Knowledge States: A Tool for Randomized Online Algorithms

Wolfgang Bein *
Center for the Advanced Study of Algorithms
School of Computer Science
University of Nevada
Las Vegas, NV 89154, U.S.A.
bein@cs.unlv.edu

Lawrence L. Larmore †
Center for the Advanced Study of Algorithms
School of Computer Science
University of Nevada
Las Vegas, NV 89154, U.S.A.
larmore@cs.unlv.edu

Rüdiger Reischuk
Institut für Theoretische Informatik
Universität zu Lübeck
Ratzeburger Allee 160
D-23538 Lübeck, Germany
reischuk@tcs.uni-luebeck.de

Abstract

We introduce the concept of knowledge states; many well-known randomized online 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 an estimator 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.

1 Motivation and Background

In online computation, an algorithm has to make decisions without knowledge of future inputs. Online problems occur naturally across the entire field of computer science with applications in many areas such as robotics, resource allocation in operating systems, network routing. Online algorithms are analyzed in terms of *competitiveness*, a measure of performance that compares the decision made online with the optimal offline solution – the solution obtainable if the entire input sequence is known at the beginning of the computation, where the lowest possible competitiveness is best. The area of online competitive algorithms has progressed to a certain level of maturity, and its relevance within computer science is well established, see *e.g.* the Borodin El-Yaniv book [8].

For many problems the use of randomization can improve algorithms. In the area of online algorithms there can be a substantial improvement in competitiveness (as well as a decrease in memory requirements) if randomization is allowed. Yet, many of these improvements rely on adhoc ideas, which work for one problem but not the other. It is worthwhile to understand the effects of online randomization in more general terms. In this paper we introduce a new method for constructing randomized online algorithms, which we call the *knowledge state* model. The model is introduced and fully described in this paper and we note that a number of online algorithms are implicitly consistent with the model although not in its full power.

Throughout this paper we will make reference to two classical problems which have been studied extensively in online algorithms: the k-server problem and the k-cache problem. (See [8] for an extensive survey.) In the first problem, there are k mobile identical servers in a metric space $\mathcal{M}$. At any time, a point $r \in M$ can be "requested," and must be "served" by moving one of the k servers to the point r. The cost of that service is defined to be the distance the server is moved. For example, the randomized algorithm RANDOM_SLACK [11] is, in fact, an extremely sim-

*Research supported by NSF grant CCR-0312093. Research conducted while on sabbatical from the University of Nevada, Las Vegas. Sabbatical support from UNLV is acknowledged.
†Research supported by NSF grant CCR-0312093.

ple 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.[1] It is not known whether there is a randomized online algorithm for the 2-server problem with competitiveness lower than 2, the known value of the deterministic competitiveness. This is surprising and it is quite intuitive that a "better than 2-competitive" algorithm should exist. In fact, breaking through this "2-competitive barrier" was ranked by the participants of the 2002 Dagstuhl Workshop [2] to be one of the three most important outstanding online problems.

The other problem is the k cache problem. Here we consider a two-level memory system, consisting of fast memory (the *cache*), which can hold k memory units (commonly called *pages*,) and an area of slow memory capable of holding a much larger number of pages. In the most basic model, if a page in slow memory is needed in fast memory this is called a *page request*. Such a request causes a *hit* if the page is already in the cache at the time of the request. But in the case of a *miss* (*i.e.* when the page is not in the cache,) the requested page must be brought into the cache – we assume unit cost for such a move – while a page in the cache must be evicted to make room for the new page. An online paging algorithm must make decisions about such evictions as the request sequence is presented to the algorithm. The algorithm EQUITABLE [1], which 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 [12], that uses the full information contained in the work function, and hence requires unlimited memory as the length of the request sequence grows. It is still an open question, whether there exists an optimally competitive order $O(k)$ bookmark randomized algorithm for the k-cache problem.[2]

The main point of this paper is to 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, to illustrate the method, it is applied to the the 2 cache problem. We give an order 2 knowledge state algorithm which is provably H_2-competitive and which illustrates the knowledge state technique in a simple way. Sections 5 summarizes more general results for the cache problem, results for the server problem, as well as other work in progress.

[1] We also note that RANDOM_SLACK is *trackless* see the recent ACM SIGACT column [6] for a summary of tracklessness; see also [4, 5, 3, 7].

[2] An affirmative answer to this question would settle an open problem listed in [8].

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* $\varrho = r^1, \ldots r^n$, an algorithm must choose a sequence of states $x^1, \ldots x^n$, 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

$(A, B, cost, \alpha, \beta)$ 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 $cost(\gamma) = \sum_{a \in A} \sum_{b \in B} \gamma(a, b) cost(a, b)$, and γ 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 [8]). 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 $cost(\pi, r, \pi')$ to be the minimum transportation cost of the transportation problem $(S, S', cost^r, \pi, \pi')$, where $cost^r = cost(\ , r, \ ) : \mathcal{X} \times \mathcal{X} \to \mathbf{R}$.

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 $cost_{\mathcal{A}}(\varrho) = \sum_{t=1}^{n} cost(\pi^{t-1}, r^t, \pi^t)$.

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 [8].)

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 $\sum_{i=1}^{m} \lambda_i cost(\pi, r, \pi_i)$ 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 $\bar{S} \subseteq \mathcal{X}$ 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 $cost_{\mathcal{A}}(k, r) = cost(\pi, r, \bar{\pi})$.

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 $cost_{\mathcal{A}}(\varrho) = \sum_{t=1}^{n} cost_{\mathcal{A}}(k^{t-1}, r^t)$.

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, $E(cost_{\mathcal{A}'}(\varrho)) = E(cost_{\mathcal{A}}(\varrho))$.*

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

$$(\mathcal{X}, \mathcal{X}, cost^r, \pi, \bar{\pi}_{\pi,m,r}), \text{ where } cost^r(x, y) = cost(x, r, y).$$

Thus

$$\pi(x) = \sum_{y \in \mathcal{X}} \gamma_{\pi,m,r}(x, y) \text{ for } x \in \mathcal{X};$$

$$\bar{\pi}_{\pi,m,r}(y) = \sum_{x \in \mathcal{X}} \gamma_{\pi,m,r}(x, y) \text{ for } y \in \mathcal{X};$$

$$cost_{\mathcal{A}}(\pi, m, r) = \sum_{x \in \mathcal{X}, y \in \mathcal{X}} \gamma_{\pi,m,r}(x, y) cost(x, r, y).$$

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

$$E(cost_{\tilde{\mathcal{A}}}(\varrho)) = E(cost_{\mathcal{A}}(\varrho)).$$

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

$$\bar{\pi}_{\pi,m,r}(y) = \sum_{x \in \mathcal{X}} \gamma_{\pi,m,r}(x, y) \text{ for all } y \in \mathcal{X},$$

and we obtain

$$q^t(y, \sigma, n) = \sum_{(x,\pi,m) \in \mathcal{X} \times \Pi \times \mathcal{M}} q^{t-1}(x, \pi, m) \Lambda_{x,\pi,m,r,y,\sigma,n}$$

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

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

$$= \sigma(y) \cdot \sum_{\substack{(\pi,m)\in\Pi\times\mathcal{M}\\ \bar{\pi}_{\pi,m,r}(y)>0}} \Big(p^{t-1}(\pi,m)\cdot\lambda_{\pi,m,r,\sigma,n} \cdot \sum_{x\in\mathcal{X}} \frac{\gamma_{\pi,m,r}(x,y)}{\bar{\pi}_{\pi,m,r}(y)} \Big)$$

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

$$= \sigma(y) \cdot \sum_{(\pi,m)\in\Pi\times\mathcal{M}} p^{t-1}(\pi,m)\cdot\lambda_{\pi,m,r,\sigma,n}$$

$$= \sigma(y)\cdot p^t(\sigma,n)$$

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

$$E(cost^t_{\bar{\mathcal{A}}}) = \sum_{\substack{\pi,\sigma\in\Pi\\ m,n\in\mathcal{M}\\ x,y\in\mathcal{X}}} q^{t-1}(x,\pi,m)\cdot\Lambda_{x,\pi,m,r,y,\sigma,n}\cdot cost(x,r,y)$$

$$= \sum_{\substack{\pi,\sigma\in\Pi\\ m,n\in\mathcal{M}\\ x,y\in\mathcal{X}\\ \pi(x)>0,\sigma(y)>0}} \Big(p^{t-1}(\pi,m)\cdot\pi(x) \cdot \frac{\gamma_{\pi,m,r}(x,y)\cdot\sigma(y)\cdot\lambda_{\pi,m,r,\sigma,n}}{\pi(x)\cdot\bar{\pi}_{\pi,m,r}(y)}\cdot cost(x,r,y) \Big)$$

$$= \sum_{\substack{\pi\in\Pi\\ m\in\mathcal{M}\\ x,y\in\mathcal{X}}} \Big(p^{t-1}(\pi,m)\cdot\gamma_{\pi,m,r}(x,y)\cdot cost(x,r,y) \cdot \sum_{\sigma\in\Pi,n\in\mathcal{M},\sigma(y)>0} \frac{\lambda_{\pi,m,r,\sigma,n}\cdot\sigma(y)}{\bar{\pi}_{\pi,m,r}(y)} \Big)$$

$$= \sum_{\substack{\pi\in\Pi\\ m\in\mathcal{M}\\ x,y\in\mathcal{X}}} p^{t-1}(\pi,m)\cdot\gamma_{\pi,m,r}(x,y)\cdot cost(x,r,y)$$

$$= \sum_{\substack{\pi\in\Pi\\ m\in\mathcal{M}}} \Big(p^{t-1}(\pi,m)\cdot\sum_{x,y\in\mathcal{X}}\gamma_{\pi,m,r}(x,y)\cdot cost(x,r,y)\Big)$$

$$= \sum_{\pi\in\Pi,m\in\mathcal{M}} p^{t-1}(\pi,m)\cdot cost_{\mathcal{A}}(\pi,r,\bar{\pi}_{\pi,m,r})$$

$$= \sum_{\pi\in\Pi,m\in\mathcal{M}} p^{t-1}(\pi,m)\cdot cost_{\mathcal{A}}(\pi,m,r) \quad = \quad E(cost^t_{\mathcal{A}})$$

and we are done. □

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. □

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 $\omega : \mathcal{X} \to \mathbf{R}$ is *Lipschitz* if $\omega(y) \le \omega(x) + d(x,y)$ for all $x, y \in \mathcal{X}$. An *estimator* is a non-negative Lipschitz function $\mathcal{X} \to \mathbf{R}$. 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 $\omega(x) \ge \omega'(x)$ for all $x \in \mathcal{X}$ if and only if $\omega(y) \ge \omega'(y)$ for all $y \in S$.*

Proof: One direction of the proof is trivial. Suppose

$$\omega(x) < \omega'(x) \text{ and } \omega(y) \ge \omega'(y) \text{ for all } y \in S.$$

Then there exists $y \in S$ such that $\omega(x) = \omega(y) + d(y,x)$. It follows that

$$\omega(y) = \omega(x) - d(y,x) < \omega'(x) - d(y,x) \le \omega'(y),$$

contradiction. □

An example of an estimator is the *work function* of a request sequence. If $x, y \in \mathcal{X}$, we write $cost^{\varrho}_{opt}(x,y)$ 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* $\omega^{\varrho} : \mathcal{X} \to \mathbf{R}$ 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* $(\omega \wedge r)(y) = \min_{x \in S} \{\omega(x) + \mathit{cost}(x, r, y)\}$.

Proof: Trivially,

$$(\omega \wedge r)(y) \leq \min_{x \in S} \{\omega(x) + \mathit{cost}(x, r, y)\}.$$

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

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

and we are done. □

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, [9]).

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 $\omega : \mathcal{X} \to \mathbf{R}$ 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 $\mathit{adjust}_{\mathcal{A}}(k, r)$, 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) $(\omega \wedge r)(x) \geq \mathit{adjust}_{\mathcal{A}}(k, r) + \sum_{i=1}^{m} \lambda_i \omega_i(x)$ 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 $\mathit{cost}_{\mathcal{A}}(k, r) = \mathit{cost}(\pi, r, \bar{\pi})$. (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

$$\begin{aligned}\mathit{cost}^t_{\mathcal{A}}(\varrho) &= \mathit{cost}_{\mathcal{A}}\big(k^{t-1}, r^t\big) \\ \mathit{adjust}^t_{\mathcal{A}}(\varrho) &= \mathit{adjust}_{\mathcal{A}}\big(k^{t-1}, r^t\big)\end{aligned}$$

and

$$\begin{aligned}\mathit{cost}_{\mathcal{A}}(\varrho) &= \sum_{t=1}^{n} \mathit{cost}^t_{\mathcal{A}}(\varrho) \\ \mathit{adjust}_{\mathcal{A}}(\varrho) &= \sum_{t=1}^{n} \mathit{adjust}^t_{\mathcal{A}}(\varrho).\end{aligned}$$

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

$$E(\mathit{cost}_{\mathcal{A}}(\varrho)) \leq C \cdot E(\mathit{adjust}_{\mathcal{A}}(\varrho) + \omega^n(x)) + K$$

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 $x \in \mathcal{X}$*

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

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:

$$\sum_{t=1}^{n} \mathit{cost}(x^{t-1}, r^t, x^t) = \mathit{cost}_{\mathit{opt}}(s^0, x).$$

By (2a):

$$E\big(\omega^t(x^t) + \mathit{adjust}^t_{\mathcal{A}}(\varrho)\big) \leq E\big(\big(\omega^{t-1} \wedge r^t\big)(x^t)\big) \text{ for all } t.$$

By definition:

$$E\big(\big(\omega^{t-1} \wedge r^t\big)(x^t)\big) \leq E\big(\omega^{t-1}(x^{t-1})\big) + \mathit{cost}(x^{t-1}, r^t, x^t)$$

for all t.

Summing the inequalities over all t, and adding to the equation, we obtain the result. □

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:

$$E(\mathit{cost}_{\mathcal{A}}(\varrho)) \leq C \cdot E(\mathit{adjust}_{\mathcal{A}}(\varrho)) + C \cdot E(\omega^n(x^n))) + K$$

$$E(\mathit{adjust}_{\mathcal{A}}(\varrho) + \omega^n(x^n)) \leq \mathit{cost}_{\mathit{opt}}(\varrho) \quad \text{(by lemma 5)}$$

We obtain:

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

□

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

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

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 $\varrho = r^1 \dots, \dots r^n$ 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 $\Phi^t_{\mathcal{A}} = \Phi_{\mathcal{A}}(k^t)$, a random variable for each t. Note that $\Phi^0_{\mathcal{A}}$ is a constant. Let $\Delta^t\Phi_{\mathcal{A}} = \Delta\Phi_{\mathcal{A}}(k^{t-1}, r^t)$. Note that $E(\Delta^t\Phi_{\mathcal{A}}) = E(\Phi^t_{\mathcal{A}} - \Phi^{t-1}_{\mathcal{A}})$. Let $x \in \mathcal{X}$ be the configuration of the optimal algorithm after n steps. Then

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

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

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, 10].)

4 A $\frac{3}{2}$-Competitive Knowledge State Algorithm for the 2-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

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

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 [12], for offset functions for the k-cache problem, which we call the *bar notation*.[3] 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 $i = 1, \ldots k$, 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 [12], 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.*

Recall that PARTITION (introduced in [12]) 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.

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.

[3] The notation of [12] differs slightly from that given here, although it is based on the same concept.

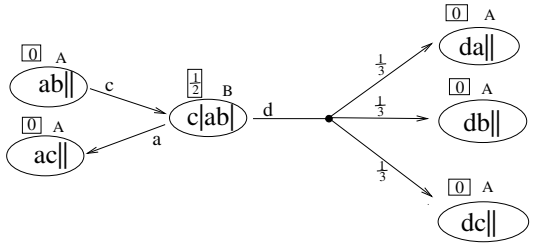

Figure 1. Schematic for the 2-Cache Knowledge State Algorithm

In the definitions given below, we say that two pages to 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 $B^{a,b,c} = \left(\frac{1}{2}\{a, b\} + \frac{1}{2}\{a, c\}, a|bc|\right)$, 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 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

$$cost + \Delta\Phi \quad \leq \quad \tfrac{3}{2} adjust \tag{1}$$

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

We first note that:

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

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. □

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:

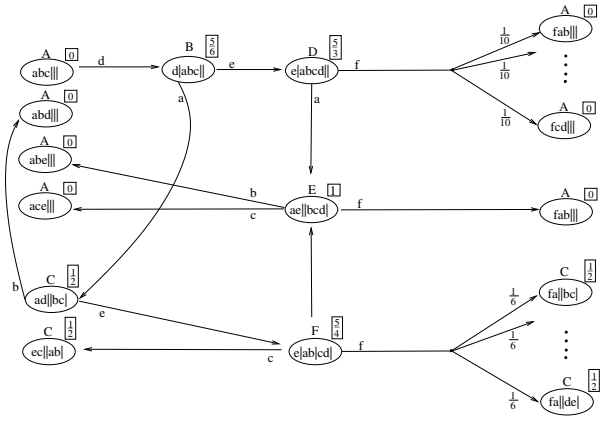

Figure 2. Schematic for the 3-Cache Knowledge State Algorithm

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 [4], there is no $\frac{3}{2}$-competitive trackless online algorithm for the 2-cache problem. □

5 Extensions and Open Problems

We have also defined 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, algorithm K_3 has six knowledge states. The number of active pages, *i.e.*, pages contained in a support configuration, is never more than five. We will give the details of the algorithm in the full journal version of this paper, but we show a schematic of the algorithm in Figure 2.

Ongoing work suggests that the technique can be extended to general values of k and we conjecture an H_k-competitive algorithm for the k-cache problem with memory requirement $O(k)$ based on knowledge states.

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 which 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 [13].) We mention that we have constructed 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 $\frac{19}{12}$.

References

[1] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234:203–218, 2000.

[2] Susanne Albers, Amos Fiat, and Gerhard Woeginger (Eds.). Online algorithms, Dagstuhl Seminar 02271. Technical Report 347, Schloss Dagstuhl, 2002. published electronically under http://www.dagstuhl.de/02271/Report/.

[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. In *Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks*, pages 307 – 312. IEEE, 2004.

[8] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[9] 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.

[10] Marek Chrobak and Lawrence L. Larmore. Generosity helps or an 11-competitive algorithm for three servers. *Journal of Algorithms*, 16:234–263, 1994.

[11] 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.

[12] Elias Koutsoupias and Christos Papadimitriou. Beyond competitive analysis. In *Proc. 35th Symp. Foundations of Computer Science (FOCS)*, pages 394–400. IEEE, 1994.

[13] Ludwig Schläfli. *Theorie der vielfachen Kontinuität*. Birkhäuser, Basel, 1857.