

# Knowledge State Algorithms

Wolfgang Bein <sup>\*</sup>   Lawrence L. Larmore <sup>†</sup>   John Noga <sup>‡</sup>   Rüdiger Reischuk <sup>§</sup>

## Abstract

We introduce the novel 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 this method to randomized  $k$ -paging. Borodin and El Yaniv [9] list as an open question whether there exists an  $H_k$ -competitive randomized algorithm which requires  $O(k)$  memory for  $k$ -paging. In this paper we answer this question in the affirmative.

*Keywords:* Competitive Analysis; Online Algorithms; Task Systems; Randomization; Paging; Server Problem.

## 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 to address the trade-off between memory and competitiveness. A *knowledge state* gives a distribution for the algorithm, while at the same time stating conditional obligations of an adversary by fixing an *estimator*, an approximation of the work function. 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. Fundamentally, the knowledge state concept enables us to remember limited information while achieving 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  $k$ -paging 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` [15], 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` [13] is an extremely simple knowledge state algorithm, which achieves randomized

---

<sup>\*</sup>Department of Computer Science, University of Nevada, Las Vegas, NV 89154. Email: [bein@cs.unlv.edu](mailto:bein@cs.unlv.edu). Research supported by NSF grant CCR-9821009.

<sup>†</sup>Department of Computer Science, University of Nevada, Las Vegas, NV 89154. Email: [larmore@cs.unlv.edu](mailto:larmore@cs.unlv.edu). Research supported by NSF grant CCR-9821009.

<sup>‡</sup>Department of Computer Science, California State University, Northridge, CA 91330. Email: [jnoga@csun.edu](mailto:jnoga@csun.edu).

<sup>§</sup>Institut für Theoretische Informatik, Universität Lübeck, Wallstraße 40, D-23560 Lübeck. Email: [reischuk@tcs.uni-luebeck.de](mailto:reischuk@tcs.uni-luebeck.de).

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 ACM SIGACT column [6] for a summary of tracklessness; see also [4, 5, 3, 7].)

The randomized  $k$ -paging algorithm EQUITABLE given by Achlioptas *et al.* [1] is  $H_k$ -competitive and uses  $O(k^2 \log k)$  memory. This competitive ratio is best possible. The randomized algorithm RMARK given by Fiat *et al.* [14] is  $(2H_k - 1)$ -competitive, but only uses  $O(k)$  memory. Borodin and El Yaniv [9] list as an open question whether there exists an  $H_k$ -competitive randomized algorithm which requires  $O(k)$  memory for  $k$ -paging. In this paper we answer this question in the affirmative using knowledge states. Chrobak, Koutsoupias and Noga [10] claim that, “From a purely practical standpoint, non-trackless algorithms are of limited interest as cache replacement strategies, as they cannot be realistically implemented.” On the other hand, Bein and Larmore [5] have shown that it is not possible for a trackless algorithm to achieve  $H_k$ -competitiveness if  $k = 2$ . Our solution seeks to provide an optimally competitive algorithm “as close as possible” to trackless.

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. Section 2 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. Section 4 gives the results for the paging problem. We start with the case  $k = 2$  to illustrate our method. The algorithm for  $k = 2$  is optimally competitive and uses provably the smallest amount of memory. Next the  $H_k$ -competitive randomized algorithm with  $O(k)$  memory is given. Section 5 summarizes knowledge state results for the 2-server problem in Cross Polytope Spaces and for the caching problem in shared memory multiprocessor systems.

## 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 paging problem, the CNN problem, etc.

Given a *request sequence*  $\rho = r^1, \dots, r^n$ , 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  $\rho$  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  $\varrho$ ) such that  $cost_{\mathcal{A}}(\varrho) \leq C \cdot cost_{opt}(\varrho) + K$  for any request sequence  $\varrho$ . 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  $\varrho$ ) such that  $E(cost_{\mathcal{A}}(\varrho)) \leq C \cdot cost_{opt}(\varrho) + K$  for any request sequence  $\varrho$ , 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  $\Pi$  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  $\pi$  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  $\pi$ . By abuse of notation, if the support of  $\pi$  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  $(A, B, cost, \alpha, \beta)$  where  $A$  and  $B$  are finite non-empty sets,  $\alpha$  is a distribution on  $A$ ,  $\beta$  is a distribution on  $B$ , and  $cost$  is a real-valued function on  $A \times B$ . A *solution* to this instance is a distribution  $\gamma$  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  $cost(\gamma) = \sum_{a \in A} \sum_{b \in B} \gamma(a, b) cost(a, b)$ , and  $\gamma$  is a *minimal* solution if  $cost(\gamma)$  is minimized over all solutions, in which case we call  $cost(\gamma)$  the *minimum transportation cost*.

There are three standard models of randomized online algorithms (see, for example [9]). 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  $\pi$  and  $S'$  be the support of  $\pi'$ . We then define  $d(\pi, \pi')$  to be the minimum transportation cost of the transportation problem  $(S, S', d, \pi, \pi')$ , and if  $r \in \mathcal{R}$ , we define  $cost(\pi, r, \pi')$  to be the minimum transportation cost of the transportation problem  $(S, S', cost^r, \pi, \pi')$ , where  $cost^r = cost(\cdot, r, \cdot) : \mathcal{X} \times \mathcal{X} \rightarrow \mathbf{R}$ .

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  $\pi$ ,  $m$ , 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 [9].)

**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  $\varrho$ , the cost (or expected cost) of each  $\mathcal{A}_i$  for  $\varrho$  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  $\Pi$  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  $\lambda_i$ . 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  $r$ ;  $k', \pi'$ , 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  $t > 0$ ,  $k^t, \pi^t$ , 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.

Let  $k = (\pi, m) \in \Pi \times \mathcal{M}$  and let  $r \in \mathcal{R}$ . Let  $S \subseteq \mathcal{X}$  be the support of  $\pi$ . 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  $\bar{S} \subseteq \mathcal{X}$  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  $\bar{S}$ . 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  $\varrho$ ,  $E(\text{cost}_{\mathcal{A}'}(\varrho)) = E(\text{cost}_{\mathcal{A}}(\varrho))$ .*

*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  $(\pi, m)$  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  $(\pi, m)$  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  $(\pi, m)$  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  $(\sigma, n)$  to be the next full state, given that the current full state is  $(\pi, m)$  and the request is  $r$ . We assume that there can be at most finitely many choices of  $(\sigma, n)$  for which  $\lambda_{\pi, m, r, \sigma, n} > 0$ . In case  $(\pi, m)$  is not a full state of  $\mathcal{A}$ , then  $\lambda_{\pi, m, r, \sigma, n}$  is defined to be zero. If  $(\pi, m)$  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  $(x, \pi, m)$  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  $(y, \sigma, n)$ , 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  $\varrho$  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  $(\pi, m)$  of  $\mathcal{A}$ , define  $p^t(\pi, m)$  to be the probability that the full state of  $\mathcal{A}$  is  $(\pi, m)$  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  $(x, \pi, m)$  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  $(\sigma, n)$  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

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

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

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  $(\pi, m)$  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  $\rho$ , the cost (or expected cost) of each  $\mathcal{A}_i$  for  $\rho$  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*  $\omega$  if, for any  $y \in \mathcal{X}$  there exists some  $x \in S$  such that  $\omega(y) = \omega(x) + d(x, y)$ . If  $\omega$  is supported by a finite set, then there is a unique minimal set  $S$  which supports  $\omega$ , which we call the *estimator support* of  $\omega$ . (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  $\omega$  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  $\omega$  and  $\omega'$  are estimators, and  $S$  is the support of  $\omega$ . 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  $\varrho$  starting at configuration  $x$  and ending at configuration  $y$ . Then, if  $\varrho$  is a request sequence, the *work function*  $\omega^\varrho : \mathcal{X} \rightarrow \mathbf{R}$  is defined by  $\omega^\varrho(x) = cost_{opt}^\varrho(s^0, \varrho, x)$ . If  $\varrho$  is a request sequence, the *offset function* is defined to be  $\bar{\omega}^\varrho = \omega^\varrho - cost_{opt}^\varrho(\varrho)$ , a zero minimum estimator. If  $\omega$  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  $\omega$  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, [11]).

**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  $(\pi, \omega)$ , 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  $offset_{\mathcal{A}}(k, r)$ , 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  $\lambda_i$  for  $i = 1, \dots, m$  such that
  - (a)  $(\omega \wedge r)(x) \geq offset_{\mathcal{A}}(k, r) + \sum_{i=1}^m \lambda_i \omega_i(x)$  for each  $x \in \mathcal{X}$ .
  - (b) For each  $i$ ,  $\mathcal{A}$  chooses  $k'$  to be  $k_i$  with probability  $\lambda_i$ .
  - (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

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

If  $S \subseteq \mathcal{X}$ , we say that a knowledge state  $(\pi, \omega)$  is *supported as a knowledge state* by  $S$  if  $\omega$  is supported by  $S$  (in the estimator sense) and  $\pi$  is supported (distributionally) by  $S$ . Note that, in this case,  $(\pi, \omega)$  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  $E(\text{cost}_{\mathcal{A}}(\varrho)) \leq C \cdot E(\text{offset}_{\mathcal{A}}(\varrho) + \omega^n(x)) + K$  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  $x \in \mathcal{X}$*

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

*Proof:* Let  $s^0 = x^0, x^1, \dots, x^n = x \in \mathcal{X}$  be the optimal service of  $\varrho$  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):  $E(\omega^t(x^t) + \text{offset}_{\mathcal{A}}^t(\varrho)) \leq E((\omega^{t-1} \wedge r^t)(x^t))$  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  $\varrho$ . Since  $\mathcal{A}$  is C-competitive as a knowledge state algorithm:

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

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

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

**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  $\varrho = r^1 \dots \dots r^n$  be a request sequence. Let  $k^1, \dots, k^n$  be the sequence of knowledge states of  $\mathcal{A}$  given the input  $\varrho$ , 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

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

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  $k$ -paging problem. (See [1, 12].)

## 4 Knowledge State Algorithms for the Paging Problem

We now consider the  $k$ -paging problem for fixed  $k \geq 2$ . The  $k$ -paging 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, which we denote as  $\mathcal{S}^k$  in the following. If the configuration of an algorithm is  $x \in \mathcal{S}^k$ , that means that the pages that constitute  $x$  are in the cache.
3. The initial configuration is the initial cache.
4. If  $x, y \in \mathcal{S}^k$ , 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  $x, y \in \mathcal{S}^k$  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 Paging Problem.** A notation system for offset functions was introduced by Koutsoupias and Papadimitriou [15, 16]; we briefly summarize that concept here. To define any offset function, suppose  $\emptyset = S_0 \subset S_1 \subset S_2 \subset \dots \subset S_k$  are sets of pages, and let  $m_i = |S_i|$ . An offset function  $\omega$  is then defined as follows:

1. We say that  $S \in \mathcal{S}^k$  is in the *support* of  $\omega$ , which we call  $\text{supp}(\omega)$ , if and only if  $|S \cap S_i| \geq i$  for all  $1 \leq i \leq k$ . We have  $\omega(S) = 0$  for all  $S \in \text{supp}(\omega)$ .
2. For any  $T \in \mathcal{S}^k$ ,  $\omega(T) = \min_{S \in \text{supp}(\omega)} \|S, T\|$ .

For convenience we also define  $L_i = S_i - S_{i-1}$ , called *layers*. We write either  $S_1|S_2|\dots|S_k|$  or equivalently  $L_1|L_2|\dots|L_k|$  to represent an offset function. From [15], we have:

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

For example for  $k = 2$ ,  $a|b|$  denotes the estimator whose support consists of just the configuration  $\{a, b\}$ , and which takes the value zero on that configuration. For  $k = 3$ ,  $b|ef|acd|$  denotes the estimator whose support consists of the configurations  $\{bea\}$ ,  $\{bfa\}$ ,  $\{bec\}$ ,  $\{bfc\}$ ,  $\{bed\}$ ,  $\{bfd\}$ , and which takes the value zero on those configurations. More generally, if the initial set of pages in the cache is  $\{s_1, s_2, \dots, s_k\}$  then  $S_i = \{s_1, s_2, \dots, s_i\}$ . Given an offset function  $S_1|S_2|\dots|S_k|$  the offset function resulting from a request to a page  $r$  is

$$\begin{array}{ll} \{r\}|S_2 + r|S_3 + r|\dots|S_k + r| & \text{if } r \notin S_k, \\ \{r\}|S_1 + r|S_2 + r|\dots|S_{i-1} + r|S_{i+1}|S_{i+2}|\dots|S_k| & \text{if } r \in S_i \text{ and } i < k, \\ \{r\}|S_1 + r|S_2 + r|\dots|S_{k-1} + r| & \text{if } r \in S_k, \end{array}$$

where the offset is increased by one in the first case. We will call requests of the first type *new* requests and requests of the second type *lazy*. We refer to  $S_k$  as the set of *active pages*. Thus, any algorithm which is based upon the offset function must use at least  $m_k - k$  bookmarks, and we say that the algorithm has to “keep track of” these pages.

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

Recall that PARTITION (introduced in [15]) is optimally competitive for the  $k$ -paging 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-paging problem, which we call  $K_2$ .

**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.

Next we give the definitions for the knowledge states and the transitions of the algorithm. In these definitions we say that two pages to are *equivalent* for a given knowledge state if they can be transposed without changing the knowledge state.

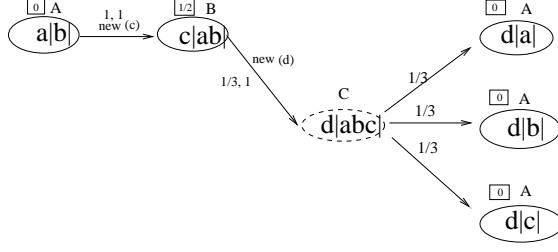


Figure 1: Schematic for the 2-Paging Knowledge State Algorithm

1. If  $a, b$  are pages, let  $A^{a,b} = (\{a, b\}, a|b|)$ . 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.*,  $B^{a,b,c} = B^{a,c,b}$ .

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

**Lemma 8**  $\Phi$  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

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

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

We first note that:

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

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,  $c$ .

$a|b|\wedge c = c|ab| + 1$ . thus  $offset = 1$ . 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  $a|bc|\wedge b = a|b|$ . Note that  $offset = 0$ , since, as functions,  $a|b| \geq a|bc|$  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  $a|bc|\wedge d \geq \frac{1}{3}d|a| + \frac{1}{3}d|b| + \frac{1}{3}d|c| + \frac{1}{3}$ , thus  $offset = \frac{1}{3}$ . 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-paging 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-paging problem never has more than two active pages, then it can have no bookmarks, hence is trackless. By Theorem 2 of [4], there is no  $\frac{3}{2}$ -competitive trackless online algorithm for the 2-paging problem.  $\square$

## 4.2 An $H_k$ -competitive Knowledge State Algorithm for $k$ -Paging with $O(k)$ Memory

We will now describe an  $H_k$ -competitive (thus optimally competitive) algorithm for the  $k$ -paging problem, which keeps track of only  $3k$  pages. We call the algorithm  $A_k$ . As mentioned above, the algorithm EQUITABLE is a knowledge state algorithm, though it was not defined in these terms. If  $\omega$  is the current offset function, the distribution  $\pi^\omega$  will be by the procedure below. (We will omit the superscript  $\omega$  from functions when the current offset function is clear from context.)

1. For any  $S \notin \text{supp}(\omega)$ ,  $\pi(S) = 0$ .
2. For any  $S \in \text{supp}(\omega)$ ,  $\pi(S) > 0$ . The following random process selects a member of  $\text{supp}(\omega)$ :
  - Initialize  $S$  to be the empty set.
  - Let  $T = S_k$ .
  - Execute the following loop until  $|S| = k$ :
    - (a) Select  $x \in T$  uniformly at random.
    - (b) Delete  $x$  from  $T$ .
    - (c) If  $S + x$  is a subset of any member of  $\text{supp}(\omega)$  let  $S = S + x$ .
  - For any  $S \in \text{supp}(\omega)$ , define  $\pi(S)$  to be the probability that  $S$  is selected in the previous procedure.

We define the function  $\Psi$  as the cost EQUITABLE incurs on a sequence of lazy requests ending when  $|S_k| = k$ . This is well defined due to the following observations given in [1]: For an offset function  $\omega$  and page  $x$ , define  $p_x^\omega$  as the probability that the page  $x$  is in the set  $S$  selected in the previous procedure. Equivalently let  $p_x = \sum_{\{S \in S^k | x \in S\}} \pi(S)$  which is the probability that  $x$  is in the cache. Note that if  $p_x > 0$  then  $x \in S_k$ .

**Observation 1** *If  $x \in L_i$  and  $y \in L_j$ , where  $i \leq j$ , then  $p_x \geq p_y$ . If  $i = j$  then  $p_x = p_y$ .*

**Observation 2** *For any offset function, all sequences of lazy requests ending when  $|S_k| = k$  have the same cost for EQUITABLE.*

For any given offset function, the algorithm  $A_k$  will maintain the same distribution as EQUITABLE. The difference between the algorithms is the forgiveness step. If  $|S_k| = 3k$  and  $r \notin S_k$  is requested then  $A_k$  moves to offset function  $\{r\}|L_1|L_2| \dots |L_{k-1}|$ .

Recall that  $m_i = |S_i|$  and define  $\Gamma = \Gamma(\omega)$  as follows:

$$\Gamma = \sum_{i=2}^k \left( \frac{m_i}{i} + H_{i-1} - H_{m_{i-1}} - 1 \right).$$

We define  $\Phi = \Psi + \Gamma$  and we wish to show that for every request

$$cost + \Delta\Psi + \Delta\Gamma \leq H_k \cdot cost_{opt}$$

where  $cost$  denotes the expected cost to the algorithm at that step and  $cost_{opt}$  is the optimal cost at that step.

In the discussion below, unprimed variables denote the values before a given request. Primed variables are the values after that request. Recall that  $m_i \geq i$ .

**Lemma 9** *On a lazy request  $r \in L_j$ ,  $cost + \Delta\Psi + \Delta\Gamma \leq H_k \cdot cost_{opt}$ .*

*Proof:* Since  $\Psi$  is the cost for EQUITABLE to serve a lazy sequence of requests ending in a cone, on a lazy request  $cost + \Delta\Psi = 0$ . For a lazy request  $cost_{opt} = 0$  by definition. So it suffices to show that  $\Delta\Gamma < 0$ . We have

$$\begin{aligned} \Delta\Gamma &= \sum_{i=2}^k \left( \frac{m'_i}{i} - H_{m'_{i-1}} - \frac{m_i}{i} + H_{m_{i-1}} \right) \\ &= \sum_{i=2}^j \left( \frac{m_{i-1} + 1 - m_i}{i} - H_{m_{i-1}} + H_{m_{i-1}} \right) \\ &= \sum_{i=2}^j \left( \frac{m_{i-1} + 1 - m_i}{i} + \sum_{\ell=m_{i-1}+1}^{m_i-1} \frac{1}{\ell} \right) \\ &\leq \sum_{i=2}^j \left( \frac{m_{i-1} + 1 - m_i}{i} + \sum_{\ell=m_{i-1}+1}^{m_i-1} \frac{1}{i} \right) \\ &= 0. \end{aligned}$$

□

**Lemma 10** *On a request  $r \notin S_k$ , if forgiveness does not occur, then  $\Delta\Psi \leq \sum_{i=2}^k \frac{1}{m_i}$ .*

*Proof:* If  $k = 1$  then  $\Psi = \Psi' = 0$  which shows that the lemma is true for  $k = 1$ . Assume  $k > 1$  and that the lemma is true for  $k - 1$ .

For every page  $s \neq r$ ,  $p_s \geq p'_s$ . Since  $p_r = 0$  and  $p'_r = 1$ ,  $\sum_{s \in S_k} p_s - p'_s = 1$ , there must be an item  $x \in S_k$  for which  $p_x - p'_x \leq 1/m_k$ . Without loss of generality,  $x \notin S_1$  because every item  $y \in S_2$  will have  $p_y - p'_y \leq p_x - p'_x$ . Let this page  $x \in S_j$  be the first item in the lazy request sequence which defines  $\Psi$  and  $\Psi'$ . Define the following offset functions:

$$\begin{aligned}
\omega &= S_1|S_2|\dots|S_k| \\
\omega' &= r|S_2+r|\dots|S_k+r| \\
\omega \wedge x &= x|S_1+x|S_2+x|\dots|S_{j-1}+x|S_{j+1}|\dots|S_k| \\
\omega' \wedge x &= x|xr|S_2+r+x|\dots|S_{j-1}+r+x|S_{j+1}+r|\dots|S_k+r| \\
\omega_{dropx} &= S_1|S_2|\dots|S_{j-1}|S_{j+1}|\dots|S_k| \\
\omega'_{dropx} &= r|S_2+r|\dots|S_{j-1}+r|S_{j+1}+r|\dots|S_k+r|.
\end{aligned}$$

Now we notice that

$$\begin{aligned}
\Delta\Psi &\leq \frac{1}{m_k} + \Psi(\omega' \wedge x) - \Psi(\omega \wedge x) \\
&= \frac{1}{m_k} + \Psi(\omega'_{dropx}) - \Psi(\omega_{dropx}) \\
&\leq \frac{1}{m_k} + \sum_{i=2}^{k-1} \frac{1}{m_i} \\
&= \sum_{i=2}^k \frac{1}{m_i}
\end{aligned}$$

where the third line follows from the inductive hypothesis.  $\square$

**Lemma 11** *On a request  $r \notin S_k$ , if forgiveness does not occur, then*

$$cost + \Delta\Psi + \Delta\Gamma \leq H_k \cdot cost_{opt}.$$

*Proof:* Since  $r \notin S_k$ ,  $m'_i = m_i + 1$ . Given lemma 10, it follows that

$$\begin{aligned}
cost + \Delta\Psi + \Delta\Gamma &\leq 1 + \sum_{i=2}^k \frac{1}{m_i} + \sum_{i=2}^k \left( \frac{m'_i}{i} - H_{m'_i-1} - \frac{m_i}{i} + H_{m_i-1} \right) \\
&= 1 + \sum_{i=2}^k \frac{1}{m_i} + \sum_{i=2}^k \left( \frac{m_i+1}{i} - H_{m_i} - \frac{m_i}{i} + H_{m_i-1} \right) \\
&= \sum_{i=1}^k \frac{1}{i} \\
&= H_k \cdot cost_{opt}.
\end{aligned}$$

$\square$

**Lemma 12** *On a request outside  $S_k$  when forgiveness occurs.*

$$cost + \Delta\Psi + \Delta\Gamma \leq 0.$$

*Proof:* A forgiveness step occurs when  $S_k = 3k$  and a page  $r \notin S_k$  is requested. The forgiveness step can be seen as placing  $r$  in  $L_k$  and then requesting  $r$ . We can easily compute  $\Delta\Gamma$  during a forgiveness step. However, it is easier to compute  $cost + \Delta\Psi$  when  $r$  is added to  $L_k$  and on the request separately.

When  $r$  is placed into  $L_k$  the distribution must be adjusted and the lazy potential changes. The transportation cost necessary to adjust the distribution is  $p'_r$ . Since the cost on all lazy sequences is the same, we can compute the change in potential by considering the sequence which begins with a page  $x \in L_k$ . From Observation 1  $cost + \Delta\Psi = p_x - p'_x + p'_r = p_x \leq k/m_k$ .

When  $r$  is requested  $cost + \Delta\Psi = 0$  because  $\Psi$  is the lazy potential. So it suffices to show that  $k/m_k + \Delta\Gamma$  is no more than 0. We have

$$\begin{aligned}
cost + \Delta\Psi + \Delta\Gamma &\leq \frac{k}{m_k} + \sum_{i=2}^k \left( \frac{m'_i}{i} - H_{m'_i-1} - \frac{m_i}{i} + H_{m_i-1} \right) \\
&= \frac{1}{3} + \sum_{i=2}^k \left( \frac{m_{i-1} + 1 - m_i}{i} - H_{m_{i-1}} + H_{m_i-1} \right) \\
&\leq \frac{1}{3} + \left( \frac{k - m_k}{k} - H_{k-1} + H_{m_k-1} \right) \\
&\leq -\frac{5}{3} + H_{3k-1} - H_{k-1} \\
&\leq 0.
\end{aligned}$$

The second inequality above holds because the  $\Delta\Gamma$  is decreasing in  $m_\ell$  for all  $\ell < k$ . So the worst case occurs when  $m_\ell = \ell$ .  $\square$

**Theorem 5**  $A_k$  is an  $H_k$ -competitive,  $O(k)$  memory, randomized algorithm for the  $k$ -paging problem.

*Proof:* Since we perform a forgiveness step when the set of active pages has size  $3k$  and a new page is requested, the set of pages we keep track of is never greater than  $3k$ . Lemmas 9, 11, and 12 show that  $cost + \Delta\Psi + \Delta\Gamma \leq H_k \cdot cost_{opt}$  on every request type. Noting that  $\Psi + \Gamma$  is initially 0 and never negative, summing over every request shows that  $A_k(\varrho) \leq H_k \cdot OPT(\varrho)$  for any request sequence  $\varrho$ .  $\square$

## 5 Summary of Further Results

One of the the most challenging problems in online algorithms is to determine the exact randomized competitiveness of the  $k$ -server problem, that is, the minimum competitiveness of any randomized online algorithm for the server problem. Even in the case  $k = 2$  it is not known whether its competitiveness is lower than 2, the known value of the deterministic competitiveness. This is surprising, since it seems intuitive that randomization should help. For the randomized 2-server problem we note that for the special case of the line, Bartal *et al.* [2] have given a randomized algorithm with a competitive ratio of better than 2. Unfortunately, the approach is specific to the line, using methods which do not appear to generalize to all metric spaces. Recently in [8] we have designed a knowledge state algorithm with a competitive ratio of  $\frac{19}{12}$  over Cross Polytope Spaces, and proved it is optimal against the oblivious adversary. Cross Polytope Spaces have been studied extensively starting as early as the 19<sup>th</sup> century; see Schläfli [17], as well as Figure 2. They are part of a larger category  $M_{2,4}$ , consisting of all metric spaces such that







Regular Polytope Family	Graph Class	Metric Space Class	3-d	4-d
Simplices	Complete Graphs	Uniform Spaces		
Cross Polytopes = Orthoplices	Circulant Graphs $C_{1, \dots, n-1}(2n)$	$M_{24}$		
Hypercubes	Hypercubes	Hamming Metric Spaces		

Figure 2: The Class  $M_{2,4}$

- all distances are 1 or 2,
- $d(x, y) + d(y, z) + d(z, x) \leq 4$ .

We note that paging can be modeled as a server problem in uniform spaces. Thus  $M_{2,4}$  is “one step up” from uniform spaces and further work would focus on  $M_{3,6}$  and so forth.

A result motivated by applications was in regards to multiprocessor caching systems [7]. 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. In [7] we have given a  $\frac{4}{3}$ -competitive randomized knowledge state algorithm for this problem for cache size of 2. We have also proved a matching lower bound, thus this online algorithm is best possible. As well a lower bound of  $\frac{3}{2}$  on the competitiveness for larger cache sizes was shown.

In this paper we have given an  $H_k$ -competitive randomized online algorithm for the  $k$ -paging problem which keeps track of only  $3k$  pages. For large  $k$ , Lemma 12 can be improved to  $\alpha k$  where  $\alpha \approx 2.2572$  satisfies  $\alpha^2 - \alpha - \alpha \log(\alpha) = 1$ . For  $k = 3$ , the technique used in this paper can be used to calculate that 7 pages suffice. Additionally for  $k = 3$  we have been able to construct an  $H_k$ -competitive randomized online algorithm, which uses only 6 active pages. We emphasize that we have not proven  $3k$  to be the minimum number of pages needed for any particular  $k$ . In fact, we conjecture that a stronger result holds than the upper bound from this paper:

**Conjecture 1** *There exists a randomized online algorithm for paging which is  $H_k$ -competitive and uses  $o(k)$  bookmarks (i.e.  $k + o(k)$  memory).*

## 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. In *Proc. 6th European Symp. on Algorithms (ESA)*, Lecture Notes in Computer Science, pages 247–258. Springer, 1998.
- [3] Yair Bartal, Marek Chrobak, and Lawrence L. Larmore. A randomized algorithm for two servers on the line. *Information and Computation*, 158:53–69, 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] Wolfgang Bein and Lawrence L. Larmore. Trackless and limited bookmark algorithms for paging. *SIGACT News*, 35:40–49, 2004.
- [7] Wolfgang Bein, Lawrence L. Larmore, and Rüdiger Reischuk. Knowledge states for the caching problem in shared memory multiprocessor systems. *International Journal of Foundations of Computer Science*, 2009. In print.
- [8] Wolfgang W. Bein, Kazuo Iwama, Jun Kawahara, Lawrence L. Larmore, and James A. Oravec. A randomized algorithm for two servers in cross polytope spaces. In *Proceedings 5th Workshop on Approximation and Online Algorithms (WAOA) Eilat, Israel, October 11-12, 2007*, volume 4927 of *Lecture Notes in Computer Science*, pages 246–259. Springer, 2008.
- [9] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- [10] Marek Chrobak, Elias Koutsoupias, and John Noga. More on randomized on-line algorithms for caching. *Theoretical Computer Science*, 290:1997–2008, 2003.
- [11] 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.
- [12] Marek Chrobak and Lawrence L. Larmore. Generosity helps or an 11-competitive algorithm for three servers. *Journal of Algorithms*, 16:234–263, 1994.
- [13] 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.
- [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] Elias Koutsoupias and Christos Papadimitriou. Beyond competitive analysis. In *Proc. 35th Symp. Foundations of Computer Science (FOCS)*, pages 394–400. IEEE, 1994.

- [16] Elias Koutsoupias and Christos Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30:300–317, 2000.
- [17] Ludwig Schläfli. *Theorie der vielfachen Kontinuität*. Birkhäuser, Basel, 1857.