifying the update condition at each

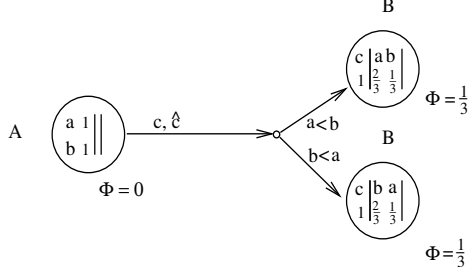


Figure 2. Transition from Knowledge State A

step. \mathcal{A} uses the 4 knowledge states defined above with potentials

$$\Phi(A_{ab}) = 0, \quad \Phi(B_{abc}) = \frac{1}{3}, \quad \Phi(C_{abcd}) = \frac{2}{3}, \quad \phi(D_{abcde}) = 1.$$

The actions of \mathcal{A} in each knowledge state are as follows.

1. If the knowledge state is A_{ab} , B_{abc} , or C_{abcd} , and the request a nothing happens (not illustrated).
2. If the knowledge state is A_{ab} , B_{abc} , or C_{abcd} , and the request \hat{a} , the algorithm ejects a and later moves (the updated version of) a back into the cache. It holds that $cost_{\mathcal{A}} = adjust = 1$, and the knowledge state remains the same (not illustrated).
3. If the knowledge state is A_{ab} and the request c or \hat{c} , \mathcal{A} considers its private request sequence. If $a < b$ it moves to the knowledge state B_{cab} , else to B_{cba} . In both cases, $cost_{\mathcal{A}} = adjust = 1$, and Φ increases by $\frac{1}{3}$, thus the update condition is exactly satisfied (see Figure 2).
4. If the knowledge state is B_{abc} and the request is b , the new knowledge state is A_{ab} . Then $cost_{\mathcal{A}} = \frac{1}{3}$, since the probability is $\frac{2}{3}$ that our cache state is already ab , and $adjust = 0$. The potential decreases by $\frac{1}{3}$ (see Figure 3).
5. If the knowledge state is B_{abc} and the request is \hat{b} , \mathcal{A} considers its private request sequence. If $a < c$ it moves to the knowledge state B_{bac} . Then $cost_{\mathcal{A}} = adjust = 1$, and the potential remains the same. Otherwise, the algorithm moves to the knowledge state B_{bca} . Then $cost_{\mathcal{A}} = \frac{4}{3}$, $adjust = 1$ without changing the potential (see Figure 3). In either case, the update condition is satisfied.
Note that in B_{abc} , a request of c or \hat{c} is impossible since $b < c$.
6. If the knowledge state is B_{abc} and the request is a new page d or \hat{d} , the algorithm moves to C_{dabc} . Then $cost_{\mathcal{A}} = adjust = 1$, and Φ increases by $\frac{1}{3}$, thus the update condition is exactly satisfied (see Figure 3).

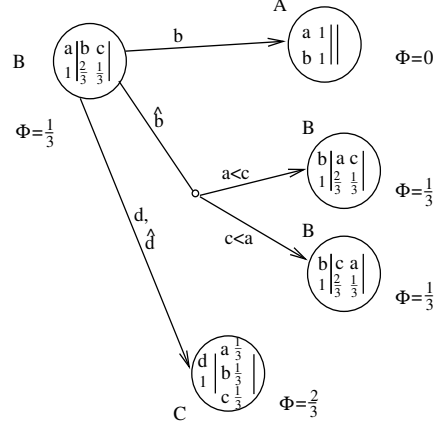


Figure 3. Transition from Knowledge State B

7. If the knowledge state is C_{abcd} and the request is either b , c , d , or \hat{b} , \hat{c} , or \hat{d} , without loss of generality, it is either b or \hat{b} , since no special distinguishing information about b , c , d has been recorded in the knowledge state. In the case of b , \mathcal{A} moves to knowledge state A_{ab} . Then $cost_{\mathcal{A}} = \frac{2}{3}$, $adjust = 0$, and the update condition is exactly satisfied (see Figure 4). If the request is \hat{b} , \mathcal{A} moves to knowledge state C_{bacd} . Then $cost_{\mathcal{A}} = adjust = 1$, the potential remaining the same (see Figure 4).
8. If the knowledge state is C_{abcd} and the request is e or \hat{e} , where e is a new page, the algorithm first moves to the transitory knowledge state D_{eabcd} . For this part, $cost_{\mathcal{A}} = adjust = 1$, and Φ increases by $\frac{1}{3}$. Then the algorithm moves to one of the four knowledge states A_{ae} , A_{be} , A_{ce} , or A_{de} , choosing each with probability $\frac{1}{4}$. For this part, by Lemma 3 below, $cost_{\mathcal{A}} = 0$, while, by Lemma 4 below, $adjust = -\frac{3}{4}$, and the potential decreases by 1. Thus, the update condition is satisfied exactly (see Figure 4).

Lemma 3 In Case 8, the cost of algorithm \mathcal{A} is 0.

Proof: Although in the behavioral model randomization must be used to choose one of the A 's, in the distributional model the choice is deterministic. If the algorithm's state is ab it chooses the knowledge state A_{ab} . If the algorithm's state is ac it chooses the knowledge state A_{ac} . If the algorithm's state is ad it chooses the knowledge state A_{ad} . If the algorithm's state is ae it chooses the knowledge state A_{ae} . Thus, the movement cost is 0. \square

Lemma 4 In Case 8, the adjustment is $-\frac{3}{4}$.

Proof: The offset function before the Las Vegas step is $a|bcde|$. The offset function after the Las Vegas step is either $ab||$, $ac||$, $ad||$, or $ae||$, each with probability $\frac{1}{4}$. The

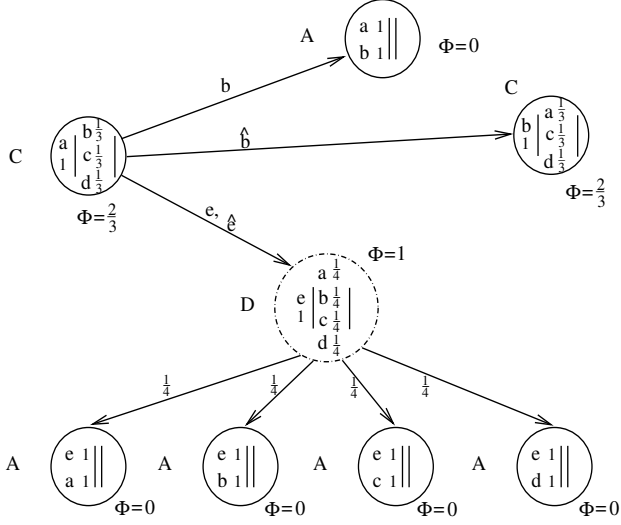


Figure 4. Transition from Knowledge State C

adjustment is thus, by definition, the maximum value of the function $a|bcde| - \frac{1}{4}ab| - \frac{1}{4}ac| - \frac{1}{4}ad| - \frac{1}{4}ae|$. This function has the same value $-\frac{3}{4}$ on each of the four pairs. \square

Note that in every case in the above list, the cost of \mathcal{A} plus the increase in potential is less than or equal to $\frac{4}{3}$ times the adjustment. This completes the proof of Theorem 1. \square

We have only given a distributional description of algorithm \mathcal{A} . One has to translate it into a behavioral description in order to implement the algorithm. That this can be done one has to check that the transitions between the different distributions of the knowledge states are indeed possible. We omit these simple calculations here.

4. A Lower Bound for Cache Size 2

Theorem 2 *For cache size 2, the competitiveness of any randomized online algorithm for the cache coherent data access problem is at least $\frac{4}{3}$.*

Proof: Let us consider a scenario where there are just three pages altogether, which we call a , b , and c . Let \mathcal{A} be any randomized online algorithm. We give a randomized adversary which forces \mathcal{A} 's expected cost to be at least $\frac{4}{3}$ the optimal cost.

Our adversary picks the private request sequence $R = (abc)^n$ for some large n , and then chooses a request sequence \hat{R} consistent with R using randomization, as follows. The adversary first picks a random string Γ of sufficient length (roughly $\frac{3n}{2}$ will be sufficient) over the alphabet $\{2, 3\}$. Each symbol of Γ is picked independently and uniformly. Γ is then used as a guide to pick \hat{R} .

The sequence \hat{R} will consist of *phases*, where each phase has length either 2 or 3 determined by the i^{th} symbol of Γ .

The last request in each phase is a simple page load (that is of type a), all other requests are page loads with an invalidation of that page before (that means of type \hat{a}). For example, if $\Gamma = 2332322$, then $\hat{R} = \hat{a}b, \hat{c}\hat{a}b, \hat{c}\hat{a}b, \hat{c}a, \hat{b}\hat{c}a, \hat{b}c, \hat{a}b$, where the commas separate the phases.

Our particular setting implies that an optimal offline algorithm can always satisfy the page load without a cache miss. Thus, depending on the length of a phase it makes either 1 or 2 pages faults. The expected number of page faults then equals $\frac{3n}{2}$.

By symmetry, let \hat{a} be the first request of a phase. Then the online algorithm \mathcal{A} will be in state ab or state ac after serving it. If \mathcal{A} is in state ac then its cost of serving the whole phase is at least 2 since the next request will generate a page fault for sure. If \mathcal{A} is in state ab its cost of serving the phase is 1 if the phase has length 2, and 3 if the phase is $\hat{a}\hat{b}c$. Thus, the expected cost of servicing one phase is at least 2. This yields a lower bound $\frac{4}{3}$ on the competitiveness of \mathcal{A} . \square

Combining Theorems 1 and 2, we obtain:

Theorem 3 *For cache size 2, the randomized competitiveness of the cache coherent data access problem is $\frac{4}{3}$.*

5. A Lower Bound for Larger Cache Sizes

Using a larger regular request sequence, but a significantly more complicated analysis, we can also established a lower bound of $\frac{3}{2}$ for larger cache sizes.

Theorem 4 *The competitiveness of any randomized OFI-online algorithm for the cache coherent data access problem for cache size at least 3 cannot be less than $\frac{3}{2}$.*

Proof: We sketch the main idea for $k = 3$. The detailed analysis requires a longer exposition that has to be skipped due to the space restriction.

It suffices to use four pages, a , b , c , and d and the private request sequence be $(abcd)^n$ for sufficiently large n . Let \mathcal{A} be a ρ -competitive randomized online algorithm in the distribution model whose probability distribution is known to the adversary, but of course not the outcomes of its future probabilistic steps. Then, there must exist a potential function $\Phi(\omega, \pi)$ defined for every work function ω and every distribution π that satisfies the update condition with competitiveness ρ . We define certain quantities and relate them by inequalities deduced from specific requests by the adversary in special situations to establish a lower bound on ρ . Each quantity is a formula involving the potential, minimized over all choices of distributions π for \mathcal{A} . Let $\hat{\pi}(a)$ be the probability that a page a is not in the cache and define

$$\alpha = \min_{\pi} \{ \Phi(a|bcd|, \pi) - \hat{\pi}(b) \}$$

$$\begin{aligned}\beta &= \min_{\pi} \{ \Phi(ab|cd|, \pi) - \hat{\pi}(d) \} \\ \gamma &= \min_{\pi} \{ \Phi(ab|cd|, \pi) + \hat{\pi}(d) \} \\ \delta &= \min_{\pi} \{ \Phi(abc||, \pi) - \hat{\pi}(a) \} \\ \epsilon &= \min_{\pi} \{ \Phi(ab|cd|, \pi) \}\end{aligned}$$

By symmetry, the names of the pages can be rotated $a \rightarrow b \rightarrow c \rightarrow d \rightarrow a$, thus, for example, $\alpha = \min_{\pi} \{ \Phi_{\pi}^{b|cda|} - \hat{\pi}(c) \}$. We can then prove the following five inequalities:

$$2\epsilon \geq \gamma + \beta, \quad \gamma \geq \delta + 1, \quad \alpha \geq \epsilon, \quad \delta \geq \alpha + 1 - \rho, \quad \beta \geq \alpha + 1 - \rho.$$

Combining the five inequalities yields $\rho \geq \frac{3}{2}$. The details are given in the full paper. \square

6. Conclusions

We mention that a forgiveness online algorithm is a knowledge state algorithm with the special restriction that there is always exactly one subsequent. It is interesting to note that historically, forgiveness came first, so we can think of the knowledge state approach as being a generalization of forgiveness.

We conjecture that a knowledge state algorithm with randomized competitiveness $\frac{3}{2}$ exists for cache size 3. Since our randomized results show better competitiveness than the deterministic results of [9] and [10], our work suggests that randomization indeed could be beneficial in practice.

References

- [1] S. Albers. The influence of lookahead in competitive paging algorithms. In *Proc. 1st European Symp. on Algorithms*, LNCS 726, pages 1–12. Springer, 1993.
- [2] S. Albers. A competitive analysis of the list update problem with lookahead. In *Proc. 19th Symp. on Mathematical Foundations of Computer Science*, LNCS , pages 201–210, Springer Verlag, 1994.
- [3] W. Bein and L. Larmore. Trackless and Limited-Bookmark Algorithms for Paging. *ACM SIGACT News*, pages 38–48, 35(1), 2004.
- [4] W. Bein, L. Larmore and R. Reischuk. Knowledge State Algorithms: Randomization with Limited Information. *To appear*.
- [5] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- [6] D. Breslauer. On competitive on-line paging with lookahead. In *Proc. 13th Symp. on Theoretical Aspects of Computer Science*, LNCS 1046, pages 593–603, 1996.
- [7] M. Chrobak and L. Larmore. Metrical task systems, the server problem, and the work function algorithm. In *Online Algorithms: State of the Art*, pages 74–94. Springer-Verlag, 1998.
- [8] Xi. Deng, E. Koutsoupias, and P. MacKenzie. Competitive implementation of parallel programs. *Algorithmica*, 23, 1999.
- [9] K. Genther and R. Reischuk. Analyzing data access strategies in cache coherent architectures. Technical Report TR A-98-25, Universität zu Lübeck, Institut für Theoretische Informatik, 1999.
- [10] K. Genther and R. Reischuk. Analyzing the competitive ratio of caching in multiprocessor systems. Technical Report TR-A-01-19, 2001, Universität zu Lübeck, Institut für Theoretische Informatik, 2001.
- [11] E. Grove. Online binpacking with lookahead. In *Proc. 6th Symp. on Discrete Algorithms*, pages 430–436, 1995.
- [12] E. Koutsoupias and C. Papadimitriou. Beyond competitive analysis. In *Proc. 35th Symp. Foundations of Computer Science*, pages 394–400, 1994.