

Knowledge State Algorithms: Randomization with Limited Information

Wolfgang W. Bein ^{*} Lawrence L. Larmore [†] Rüdiger Reischuk [‡]

Abstract

We introduce the concept of knowledge states; many well-known algorithms can be viewed as knowledge state algorithms. The knowledge state approach can be used to construct competitive randomized online algorithms and study the tradeoff between competitiveness and memory. A knowledge state simply states conditional obligations of an adversary, by fixing a work function, and gives a distribution for the algorithm. When a knowledge state algorithm receives a request, it then calculates one or more “subsequent” knowledge states, together with a probability of transition to each. The algorithm then uses randomization to select one of those subsequents to be the new knowledge state. We apply the method to the paging problem. We present optimally competitive algorithm for paging for the cases where the cache sizes are $k = 2$ and $k = 3$. These algorithms use only a very limited number of bookmarks.

Keywords: Design of Algorithms; Online Algorithms; Randomized Algorithms, Paging.

1 Motivation and Background

In this paper we introduce a new method for constructing randomized online algorithms, which we call the *knowledge state* model. The purpose of this method is the address the trade-off between memory and competitiveness. The model is introduced and fully described for the first time in this publication, but we note that a number of published algorithms are implicitly consistent with the model although not in its full power. For example, the algorithm EQUITABLE [1] is a knowledge state algorithm for the k -cache problem that achieves the optimal randomized competitiveness of H_k for each k , using only $O(k^2 \log k)$ memory, as opposed to the prior algorithm, PARTITION [11], that uses the full information contained in the work function, and hence requires unlimited memory as the length of the request sequence grows. At the other end of the scale, the randomized algorithm RANDOM_SLACK [10] is in fact an extremely simple knowledge state algorithm, which achieves randomized 2-competitiveness for the 2-server problem for all metric spaces, and which achieves randomized k -competitiveness for the k -server problem on some spaces, including trees. We also note that RANDOM_SLACK is *trackless* and is an order 1 knowledge state algorithm, *i.e.*, its distribution is supported by only one state. (See the recent ACM SIGACT column [5] for a summary of tracklessness; see also [3, 4, 2, 6].) We also note that we have recently used the knowledge state technique to develop an optimally competitive algorithm for the caching problem in shared memory multiprocessor systems [6].

^{*}Department of Computer Science, Center for the Advanced Study of Algorithms, University of Nevada, Las Vegas, NV 89154. Email: bein@cs.unlv.edu. Research supported by NSF grant CCR-0312093.

[†]Department of Computer Science, Center for the Advanced Study of Algorithms, University of Nevada, Las Vegas, NV 89154. Email: larmore@cs.unlv.edu. Research supported by NSF grant CCR-0312093.

[‡]Institut für Theoretische Informatik, Universität Lübeck, Wallstraße 40, D-23560 Lübeck

It is still an open question, whether there exists an optimally competitive order $O(k)$ bookmark randomized algorithm for the k -cache problem. An affirmative answer to this question would settle an open problem listed in [7]. In this paper we describe progress on this question. We give an order 2 knowledge state algorithm which is provably H_2 -competitive. Since an equivalent behavioral algorithm must keep one “bookmark,” namely the address of an ejected page, it is not an improvement over our earlier result [3], but it does illustrate the knowledge state technique in a simple way. We then give an order 3 knowledge state algorithm which is provably H_3 -competitive, which is an improvement, in terms of memory requirements, over EQUITABLE for the case $k = 3$ (Section 4).

We also consider the problem of breaking the 2-competitive barrier for the randomized competitiveness of the 2-server problem, a goal which has, as yet, been achieved only in special cases (Section 5). For the class of uniform spaces, this barrier was broken by PARTITION [11]. For the line, a $\frac{155}{78}$ -competitive algorithm was given by Bartal *et al.* [2].

In this paper we give a formal description of the knowledge state method. It is defined using the mixed model of online computation, which is described in Section 2. This section relates the mixed model to the standard models of online computation, and explains how a behavioral algorithm can be derived from a mixed model description. Section 3 defines the knowledge state method (in terms of the mixed model) and shows how potentials can be used to derive the competitive ratio of a knowledge state algorithm. Even though the concepts in Section 2 and 3 are natural and intuitive some of the formal arguments to prove our method are somewhat involved. In Section 4 the method is applied to the paging problem; two optimally competitive algorithms are presented. We discuss ongoing experimental work for the server problem in Section 5.

2 The Mixed Model of Online Computation

We will introduce a new model of randomized online computation which is a generalization of both the classic behavioral and distributional models. We assume that we are given an online problem with states \mathcal{X} (also called configurations), a fixed *start state* $x^0 \in \mathcal{X}$, and a requests \mathcal{R} . If the current state is $x \in \mathcal{X}$ and a request $r \in \mathcal{R}$ is given, an algorithm for the problem must *service* the request by choosing a new state y and paying a cost, which we denote $cost(x, r, y)$. It is convenient to assume that there is a “distance” function d on \mathcal{X} , and it is possible to choose to move from state x to state y at cost $d(x, y)$ at any time, given no request. We will assume that $d(x, x) = 0$ and $d(x, z) \leq d(x, y) + d(y, z)$ for any states x, y, z . It follows that $cost(u, r, v) \leq d(u, x) + cost(x, r, y) + d(y, v)$ for any states u, x, y, v and request r . Formally in this paper we refer to an online problem as an ordered triple $\mathcal{P} = (\mathcal{X}, \mathcal{R}, d)$. Examples of online problems satisfying these conditions abound, such as the server problem, the cache problem, *etc.*

Given a *request sequence* $\varrho = r^1, \dots, r^n$, an algorithm must choose a sequence of states x^1, \dots, x^n , the *service*. The *cost* of this service is defined to be $\sum_{t=1}^n cost(x^{t-1}, r^t, x^t)$. An *offline* algorithm knows ϱ before choosing the service sequence, while an *online* algorithm must choose x^t without knowledge of the future requests. We will assume that there is an optimal offline algorithm, *opt*, which computes an optimal service sequence for any given request sequence. As is customary we say that a deterministic online algorithm \mathcal{A} is C -competitive for a given number C if there exists a constant K (not dependent on ϱ) such that $cost_{\mathcal{A}}(\varrho) \leq C \cdot cost_{opt}(\varrho) + K$ for any request sequence ϱ . Similarly, we say that a randomized online algorithm \mathcal{A} is C -competitive for a given number C if there exists a constant K (not dependent on ϱ) such that $E(cost_{\mathcal{A}}(\varrho)) \leq C \cdot cost_{opt}(\varrho) + K$ for any request sequence ϱ , where E denotes expected value.

In order to make the description of various models of randomized online computation more

precise, we introduce the following notation. Let Π be the set of all finite distributions on \mathcal{X} . If $\pi \in \Pi$ and $S \subseteq \mathcal{X}$, we say that S *supports* the distribution π if $\pi(S) = 1$. The *distributional support* (or “*support*” for short) of any $\pi \in \Pi$ is defined to be the unique minimal set which supports π . By an abuse of notation, if the support of π is a singleton $\{x\}$, we write $\pi = x$.

An instance of the *transportation problem* is a weighted directed bipartite graph with distributions on both parts. Formally, an instance is an ordered quintuple $(A, B, \text{cost}, \alpha, \beta)$ where A and B are finite non-empty sets, α is a distribution on A , β is a distribution on B , and cost is a real-valued function on $A \times B$. A *solution* to this instance is a distribution γ on $A \times B$ such that

1. $\gamma(\{a\} \times B) = \alpha(a)$ for all $a \in A$.
2. $\gamma(A \times \{b\}) = \beta(b)$ for all $b \in B$.

Then $\text{cost}(\gamma) = \sum_{a \in A} \sum_{b \in B} \gamma(a, b) \text{cost}(a, b)$, and γ is a *minimal* solution if $\text{cost}(\gamma)$ is minimized over all solutions, in which case we call $\text{cost}(\gamma)$ the *minimum transportation cost*.

There are three standard models of randomized online algorithms (see, for example [7]). We introduce a new model in this paper, which we call the *mixed model*. Those three standard models are: distribution of deterministic online algorithms, the behavioral model, and the distributional model. We very briefly describe the three standard models.

Distribution of Deterministic Online Algorithms. In this model, \mathcal{A} is a random variable whose value is a deterministic online algorithm. If the random variable has a finite distribution, we say that \mathcal{A} is *barely random*.

Behavioral Online Algorithms. In this model \mathcal{A} uses randomization at each step to pick the next configuration. We assume that \mathcal{A} has memory. Let \mathcal{M} be the set of all possible memory states of \mathcal{A} . We define a *full state* of \mathcal{A} to be an ordered pair $k = (x, m) \in \mathcal{X} \times \mathcal{M}$. Let $m^0 \in \mathcal{M}$ be the initial memory state, and let m^t be the memory state of \mathcal{A} after servicing the first t requests.

Then \mathcal{A} uses randomization to compute $k^t = (x^t, m^t)$, the full state after t steps, given only k^{t-1} and r^t . A behavioral algorithm can then be thought of as a function on $\mathcal{X} \times \mathcal{M} \times \mathcal{R}$ whose values are random variables in $\mathcal{X} \times \mathcal{M}$.

Distributional Online Algorithms. If $\pi, \pi' \in \Pi$, let S be the support of π and S' be the support of π' . We then define $d(\pi, \pi')$ to be the minimum transportation cost of the transportation problem (S, S', d, π, π') , and if $r \in \mathcal{R}$, we define $\text{cost}(\pi, r, \pi')$ to be the minimum transportation cost of the transportation problem $(S, S', \text{cost}^r, \pi, \pi')$, where $\text{cost}^r = \text{cost}(\cdot, r, \cdot) : \mathcal{X} \times \mathcal{X} \rightarrow \mathbf{R}$.

A distributional online algorithm \mathcal{A} is then defined as follows.

1. There is a set \mathcal{M} of memory states of \mathcal{A} . There is a start memory state $m^0 \in \mathcal{M}$.
2. A *full state* of \mathcal{A} is a pair $k = (\pi, m) \in \Pi \times \mathcal{M}$. The initial full state is $k^0 = (\pi^0, m^0)$, where $\pi^0 = s^0$.
3. For any given full state $k = (\pi, m)$ and request r , \mathcal{A} deterministically computes a new full state $k' = (\pi', m')$, using only the inputs π , m , and r . We write $\mathcal{A}(\pi, m, r) = (\pi', m')$ or alternatively $\mathcal{A}(k, r) = k'$. Thus, \mathcal{A} is a function from $\Pi \times \mathcal{M} \times \mathcal{R}$ to $\Pi \times \mathcal{M}$.
4. Given any input sequence $\varrho = r^1 \dots r^n$, \mathcal{A} computes a sequence of full states $\mathcal{A}(\varrho) = k^1, \dots, k^n$, following the rule that $k^t = (\pi^t, m^t) = \mathcal{A}(k^{t-1}, r^t)$ for all $t \geq 1$. Define $\text{cost}_{\mathcal{A}}(\varrho) = \sum_{t=1}^n \text{cost}(\pi^{t-1}, r^t, \pi^t)$.

We note that a distributional online algorithm, despite being a model for a randomized online algorithm, is in fact deterministic, in the sense that the full states $\{k^t\}$ are computed deterministically.

The following theorem is well-known. (It is, for example, implicit in Chapter 6 of [7].)

Theorem 1 *All three of the above models of randomized online algorithms are equivalent, in the following sense. If \mathcal{A}_1 is an algorithm of one of the models, there exist algorithms $\mathcal{A}_2, \mathcal{A}_3$, of each of the other models, such that, given any request sequence ϱ , the cost (or expected cost) of each \mathcal{A}_i for ϱ is no greater than the cost (or expected cost) of \mathcal{A}_1 .*

The Mixed Model. The *mixed model* of randomized algorithms is a generalization of both the behavioral model and the distributional model. A mixed online algorithm chooses a distribution at each step, but, as opposed to a distributional algorithm, which must make that choice deterministically, can use randomization to choose the distribution.

A *mixed* online algorithm \mathcal{A} for an online problem $\mathcal{P} = (\mathcal{X}, \mathcal{R}, d)$ is defined as follows. As before, let Π be the set of finite distributions on \mathcal{X} .

1. There is a set \mathcal{M} of memory states of \mathcal{A} . There is a start memory state $m^0 \in \mathcal{M}$.
2. A *full state* of \mathcal{A} is a pair $k = (\pi, m) \in \Pi \times \mathcal{M}$. The initial full state is $k^0 = (\pi^0, m^0)$, where $\pi^0 = s^0$.
3. For any given full state $k = (\pi, m)$ and request r , there exists a finite set of full states k_1, \dots, k_m and probabilities $\lambda_1 \dots \lambda_m$, where $\sum_{i=1}^m \lambda_i = 1$, such that if the current full state is k and the next request is r , \mathcal{A} uses randomization to compute a new full state $k' = (\pi', m')$, by selecting $k' = k_i$ for some i . The probability that \mathcal{A} selects each given k_i is λ_i . We call the $\{k_i\}$ the *subsequents* and the $\{\lambda_i\}$ the *weights* of the subsequents, for the request r from the full state k .

\mathcal{A} is a function on $\Pi \times \mathcal{M} \times \mathcal{R}$ whose values are random variables in $\Pi \times \mathcal{M}$. We can write $\mathcal{A}(\pi, m, r) = (\pi', m')$. Alternatively, we write $\mathcal{A}(k, r) = k'$. For fixed k and r ; k', π' , and m' can be regarded as random variables.

4. Given any input sequence $\varrho = r^1 \dots r^n$, \mathcal{A} computes a sequence of full states $\mathcal{A}(\varrho) = (\pi^1, m^1) \dots (\pi^n, m^n)$, following the rule that $k^t = (\pi^t, m^t) = \mathcal{A}(k^{t-1}, r^t)$ for all $t > 1$. Note that, for all $t > 0$, k^t , π^t , and m^t are random variables.

Computing the cost of a step of a mixed model online algorithm \mathcal{A} is somewhat tricky. We note that it might seem that $\sum_{i=1}^m \lambda_i \text{cost}(\pi, r, \pi_i)$ would be that cost; however, this is an overestimate.

Without loss of generality, \mathcal{A} is sensible. Let $k = (\pi, m) \in \Pi \times \mathcal{M}$ and let $r \in \mathcal{R}$. Let $S \subseteq \mathcal{X}$ be the support of π . Let $\{k_i = (\pi_i, m_i)\}$ be the subsequents and $\{\lambda_i\}$ the weights of the subsequents, for the request r from the full state k . Let $\tilde{S} \subseteq \mathcal{X}$ be the union of the supports of the $\{\pi_i\}$. Define $\bar{\pi} = \sum_{i=1}^m \lambda_i \pi_i$. Note that $\bar{\pi} \in \Pi$, and its support is \tilde{S} . Define $\text{cost}_{\mathcal{A}}(k, r) = \text{cost}(\pi, r, \bar{\pi})$.

Finally, if $\varrho = r^1 \dots r^n$ is the input request sequence, and the sequence of full states of \mathcal{A} is $k^1 \dots k^n$, we define $\text{cost}_{\mathcal{A}}(\varrho) = \sum_{t=1}^n \text{cost}_{\mathcal{A}}(k^{t-1}, r^t)$.

We now prove that the mixed model for randomized online algorithms is equivalent to the three standard models.

Lemma 1 *If \mathcal{A} is a mixed online algorithm, there is a behavioral online algorithm \mathcal{A}' such that, for any request sequence ϱ , $E(\text{cost}_{\mathcal{A}'}(\varrho)) = E(\text{cost}_{\mathcal{A}}(\varrho))$.*

Proof: A memory state of $\tilde{\mathcal{A}}$ will be a full state of \mathcal{A} , i.e., we could write $\tilde{\mathcal{M}} \subseteq \Pi \times \mathcal{M}$. By a slight abuse of notation, we also define a full state of $\tilde{\mathcal{A}}$ to be an ordered triple $(x, \pi, m) \in \mathcal{X} \times \Pi \times \mathcal{M}$

such that (π, m) is a full state of \mathcal{A} and $\pi(x) > 0$. Intuitively, $\tilde{\mathcal{A}}$ keeps track of its true state $x \in \mathcal{X}$, while remembering the full state (π, m) of an emulation of \mathcal{A} .

For clarity of the proof, we introduce more complex notation for some of the quantities defined earlier. Let $\pi, \sigma \in \Pi$, $m, n \in \mathcal{M}$, and $r \in \mathcal{R}$. If (π, m) is a full state of \mathcal{A} , define $\lambda_{\pi, m, r, \sigma, n}$ to be the probability that $\mathcal{A}(\pi, m, r) = (\sigma, n)$, i.e., the conditional probability that \mathcal{A} chooses (σ, n) to be the next full state, given that the current full state is (π, m) and the request is r . We assume that there can be at most finitely many choices of (σ, n) for which $\lambda_{\pi, m, r, \sigma, n} > 0$. In case (π, m) is not a full state of \mathcal{A} , then $\lambda_{\pi, m, r, \sigma, n}$ is defined to be zero. If (π, m) is a full state of \mathcal{A} and $r \in \mathcal{R}$, write $\bar{\pi}_{\pi, m, r} = \sum_{\sigma \in \Pi, n \in \mathcal{M}} \lambda_{\pi, m, r, \sigma, n} \cdot \sigma \in \Pi$, and choose a finite distribution $\gamma_{\pi, m, r}$ on $\mathcal{X} \times \mathcal{X}$ which is a minimal solution to the transportation problem $(\mathcal{X}, \mathcal{X}, \text{cost}^r, \pi, \bar{\pi}_{\pi, m, r})$, where $\text{cost}^r(x, y) = \text{cost}(x, r, y)$. Thus $\pi(x) = \sum_{y \in \mathcal{X}} \gamma_{\pi, m, r}(x, y)$ for $x \in \mathcal{X}$; $\bar{\pi}_{\pi, m, r}(y) = \sum_{x \in \mathcal{X}} \gamma_{\pi, m, r}(x, y)$ for $y \in \mathcal{X}$; $\text{cost}_{\mathcal{A}}(\pi, m, r) = \sum_{x \in \mathcal{X}, y \in \mathcal{X}} \gamma_{\pi, m, r}(x, y) \text{cost}(x, r, y)$.

We now formally describe the action of the behavioral algorithm $\tilde{\mathcal{A}}$. The initial full state of $\tilde{\mathcal{A}}$ is $(x^0, k^0) = (x^0, \pi^0, m^0)$. Given that the full state of $\tilde{\mathcal{A}}$ is (x, π, m) and the next request is $r \in \mathcal{R}$, and given any $(y, \sigma, n) \in \mathcal{X} \times \Pi \times \mathcal{M}$, we define $\Lambda_{x, \pi, m, r, y, \sigma, n}$, the probability that $\tilde{\mathcal{A}}$ chooses the next full state to be (y, σ, n) , as follows:

If $\bar{\pi}_{\pi, m, r}(y) = 0$, then $\Lambda_{x, \pi, m, r, y, \sigma, n} = 0$.

Otherwise, $\Lambda_{x, \pi, m, r, y, \sigma, n} = \frac{\gamma_{\pi, m, r}(x, y) \cdot \sigma(y) \cdot \lambda_{\pi, m, r, \sigma, n}}{\pi(x) \cdot \bar{\pi}_{\pi, m, r}(y)}$.

Let ϱ be a given request sequence. We now prove that $E(\text{cost}_{\tilde{\mathcal{A}}}(\varrho)) = E(\text{cost}_{\mathcal{A}}(\varrho))$. For any $t \geq 0$ and any knowledge state (π, m) of \mathcal{A} , define $p^t(\pi, m)$ to be the probability that the full state of \mathcal{A} is (π, m) after t steps. Additionally, if $x \in \mathcal{X}$, define $q^t(x, \pi, m)$ to be the probability that the full state of $\tilde{\mathcal{A}}$ is (x, π, m) after t steps.

To prove the lemma we consider first the following two claims:

1. For any $t \geq 0$, $x \in \mathcal{X}$, $\pi \in \Pi$, and $m \in \mathcal{M}$, $q^t(x, \pi, m) = p^t(\pi, m) \cdot \pi(x)$.
2. For any $t \geq 0$, $\pi \in \Pi$, and $m \in \mathcal{M}$, $\sum_{x \in \mathcal{X}} q^t(x, \pi, m) = p^t(\pi, m)$.

We prove claims 1 and 2 by simultaneous induction on t . If $t = 0$, both claims are trivial by definition. Now, suppose $t > 0$. We verify claim 1 for t . By the inductive hypothesis, claim 2 holds for $t - 1$. Write $r = r^t$. Let $y, \sigma, n \in \mathcal{X} \times \Pi \times \mathcal{M}$. If (σ, n) is not a full state of \mathcal{A} or $\sigma(y) = 0$, we are done. Otherwise, recall that $\bar{\pi}_{\pi, m, r}(y) = \sum_{x \in \mathcal{X}} \gamma_{\pi, m, r}(x, y)$ for all $y \in \mathcal{X}$, and we obtain

$$\begin{aligned}
q^t(y, \sigma, n) &= \sum_{(x, \pi, m) \in \mathcal{X} \times \Pi \times \mathcal{M}} q^{t-1}(x, \pi, m) \Lambda_{x, \pi, m, r, y, \sigma, n} \\
&= \sum_{(x, \pi, m) \in \mathcal{X} \times \Pi \times \mathcal{M}, \pi(x) > 0, \bar{\pi}_{\pi, m, r}(y) > 0} p^{t-1}(\pi, m) \pi(x) \cdot \frac{\gamma_{\pi, m, r}(x, y) \cdot \sigma(y) \cdot \lambda_{\pi, m, r, \sigma, n}}{\pi(x) \cdot \bar{\pi}_{\pi, m, r}(y)} \\
&= \sum_{(x, \pi, m) \in \mathcal{X} \times \Pi \times \mathcal{M}, \bar{\pi}_{\pi, m, r}(y) > 0} p^{t-1}(\pi, m) \cdot \frac{\gamma_{\pi, m, r}(x, y) \cdot \sigma(y) \cdot \lambda_{\pi, m, r, \sigma, n}}{\bar{\pi}_{\pi, m, r}(y)} \\
&= \sigma(y) \cdot \sum_{(\pi, m) \in \Pi \times \mathcal{M}, \bar{\pi}_{\pi, m, r}(y) > 0} \left(p^{t-1}(\pi, m) \cdot \lambda_{\pi, m, r, \sigma, n} \cdot \sum_{x \in \mathcal{X}} \frac{\gamma_{\pi, m, r}(x, y)}{\bar{\pi}_{\pi, m, r}(y)} \right) \\
&= \sigma(y) \cdot \sum_{(\pi, m) \in \Pi \times \mathcal{M}, \bar{\pi}_{\pi, m, r}(y) > 0} p^{t-1}(\pi, m) \cdot \lambda_{\pi, m, r, \sigma, n} \\
&= \sigma(y) \cdot \sum_{(\pi, m) \in \Pi \times \mathcal{M}} p^{t-1}(\pi, m) \cdot \lambda_{\pi, m, r, \sigma, n} = \sigma(y) \cdot p^t(\sigma, n)
\end{aligned}$$

which verifies claim 1 for t . Claim 2 for t follows trivially.

For the conclusion of the lemma, let $t > 0$, and let $r = r^t$. We use claim 1 for $t - 1$. Recall that $\bar{\pi}_{\pi, m, r} = \sum_{\sigma \in \Pi, n \in \mathcal{M}} \lambda(\pi, m, r, \sigma, n) \cdot \sigma$ for any full state (π, m) of \mathcal{A} . Then

$$\begin{aligned}
E(\text{cost}_{\mathcal{A}}^t) &= \sum_{\pi, \sigma \in \Pi, m, n \in \mathcal{M}, x, y \in \mathcal{X}} q^{t-1}(x, \pi, m) \cdot \Lambda_{x, \pi, m, r, y, \sigma, n} \cdot \text{cost}(x, r, y) \\
&= \sum_{\substack{\pi, \sigma \in \Pi, m, n \in \mathcal{M}, x, y \in \mathcal{X} \\ \pi(x) > 0, \sigma(y) > 0}} p^{t-1}(\pi, m) \cdot \pi(x) \cdot \frac{\gamma_{\pi, m, r}(x, y) \cdot \sigma(y) \cdot \lambda_{\pi, m, r, \sigma, n}}{\pi(x) \cdot \bar{\pi}_{\pi, m, r}(y)} \cdot \text{cost}(x, r, y) \\
&= \sum_{\pi \in \Pi, m \in \mathcal{M}, x, y \in \mathcal{X}} \left(p^{t-1}(\pi, m) \cdot \gamma_{\pi, m, r}(x, y) \cdot \text{cost}(x, r, y) \cdot \sum_{\sigma \in \Pi, n \in \mathcal{M}, \sigma(y) > 0} \frac{\lambda_{\pi, m, r, \sigma, n} \cdot \sigma(y)}{\bar{\pi}_{\pi, m, r}(y)} \right) \\
&= \sum_{\pi \in \Pi, m \in \mathcal{M}, x, y \in \mathcal{X}} p^{t-1}(\pi, m) \cdot \gamma_{\pi, m, r}(x, y) \cdot \text{cost}(x, r, y) \\
&= \sum_{\pi \in \Pi, m \in \mathcal{M}} \left(p^{t-1}(\pi, m) \cdot \sum_{x, y \in \mathcal{X}} \gamma_{\pi, m, r}(x, y) \cdot \text{cost}(x, r, y) \right) \\
&= \sum_{\pi \in \Pi, m \in \mathcal{M}} p^{t-1}(\pi, m) \cdot \text{cost}_{\mathcal{A}}(\pi, r, \bar{\pi}_{\pi, m, r}) \\
&= \sum_{\pi \in \Pi, m \in \mathcal{M}} p^{t-1}(\pi, m) \cdot \text{cost}_{\mathcal{A}}(\pi, m, r) = E(\text{cost}_{\mathcal{A}}^t)
\end{aligned}$$

and we are done. \square

Theorem 2 *If \mathcal{A} is a mixed model online algorithm for an online problem \mathcal{P} , there exist algorithms \mathcal{A}_1 , \mathcal{A}_2 , and \mathcal{A}_3 for \mathcal{P} , of each of the standard models, such that, given any request sequence ϱ , the cost (or expected cost) of each \mathcal{A}_i for ϱ is no greater than the cost (or expected cost) of \mathcal{A} .*

Proof: From Lemma 1 and Theorem 1. \square

Corollary 1 *If there is a C -competitive mixed model online algorithm for an online problem \mathcal{P} , there is a C -competitive online algorithm for \mathcal{P} for each of the three standard models of randomized online algorithms.*

3 Knowledge State Algorithms

We say that a function $\omega : \mathcal{X} \rightarrow \mathbf{R}$ is *Lipschitz* if $\omega(y) \leq \omega(x) + d(x, y)$ for all $x, y \in \mathcal{X}$. An *estimator* is a non-negative Lipschitz function $\mathcal{X} \rightarrow \mathbf{R}$. If $S \subseteq \mathcal{X}$, we say that S *supports an estimator* ω if, for any $y \in \mathcal{X}$ there exists some $x \in S$ such that $\omega(y) = \omega(x) + d(x, y)$. If ω is supported by a finite set, then there is a unique minimal set S which supports ω , which we call the *estimator support* of ω . (We use the term “support” instead of “estimator support” if the context excludes ambiguity.) We note that all estimators considered in this paper have finite support. We say that an estimator ω has *zero minimum* if $\min_{x \in \mathcal{X}} \omega(x) = 0$. The next lemma allows us to compare estimators by examining finitely many values.

Lemma 2 *Suppose ω and ω' are estimators, and S is the support of ω . Then $\omega(x) \geq \omega'(x)$ for all $x \in \mathcal{X}$ if and only if $\omega(y) \geq \omega'(y)$ for all $y \in S$.*

Proof: One direction of the proof is trivial. Suppose $\omega(x) < \omega'(x)$ and $\omega(y) \geq \omega'(y)$ for all $y \in S$. Then there exists $y \in S$ such that $\omega(x) = \omega(y) + d(y, x)$. It follows that $\omega(y) = \omega(x) - d(y, x) < \omega'(x) - d(y, x) \leq \omega'(y)$, contradiction. \square

An example of an estimator is the *work function* of a request sequence. If $x, y \in \mathcal{X}$, we write $cost_{opt}^\varrho(x, y)$ to denote the minimal cost of servicing the request sequence ϱ starting at configuration x and ending at configuration y . Then, if ϱ is a request sequence, the *work function* $\omega^\varrho : \mathcal{X} \rightarrow \mathbf{R}$ is defined by $\omega^\varrho(x) = cost_{opt}(s^0, \varrho, x)$. If ϱ is a request sequence, the *offset function* is defined to be $\bar{\omega}^\varrho = \omega^\varrho - cost_{opt}(\varrho)$, a zero minimum estimator. If ω is an estimator and if $r \in \mathcal{R}$ is a request, we define function $\omega \wedge r$ as $(\omega \wedge r)(y) = \min_{x \in \mathcal{X}} \{\omega(x) + cost(x, r, y)\}$. We call “ \wedge ” the *update operator*. The following lemma allows us to compute the update in finitely many steps.

Lemma 3 *If ω is supported by S , then $(\omega \wedge r)(y) = \min_{x \in S} \{\omega(x) + cost(x, r, y)\}$.*

Proof: Trivially, $(\omega \wedge r)(y) \leq \min_{x \in S} \{\omega(x) + cost(x, r, y)\}$. Pick $z \in \mathcal{X}$ such that $(\omega \wedge r)(y) = \omega(z) + cost(z, r, y)$. Pick $x \in S$ such that $\omega(z) = \omega(x) + d(x, z)$. Then

$$\begin{aligned} (\omega \wedge r)(y) &= \omega(z) + cost(z, r, y) = \omega(x) + d(x, z) + cost(z, r, y) \\ &\geq \omega(x) + cost(x, r, y) \geq (\omega \wedge r)(y) \end{aligned}$$

and we are done. \square

We note that it is easy to verify that $\omega \wedge r$ is also an estimator. We briefly note the following lemma, which is well-known (see, for example, [8]).

Lemma 4 *If $\varrho = r^1 \dots r^n$, let $\varrho^t = r^1 \dots r^t$ for all $t \leq n$. Then $\omega^0(x) = d(s^0, x)$ for all $x \in \mathcal{X}$ and $\omega(\varrho^t) = \omega(\varrho^{t-1}) \wedge r^t$ for all $t > 0$.*

We use *estimators* and *adjustments* to analyze the competitiveness of an online algorithm \mathcal{A} . More specifically, the combination of estimators and adjustments allows us to estimate the optimal cost. An online algorithm does not know the optimal offline algorithm’s cost at any given time, but can keep track of the estimator, and use it as a guide. 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. Both the estimator and the adjustment may be calculated using randomization.

A *knowledge state algorithm* is a mixed online algorithm that computes an adjustment and an estimator at each step, and uses the current estimator as its memory state. 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 (π, ω) , where $\pi \in \Pi$ and $\omega : \mathcal{X} \rightarrow \mathbf{R}$ is the current estimator. We call that pair the *current knowledge state*.
2. If $k = (\pi, \omega)$ is the knowledge state and the next request is r , then \mathcal{A} computes an adjustment, a number which we call $adjust_{\mathcal{A}}(k, r)$, and uses randomization to pick a new knowledge state $k' = (\pi', \omega')$. More precisely, there are subsequent knowledge states $k_i = (\pi_i, \omega_i)$ and subsequent weights λ_i for $i = 1, \dots, m$ such that
 - (a) $(\omega \wedge r)(x) \geq adjust_{\mathcal{A}}(k, r) + \sum_{i=1}^m \lambda_i \omega_i(x)$ for each $x \in \mathcal{X}$.
 - (b) For each i , \mathcal{A} chooses k' to be k_i with probability λ_i .
 - (c) Let $\bar{\pi} = \sum_{i=1}^m \lambda_i \pi_i$. Define $cost_{\mathcal{A}}(k, r) = cost(\pi, r, \bar{\pi})$. (As defined in the previous section in terms of the transportation problem)
3. Finally, if $\varrho = r^1 \dots r^n$ is the input request sequence, and the sequence of full states of \mathcal{A} is $k^1 \dots k^n$, where $k^t = (\pi^t, \omega^t)$, we define

$$\begin{aligned} cost_{\mathcal{A}}^t(\varrho) &= cost_{\mathcal{A}}(k^{t-1}, r^t) \text{ and } adjust_{\mathcal{A}}^t(\varrho) = adjust_{\mathcal{A}}(k^{t-1}, r^t), \\ cost_{\mathcal{A}}(\varrho) &= \sum_{t=1}^n cost_{\mathcal{A}}^t(\varrho) \text{ and } adjust_{\mathcal{A}}(\varrho) = \sum_{t=1}^n adjust_{\mathcal{A}}^t(\varrho). \end{aligned}$$

If $S \subseteq \mathcal{X}$, we say that a knowledge state (π, ω) is *supported as a knowledge state* by S if ω is supported by S (in the estimator sense) and π is supported (distributionally) by S . Note that, in this case, (π, ω) can be represented by the finite set of triples $\{(x, \pi(x), \omega(x))\}_{x \in S}$. We say that a knowledge state algorithm *has finite support* if there is a uniform bound on the cardinality of the supports of the knowledge states. This bound is also called the *order* of the knowledge state algorithm.

We say that \mathcal{A} is *C-competitive as a knowledge state algorithm* if there is a constant K such that $E(\text{cost}_{\mathcal{A}}(\varrho)) \leq C \cdot E(\text{adjust}_{\mathcal{A}}(\varrho) + \omega^n(x)) + K$ for any request sequence $\varrho = r^1 \dots r^n$ and any $x \in \mathcal{X}$.

Lemma 5 *Given a request sequence $\varrho = r^1 \dots r^n$, then for all $x \in \mathcal{X}$*

$$E(\omega^n(x) + \text{adjust}_{\mathcal{A}}(\varrho)) \leq \text{cost}_{\text{opt}}^{\varrho}(s^0, x)$$

Proof: Let $s^0 = x^0, x^1, \dots, x^n = x \in \mathcal{X}$ be the optimal service of ϱ that ends in x . Thus: $\sum_{t=1}^n \text{cost}(x^{t-1}, r^t, x^t) = \text{cost}_{\text{opt}}(s^0, x)$. By (2a): $E(\omega^t(x^t) + \text{adjust}_{\mathcal{A}}^t(\varrho)) \leq E((\omega^{t-1} \wedge r^t)(x^t))$ for all t . By definition: $E((\omega^{t-1} \wedge r^t)(x^t)) \leq E(\omega^{t-1}(x^{t-1})) + \text{cost}(x^{t-1}, r^t, x^t)$ for all t . Summing the inequalities over all t , and adding to the equation, we obtain the result. \square

Lemma 6 *If a knowledge state algorithm \mathcal{A} is C-competitive as a knowledge state algorithm, then \mathcal{A} is C-competitive.*

Proof: Let K be the constant given in the definition of C-competitiveness for a knowledge state algorithm. Let $\varrho = r^1 \dots r^n$ be any request sequence, and let $s^0 = x^0, x^1, \dots, x^n \in \mathcal{X}$ be the optimal service of ϱ . Since \mathcal{A} is C-competitive as a knowledge state algorithm:

$$\begin{aligned} E(\text{cost}_{\mathcal{A}}(\varrho)) &\leq C \cdot E(\text{adjust}_{\mathcal{A}}(\varrho)) + C \cdot E(\omega^n(x^n)) + K \\ E(\text{adjust}_{\mathcal{A}}(\varrho) + \omega^n(x^n)) &\leq \text{cost}_{\text{opt}}(\varrho) \quad (\text{by lemma 5}) \end{aligned}$$

We obtain:

$$E(\text{cost}_{\mathcal{A}}(\varrho)) \leq C \cdot \text{cost}_{\text{opt}}(\varrho) + K$$

\square

We now define a *C-knowledge state potential* (C-ks-potential, for short) for a given knowledge state algorithm \mathcal{A} . Let $\Phi_{\mathcal{A}}$ be a real-valued function on knowledge states. Then we say that $\Phi_{\mathcal{A}}$ is a *C-ks-potential for \mathcal{A}* if

1. $\Phi_{\mathcal{A}}(k) \geq 0$ for any k .
2. If $k = (\pi, \omega)$ is the current knowledge state and r is the next request, $\{k_i = (\pi_i, \omega_i)\}$ are the subsequents of that request, and $\{\lambda_i\}$ are the weights of the subsequents, let $\Delta\Phi_{\mathcal{A}}(k, r) = \sum_{i=1}^m \lambda_i \Phi_{\mathcal{A}}(\pi_i, \omega_i) - \Phi_{\mathcal{A}}(\pi, \omega)$. Then

$$\text{cost}_{\mathcal{A}}(k, r) + \Delta\Phi_{\mathcal{A}}(k, r) \leq C \cdot \text{adjust}_{\mathcal{A}}(k, r).$$

Theorem 3 *If a knowledge state algorithm \mathcal{A} has a C-ks-potential, then \mathcal{A} is C-competitive.*

Proof: The proof follows easily from the definition of a C-ks-potential and Lemmas 5 and 6 by straightforward arguments. Let $\varrho = r^1 \dots r^n$ be a request sequence. Let k^1, \dots, k^n be the sequence of knowledge states of \mathcal{A} given the input ϱ , where $k^t = (\pi^t, \omega^t)$. Let $\Phi_{\mathcal{A}}^t = \Phi_{\mathcal{A}}(k^t)$, a random variable for each t . Note that $\Phi_{\mathcal{A}}^0$ is a constant. Let $\Delta^t\Phi_{\mathcal{A}} = \Delta\Phi_{\mathcal{A}}(k^{t-1}, r^t)$. Note that $E(\Delta^t\Phi_{\mathcal{A}}) = E(\Phi_{\mathcal{A}}^t - \Phi_{\mathcal{A}}^{t-1})$. Let $x \in \mathcal{X}$ be the configuration of the optimal algorithm after n steps.

Then

$$\begin{aligned}
C \cdot \text{cost}_{\text{opt}}(\varrho) - E(\text{cost}_{\mathcal{A}}(\varrho)) &\geq \\
C \cdot E(\omega^n(x) + \text{adjust}_{\mathcal{A}}(\varrho)) - E(\text{cost}_{\mathcal{A}}(\varrho)) &= \\
C \cdot E\left(\omega^n(x) + \sum_{t=1}^n \text{adjust}_{\mathcal{A}}^t(\varrho)\right) - E\left(\sum_{t=1}^n \text{cost}_{\mathcal{A}}^t(\varrho)\right) &= \\
E\left(C \cdot \omega^n(x) + \sum_{t=1}^n (C \cdot \text{adjust}_{\mathcal{A}}^t(\varrho) - \text{cost}_{\mathcal{A}}^t(\varrho))\right) &= \\
E\left(C \cdot \omega^n(x) + \Phi_{\mathcal{A}}^n + \sum_{t=1}^n (C \cdot \text{adjust}_{\mathcal{A}}^t(\varrho) - \text{cost}_{\mathcal{A}}^t(\varrho) - \Delta^t \Phi_{\mathcal{A}})\right) - \Phi_{\mathcal{A}}^0 &\geq \\
E(C \cdot \omega^n(x) + \Phi_{\mathcal{A}}^n) - \Phi_{\mathcal{A}}^0 &\geq -\Phi_{\mathcal{A}}^0
\end{aligned}$$

The first inequality above is from Lemma 5. The last two inequalities are from the definition of a C -ks-potential. It follows that $E(\text{cost}_{\mathcal{A}}(\varrho)) \leq C \cdot \text{cost}_{\text{opt}}(\varrho) + \Phi_{\mathcal{A}}^0$, and, by Lemma 6, we are done. \square

We can define a *forgiveness* online algorithm to be a knowledge state algorithm with the special restriction that there is always exactly one subsequent. We note that historically, forgiveness came first, so we can think of the knowledge state approach as being a generalization of forgiveness. A forgiveness algorithm can be deterministic, such as EQUIPOISE, a deterministic online 11-competitive algorithm for the 3-server problem (that was the best known competitiveness for that problem at that time), or distributional, such as EQUITABLE, an H_k -competitive distributional online algorithm for the k -cache problem. (See [1, 9].)

4 Knowledge State Algorithms for the Cache Problem

We now consider the k -cache problem for fixed $k \geq 2$. The k -cache problem reduces to online optimization, as defined in Section 2 of this paper, as follows:

1. There is a set of *pages*.
2. \mathcal{X} is the set of all k -tuples of distinct pages. If the configuration of an algorithm is $x \in \mathcal{X}$, that means that the pages that constitute x are in the cache.
3. The initial configuration is the initial cache.
4. If $x, y \in \mathcal{X}$, then $d(x, y)$ is the cost of changing the cache from x to y . Since we assume that it costs 1 to eject a page and bring in a new page, $d(x, y)$ is the cardinality of the set $x - y$.
5. \mathcal{R} is simply the set of all pages. If a page r is requested, it means that the algorithm must ensure that r is in the cache at some point as it moves between configurations. Thus, for any $x, y \in \mathcal{X}$ and any $r \in \mathcal{R}$, we have

$$\text{cost}(x, r, y) = \begin{cases} 2 & \text{if } x = y, r \notin x \\ d(x, y) & \text{if } r \in x \text{ or } r \in y \\ d(x, y) + 1 & \text{otherwise} \end{cases}$$

To complete the reduction, we observe that the support of any configuration request pair (x, r) is finite. If $r \in x$, that support has only one element, namely x , while otherwise, it has k elements, namely $\{x - a + r \mid a \in x\}$.

Bar Notation for the Cache Problem. We introduce a convenient notation, a modification of the bar notation of Koutsoupias and Papadimitriou [11], for offset functions for the k -cache problem, which we call the *bar notation*.¹ Let α 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 α defines an offset function ω as follows. Let $S \subseteq \mathcal{X}$ be the set of all configurations x 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. Let ω be the estimator such that S is the support of ω , and such that $\omega(x) = 0$ for each $x \in S$. For example for $k = 2$, $ab||$ denotes the estimator whose support consists of just the configuration $\{a, b\}$, and which takes the value zero on that configuration. For $k = 4$, $ab||cd|ef|$ denotes the estimator whose support consists of the configurations $\{a, b, c, d\}$, $\{a, b, c, e\}$, $\{a, b, c, f\}$, $\{a, b, d, e\}$, and $\{a, b, d, f\}$, and which takes the value zero on those configurations. From [11], we have:

Lemma 7 *A function ω is an offset function for the k -cache problem if and only if it can be expressed using the bar notation.*

4.1 A $\frac{3}{2}$ -Competitive Knowledge State Algorithm for the 2-Cache Problem

Recall that PARTITION (introduced in [11]) is optimally competitive for the k -cache problem, but uses unbounded memory to achieve the optimal competitiveness of H_k . The memory state of PARTITION is, in fact, the classic offset function, which, in the worst case, requires keeping track of every past request. We now show how the use of knowledge states simplifies the definition, and in fact the memory requirement, of an optimally competitive randomized algorithm for the 2-cache problem, which we call K_2 .

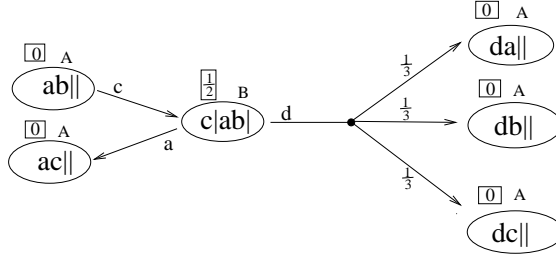


Figure 1: Schematic for the 2-Cache Knowledge State Algorithm

Knowledge States of K_2 . We will follow the rule that, at each step, the adjustment is as large as possible, so that the minimum of the estimator will always be zero. This guarantees that any potential will always be non-negative. If there are infinitely many pages, K_2 has infinitely many knowledge states, but, up to symmetry, it has only two. Each such knowledge state of K_2 is supported by a set of cardinality at most 2, hence has at most three active pages, and therefore its equivalent behavioral algorithm has at most one bookmark.

In the definitions given below, we say that two pages are *equivalent* for a given knowledge state if they can be transposed without changing the knowledge state.

1. If a, b are pages, let $A^{a,b} = (\{a, b\}, ab||)$. In this case, a and b are equivalent, *i.e.*, $A^{a,b} = A^{b,a}$.
2. If a, b, c are pages, let $B^{a,b,c} = (\frac{1}{2}\{a, b\} + \frac{1}{2}\{a, c\}, a|bc|)$, where $\frac{1}{2}\{a, b\} + \frac{1}{2}\{a, c\}$ denotes the distribution which is $\frac{1}{2}$ on the configuration $\{a, b\}$ and $\frac{1}{2}$ on the configuration $\{a, c\}$. In this case b and c are equivalent, *i.e.*, $B^{a,b,c} = B^{a,c,b}$.

¹The notation of [11] differs slightly from that given here, although it is based on the same concept.

We list below the action of K_2 . In each case, a, b, c, d are distinct pages.

1. If $\{a, b\}$ is the initial cache, the initial knowledge state is $A^{a,b}$.
2. If the current knowledge state is $A^{a,b}$ then
 - (i) if the request is a , the new knowledge state is $A^{a,b}$.
 - (ii) if the request is c , then the new knowledge state is $B^{c,a,b}$.
3. If the current knowledge state is $B^{a,b,c}$ then
 - (i) if the new request is a , the new knowledge state is $B^{a,b,c}$.
 - (ii) if the new request is b , the new knowledge state is $A^{b,a}$.
 - (iii) if the new request is $d \notin \{a, b, c\}$, then there are three subsequents, namely $A^{d,a}$, $A^{d,b}$, $A^{d,c}$. The distribution on the subsequents is uniform, *i.e.*, each is chosen with probability $\frac{1}{3}$.

Actions 2i and 3i are requests to the first block of pages, in the sense of the bar notation. Since the bar notation implies that each page in the first block can be assumed to be in the cache, such a request is ignored by any sensible online algorithm, which means, in our case, that the estimator is unchanged and the adjustment is zero. We call such requests *trivial*.

We define a potential Φ by $\Phi(A^{a,b}) = 0$ and $\Phi(B^{a,b,c}) = \frac{1}{2}$.

Lemma 8 Φ is a $\frac{3}{2}$ -ks-potential for K_2 .

Proof: Let k be the current knowledge state and r the new request. Write $\Delta\Phi$ for increase in potential in the given step. We will show that

$$cost + \Delta\Phi \leq \frac{3}{2}adjust \quad (1)$$

in all cases. In trivial actions, namely Cases 2i and 3i, $cost = \Delta\Phi = adjust$, and we are done.

We first note that:

$$\begin{aligned} ab|| \wedge c &= c|ab| + 1 \\ a|bc| \wedge b &= ab|| \\ a|bc| \wedge d &\geq \frac{1}{3}da|| + \frac{1}{3}db|| + \frac{1}{3}dc|| + \frac{1}{3} \end{aligned}$$

By Lemma 2, the last inequality need only be verified for configurations in $\{\{d, a\}, \{d, b\}, \{d, c\}\}$, the support set of $a|bc| \wedge d$.

Case Action 2ii: In this case $k = A^{a,b}$ and r is a new page, c .

$ab|| \wedge c = c|ab| + 1$. thus $adjust = 1$. Since the algorithm must bring in a new page, and since the probability is zero that the minimum transport brings in any other page, $cost = 1$. $\Delta\Phi = \frac{1}{2}$, and we are done.

Case Action 3ii: *i.e.*, $k = B^{a,b,c}$ and $r = b$.

Recall $a|bc| \wedge b = ab||$. Note that $adjust = 0$, since, as functions, $ab|| \geq a|bc|$ on the set of all configurations. $cost = \frac{1}{2}$, since the probability is $\frac{1}{2}$ that the algorithm does nothing, and the probability is $\frac{1}{2}$ that it ejects c and brings in b . $\Delta\Phi = -\frac{1}{2}$, and we are done.

Case Action 3iii: *i.e.*, $k = B^{a,b,c}$ and r is a new page, d .

Recall $a|bc| \wedge d \geq \frac{1}{3}da|| + \frac{1}{3}db|| + \frac{1}{3}dc|| + \frac{1}{3}$, thus $adjust = \frac{1}{3}$. Since the algorithm must bring in a new page, and since the probability is zero that the minimum transport brings in any other page, $cost = 1$. $\Delta\Phi = -\frac{1}{2}$, and we are done.

This completes the proof of all cases. \square

We have:

Corollary 2 K_2 is $\frac{3}{2}$ -competitive.

We note that the number of active pages, *i.e.*, pages contained in a support configuration, is never more than three. The number three is minimal, as given by the theorem below:

Theorem 4 *There is no knowledge state algorithm for the 2-cache problem that is $\frac{3}{2}$ -competitive as a knowledge state algorithm, and which never has more than two active pages, *i.e.*, no bookmarks.*

Proof: If a knowledge state algorithm for the 2-cache problem never has more than two active pages, then it can have no bookmarks, hence is trackless. By Theorem 2 of [3], there is no $\frac{3}{2}$ -competitive trackless online algorithm for the 2-cache problem. \square

4.2 An Optimally Competitive Knowledge State Algorithm for the 3-Cache Problem

We define a knowledge state algorithm K_3 which is H_3 -competitive for the 3-cache problem. Recall that $H_3 = \frac{11}{6}$. Up to symmetry, K_3 has six knowledge states. The number of active pages, *i.e.*, pages contained in a support configuration, is never more than five.

The knowledge states of K_3 will be defined as follows. As in the case of K_2 , We say that two pages are *equivalent* if they can be transposed without changing the knowledge state.

1. $A^{a,b,c} = (\{a, b, c\}, abc|)|)$ for any three pages a, b, c . The pages a, b , and c are all equivalent, *i.e.*, $A^{a,b,c} = A^{b,a,c} = A^{a,c,b}$, *etc.*
2. $B^{a,b,c,d} = (\frac{1}{3}\{a, b, c\} + \frac{1}{3}\{a, b, d\} + \frac{1}{3}\{a, c, d\}, a|bcd|)|)$ for any four pages a, b, c, d . The pages b, c , and d are all equivalent.
3. $C^{a,b,c,d} = (\frac{1}{2}\{a, b, c\} + \frac{1}{2}\{a, b, d\}, ab||cd)$ for any four pages a, b, c, d . The pages a and b are equivalent, and c and d are equivalent.
4. $D^{a,b,c,d,e} = (\frac{1}{6}\{a, b, c\} + \frac{1}{6}\{a, b, d\} + \frac{1}{6}\{a, b, e\} + \frac{1}{6}\{a, c, d\} + \frac{1}{6}\{a, c, e\} + \frac{1}{6}\{ade\}, a|bcde|)|)$ for any five pages a, b, c, d, e . The pages b, c, d, e are equivalent.
5. $E^{a,b,c,d,e} = (\frac{1}{2}\{a, b, c\} + \frac{1}{4}\{a, b, d\} + \frac{1}{4}\{a, b, e\}, ab||cde|)|)$ for any five pages a, b, c, d, e . The pages a and b are equivalent, and d and e are equivalent.
6. $F^{a,b,c,d,e} = (\frac{1}{2}\{a, b, c\} + \frac{1}{8}\{a, b, d\} + \frac{1}{8}\{a, b, e\} + \frac{1}{8}\{a, c, d\} + \frac{1}{8}\{a, c, e\}, a|bc|de|)|)$ for any five pages a, b, c, d, e . The pages b and c are equivalent, and d and e are equivalent.

The actions of K_3 are formally defined below. In each case, a, b, c, d, e, f are distinct pages. We do not need to consider separate cases for requests to pages which are equivalent.

1. If $\{a, b, c\}$ is the initial cache, the initial knowledge state is $A^{a,b,c}$.
2. If the current knowledge state is $A^{a,b,c}$ then
 - (i) if the new request is a , the new knowledge state is $A^{a,b,c}$.
 - (ii) if the new request is some page $d \notin \{a, b, c\}$, the new knowledge state is $B^{d,a,b,c}$.
3. If the current knowledge state is $B^{a,b,c,d}$ then
 - (i) if the new request is a , the new knowledge state is $B^{a,b,c,d}$.
 - (ii) if the new request is b , the new knowledge state is $C^{a,b,c,d}$.
 - (iii) if new request is some page $e \notin \{a, b, c, d\}$, the new knowledge state is $D^{e,a,b,c,d}$.

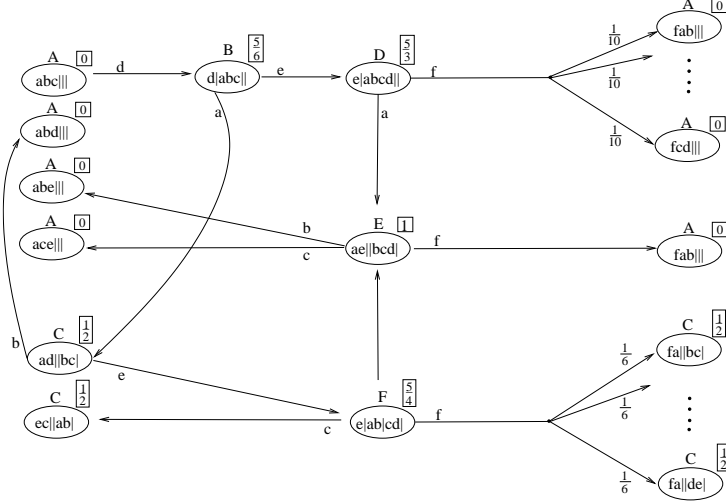


Figure 2: Schematic for the 3-Cache Knowledge State Algorithm

4. If the current knowledge state is $C^{a,b,c,d}$ then
 - (i) if the new request is a , the new knowledge state is $C^{a,b,c,d}$.
 - (ii) if the new request is c , the new knowledge state is $A^{a,b,c}$.
 - (iii) if the new request is some page $e \notin \{a, b, c, d\}$, the new knowledge state is $F^{e,a,b,c,d}$.
5. If the current knowledge state is $D^{a,b,c,d,e}$ then
 - (i) if the new request is a , the new knowledge state is $D^{a,b,c,d,e}$.
 - (ii) if the new request is b , the new knowledge state is $E^{a,b,c,d,e}$.
 - (iii) if the new request is some page $f \notin \{a, b, c, d, e\}$, then the new knowledge state is chosen uniformly from among the following ten knowledge states: A^{abc} , A^{abd} , A^{abe} , A^{abf} , A^{acd} , A^{ace} , A^{acf} , A^{ade} , A^{adf} , and A^{aef} .
6. If the current knowledge state is $E^{a,b,c,d,e}$ then
 - (i) if the new request is a , the new knowledge state is $E^{a,b,c,d,e}$.
 - (ii) if the new request is c , the new knowledge state is $A^{a,b,c}$.
 - (iii) if the new request is d , the new knowledge state is $A^{a,b,d}$.
 - (iv) if the new request is some page $f \notin \{a, b, c, d, e\}$, then the new knowledge state is $A^{f,a,b}$.
7. If the current knowledge state is $F^{a,b,c,d,e}$ then
 - (i) if the new request is a , the new knowledge state is $F^{a,b,c,d,e}$.
 - (ii) if the new request is b , the new knowledge state is $E^{a,b,c,d,e}$.
 - (iii) if the new request is d , the new knowledge state is $C^{a,d,b,c}$.
 - (iv) if the new request is some page $f \notin \{a, b, c, d, e\}$, the new knowledge state is chosen uniformly from among the following six knowledge states: $C^{f,a,b,c}$, $C^{f,b,a,c}$, $C^{f,c,a,b}$, $C^{f,a,d,e}$, $C^{f,b,d,e}$, and $C^{f,c,d,e}$.

We define a potential Φ on the knowledge states as follows: $\Phi(A^{a,b,c}) = 0$, $\Phi(B^{a,b,c,d}) = \frac{5}{6}$, $\Phi(C^{a,b,c,d}) = \frac{1}{2}$, $\Phi(D^{a,b,c,d,e}) = \frac{1}{2}$, $\Phi(E^{a,b,c,d,e}) = 1$, and $\Phi(F^{a,b,c,d,e}) = \frac{5}{4}$.

Lemma 9 Φ is an $\frac{11}{6}$ -ks-potential for K_3 .

Proof: For each action of K_3 , let $\Delta\Phi$ be the increase in potential. We will show that

$$cost + \Delta\Phi \leq \frac{11}{6} adjust \quad (2)$$

In each case, the value of $\Delta\Phi$ can be computed by simple subtraction. We need only compute the values of $cost$ and $adjust$ for each action, after which the inequality (2) follows by simple arithmetic. Case Actions 2i, 3i, 4i, 5i, 6i, 7i:. These actions are trivial, and thus $adjust = cost = \Delta\Phi = 0$, and we are done.

Case Actions 2ii, 3iii, 4iii:. In these actions, the request is to a new page, and the probability that any other page is in the cache after the action does not increase: thus $cost = 1$. We also know that $adjust = 1$ because

$$\begin{aligned} abc||\wedge d &= d|abc|| + 1 \\ a|bcd||\wedge e &= e|abcd|| + 1 \\ ab||cd|\wedge e &= e|ab|cd|| + 1 \end{aligned}$$

The remainder of the verification of (2) for each of those actions consists of simple arithmetic.

Case Actions 6ii and 6iii:. Note that $adjust = 0$ since

$$\begin{aligned} ab|cde||\wedge c &= abc||| \\ ab|cde||\wedge d &= abd||| \end{aligned}$$

In each case, we must keep a and b and eject the other two unrequested pages. The probability is $\frac{1}{2}$ that c is in our cache, and $\frac{1}{4}$ that d is in our cache, thus $cost = \frac{1}{2}$ for Action 6ii, and $cost = \frac{3}{4}$ for Action 6iii. Since $\Delta\Phi = -\frac{1}{2}$ for both actions, we are done.

Case 5ii and 7ii:. Note that $adjust = 0$ since

$$\begin{aligned} a|bcde||\wedge b &= ab|cde|| \\ a|bc|de|\wedge b &= ab|cde|| \end{aligned}$$

For Action 5ii, recall that the distribution of $D^{a,b,c,d,e}$ is uniform on six configurations. To compute $cost$, we describe a minimal transport between the distribution of $D^{a,b,c,d,e}$ and the distribution of $E^{a,b,c,d,e}$. That transport is defined as follows:

If the previous configuration is $\{a, b, c\}$, $\{a, b, d\}$, or $\{a, b, e\}$, do nothing.

If the previous configuration is $\{a, c, d\}$, eject d .

If the previous configuration is $\{a, c, e\}$, eject e .

If the previous configuration is $\{a, d, e\}$, eject d with probability $\frac{1}{2}$, and eject e with probability $\frac{1}{2}$.

Thus, $cost = \frac{1}{2}$. It is a routine verification that the required distribution for $E^{a,b,c,d,e}$ is achieved. Since $\Delta\Phi = -\frac{2}{3}$, we have verified (2) for Action 5ii.

For Action 7ii, recall that the distribution of $F^{a,b,c,d,e}$ is $\frac{1}{2}$ on $\{a, b, c\}$, and is $\frac{1}{8}$ on each of $\{a, b, d\}$, $\{a, b, e\}$, $\{a, c, d\}$, and $\{a, c, e\}$. A minimal transport can be defined as follows: if b is

already in the cache we do nothing, while otherwise, we eject c . Thus, $cost = \frac{1}{4}$. It is a routine verification that the required distribution for $E^{a,b,c,d,e}$ is achieved. Since $\Delta\Phi = -\frac{1}{4}$, we have verified (2) for Action 7ii.

Case Actions 3ii, 4ii, 7iii: Note that $adjust = 0$ since

$$\begin{aligned} a|bcd||\wedge b &= ab|cd| \\ ab||cd|\wedge c &= abc|| \\ a|bc|de|\wedge d &= ad||bc| \end{aligned}$$

For Action 3ii, recall that the distribution of $B^{a,b,c,d}$ is uniform on $\{a, b, c\}$, $\{a, b, d\}$, and $\{a, c, d\}$. If b is already in the cache we do nothing, while otherwise, we eject c with probability $\frac{1}{2}$ and eject d with probability $\frac{1}{2}$. Thus, $cost = \frac{1}{3}$. It is a routine verification that the required distribution for $C^{a,b,c,d}$ is achieved. Since $\Delta\Phi = -\frac{1}{3}$, we have verified (2) for Action 3ii.

For Action 4ii, recall that the distribution of $C^{a,b,c,d}$ is uniform on $\{a, b, c\}$ and $\{a, b, d\}$. If c is already in the cache we do nothing, while otherwise, we eject d . Thus, $cost = \frac{1}{2}$. The resulting distribution is concentrated at $\{a, b, c\}$, as required for the knowledge state $A^{a,b,c}$. Since $\Delta\Phi = -\frac{1}{2}$, we have verified (2) for Action 4ii.

For Action 7iii, recall that the distribution of $F^{a,b,c,d,e}$ is $\frac{1}{2}$ on $\{a, b, c\}$, and is $\frac{1}{8}$ on each of $\{a, b, d\}$, $\{a, b, e\}$, $\{a, c, d\}$, and $\{a, c, e\}$. If d is already in the cache we do nothing. If e is in the cache, we eject e . Otherwise, the cache must be $\{a, b, c\}$, in which case we eject b or c with equal probability. Thus, $cost = \frac{3}{4}$. It is a routine verification that the required distribution for $C^{a,d,b,c}$ is achieved. Since $\Delta\Phi = -\frac{3}{4}$, we have verified (2) for Action 7iii.

Case Action 6iv: Note that $adjust \geq 0$ since $ab||cde|\wedge f = f|ab|cde| + 1 \geq abf|||$. By Lemma 2, this inequality need only be verified for the configurations in the support of $ab||cde|\wedge f$. Whatever the initial configuration is, a and b are in the cache. Simply eject the other page. Thus, $cost = 1$. $\Delta\Phi = -1$, and we are done.

Case Actions 5iii, 7iv: Let

$$\omega^{Df} = \frac{1}{10}(fab||| + fac||| + fad||| + fae||| + fbc||| + fbd||| + fbe||| + fcd||| + fde|||),$$

and let

$$\omega^{Ff} = \frac{1}{6}(fa||bc| + fb||ac| + fc||ab| + fa||de| + fb||de| + fc||de|).$$

We note:

$$\begin{aligned} a|bcde||\wedge f &= f|abcde|| + 1 \geq \omega^{Df} \\ a|bc|de|\wedge f &= f|abc|de| + 1 \geq \omega^{Ff} - \frac{1}{6} \end{aligned}$$

By Lemma 2, these inequalities need only be verified for the configurations in the support of $a|bcde||\wedge f$ and $a|bc|de|\wedge f$, respectively. We thus have $adjust \geq 0$ for 5iii, and $adjust \geq \frac{1}{6}$ for 7iv.

To compute $cost$, we give minimal transportations from the distribution of $D^{a,b,c,d,e}$, respectively $F^{a,b,c,d,e}$, to the weighted sum of distributions of the subsequents, for each of the two cases. For Action 5iii, whatever the initial configuration is, a is in the cache. Eject a with probability $\frac{3}{5}$, and eject each of the other two pages with probability $\frac{1}{5}$ each. It is a routine verification that the required distribution is achieved. Thus, $cost = 1$. $\Delta\Phi = -\frac{5}{3}$, and we are done.

For Action 7iv, the probability is $\frac{1}{2}$ that the initial configuration is $\{a, b, c\}$. In this case, eject one of the three pages, each with probability $\frac{1}{3}$. Otherwise, the cache will contain a , and either

b or c but not both: eject a with probability $\frac{2}{3}$, and otherwise eject either b or c . It is a routine verification that the required distribution is achieved. Thus, $\text{cost} = 1$. $\Delta\Phi = -\frac{3}{4}$, and we are done.

This completes the proof of all cases. \square

Corollary 3 K_3 is $\frac{11}{6}$ -competitive.

5 Experimental Work and the Server Problem

It is our hope that our technique will yield an order 2 knowledge state algorithm whose competitiveness is provably less than 2 for all metric spaces.

We mention briefly progress by giving results for a class of is “one step up” in complexity from the class of uniform metric spaces. We consider the class of metric spaces $M_{2,4}$, which consists of all metric spaces where every distance is either 1 or 2, and where the perimeter of every triangle is either 3 or 4. (The classic octahedral graph, which has six points, is a member of this class, as defined by Schläfli [12].) We have a computer generated order 2 knowledge state algorithm for the 2-server problem in this class: its competitiveness is $\frac{7}{4}$. We note that we also have calculated (through computer experimentation) the minimum value of C in the sense that no lower competitiveness for any order 2 knowledge state algorithm for $M_{2,4}$ can be proved using the methods described here. This value is $C = \frac{173+\sqrt{137}}{112}$. We briefly mention that there is an order 3 knowledge state algorithm for $M_{2,4}$ which has, up to equivalence, only seven knowledge states, and is $\frac{19}{12}$ -competitive. We also can prove that no randomized online algorithm for the 2-server problem for $M_{2,4}$ can achieve competitiveness less than $\frac{19}{12}$. All knowledge states and probabilities in this order 3 algorithm can be described using only rational numbers.

These results, as well as our results for the server problem in uniform spaces (equivalent to the caching problem), indicate a natural trade-off between competitiveness and memory of online randomized algorithms.

References

- [1] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234:203–218, 2000.
- [2] Yair Bartal, Marek Chrobak, and Lawrence L. Larmore. A randomized algorithm for two servers on the line. *Information and Computation*, 158:53–69, 2000.
- [3] Wolfgang Bein, Rudolph Fleischer, and Lawrence L. Larmore. Limited bookmark randomized online algorithms for the paging problem. *Information Processing Letters*, 76:155–162, 2000.
- [4] Wolfgang Bein and Lawrence L. Larmore. Trackless online algorithms for the server problem. *Information Processing Letters*, 74:73–79, 2000.
- [5] Wolfgang Bein and Lawrence L. Larmore. Trackless and limited bookmark algorithms for paging. *SIGACT News*, 35:40–49, 2004.
- [6] Wolfgang Bein, Lawrence L. Larmore, and Rüdiger Reischuk. Knowledge states for the caching problem in shared memory multiprocessor systems. In *Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks*, pages 307–312, IEEE, 2004.
- [7] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

- [8] Marek Chrobak and Lawrence L. Larmore. The server problem and on-line games. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 7, pages 11–64, 1992.
- [9] Marek Chrobak and Lawrence L. Larmore. Generosity helps or an 11-competitive algorithm for three servers. *Journal of Algorithms*, 16:234–263, 1994.
- [10] Don Coppersmith, Peter G. Doyle, Prabhakar Raghavan, and Marc Snir. Random walks on weighted graphs and applications to online algorithms. In *Proc. 22nd Symp. Theory of Computing (STOC)*, pages 369–378. ACM, 1990.
- [11] Elias Koutsoupias and Christos Papadimitriou. Beyond competitive analysis. In *Proc. 35th Symp. Foundations of Computer Science (FOCS)*, pages 394–400. IEEE, 1994.
- [12] Ludwig Schläfli. *Theorie der vielfachen Kontinuität*. Birkhäuser, Basel, 1857.