Derandomizing Online Algorithms
(Extended Abstract)

by

S. Ben-David and E. Dichterman

Technical Report #691
October 1991
Derandomizing Online Algorithms (Extended Abstract)

Shai Ben-David*  Eli Dichterman†

Dept. of Computer Science
Technion, Haifa 32000, Israel

July 1991

Abstract

The aim of this work is to find methods for reducing the amount of randomness needed for implementing online algorithms, without hurting their efficiency. We call an online algorithm pseudo-randomized if it uses a bounded number of random bits, regardless of its input length. We show that such algorithms can be effectively transformed into deterministic ones with a bounded increase of their competitive ratio. The main contribution of this work is the presentation of two constructible techniques for transforming randomized online algorithms into pseudo-randomized algorithms. The first technique replaces a given randomized algorithm by another that approximates its behavior. We show that such approximations result in only a bounded reduction in performance (depending upon the quality of the approximation) and that, in many cases, good approximations can be achieved by pseudo-randomized algorithms. The second technique is recycling of random bits. We introduce a notion of phased algorithms - algorithms that allow dividing any input sequence into phases such that for each phase they use only a limited amount of random bits. We show that phased algorithms can use the same random bits for each of the input's phases, without any loss in their performance. We apply this technique to present a pseudo-randomized version of the marking algorithm for paging, which was presented by Fiat et al. [FKL+88].

1 Introduction

In recent years, randomized algorithms have become central in diverse fields of computer science, such as computational number theory, computational geometry, parallel and distributed computing, and cryptography. Nevertheless, there are several obvious drawbacks to the use of randomized algorithms; Truly independent random bits are hard to get. In fact, random bits should be considered as an expensive computational resource. A second consideration that should be taken into account is that the success of a randomized algorithm is only on the average, there is usually no guarantee that on a given single run such an algorithm will demonstrate its merit (be it efficiency or even correctness). Therefore, much effort is being invested in minimization of the number of random bits used by randomized algorithms [San87, IZ89], or even transforming them into deterministic
algorithms. This line of research has led to the concepts of pseudo-random numbers [KR88] and the construction of small probability spaces for randomized algorithms [NN90]. In this work we consider the possibility of reducing the amount of randomness needed for the implementation of (weakly) competitive randomized online algorithms.

An online problem is a scenario in which one has to perform a sequence of tasks one at a time, without any information about the future tasks. Typical examples of such problems are caching problems (such as managing two-level paged memory systems [RS89]), scheduling problems (such as planning the motion of the heads of a disk drive [CCL85]) and data structure problems (such as heuristics for linear search [ST85]).

Several abstract models for online problems have been formulated during the last few years, among them are task systems [BLS87], k-server problems [MMS90] and request-answer games [BDBK+90]. In this work we focus our attention on the k-server problem, just the same many of the ideas we discuss transcend to other models.

As in other fields of computer science, it was soon realized that randomization could be helpful for improving the performance of online algorithms [CDRS90, FKL+88]. There are several different ways to define the efficiency of a randomized online algorithm. Ben-David et al. [BDBK+90] discuss the relative strength of different definitions of the competitive ratio of randomized online algorithms. They show that if a randomized algorithm is competitive in a strong enough way then it can be effectively transformed into a competitive deterministic algorithm. These ideas have been proved useful in constructing some of the most efficient online algorithms known [FRR90, Gro91]. We focus on the measure that renders most power to randomization - the oblivious adversary model. In this model the sequence of requests is determined independently of the values of random bits used by the online algorithm. So far there are no known methods for derandomizing algorithms that are competitive only against an oblivious adversary, without risking the loss of their competitiveness.

The importance of randomization for the online problem stems from two sources. From a purely theoretical point of view there is an inherent gap between the possible efficiency of randomized algorithms and that of deterministic ones. Manasse, McGeoch and Sleator [MMS90] have proved a lower bound of $k$ on the competitive ratio of any k-server deterministic online algorithm. On the other hand the best known lower bound for randomized algorithms is $H_k$ [BLS87, FKL+88] and there exist algorithms meeting this bound [FKL+88, MS89]. The interest in randomized online algorithms has also a practical motivation. Our current experience is that it is much easier to achieve efficiency in the realm of randomized algorithms. The best known general purpose deterministic online algorithm, Grove's deterministic version of HARMONIC [Gro91], achieves a ratio of $O(2^{2k})$, while the randomized version of the same algorithm is $k$-competitive (only for resistive graphs though) [CDRS90]. Furthermore, as we've already mentioned, both Grove's algorithm and Fiat et al. [FRR90] are a deterministic transformation of an originally randomized algorithm.

So far, any randomized algorithm that beats the best competitive ratio obtained by known deterministic algorithms uses unbounded amount of randomness - the number of random bits needed for its implementation grows unboundedly with the length of the request sequence it is serving.

The main question we address in this paper is the existence of efficient randomized online algorithms that can be implemented using a bounded number of random bits (to serve request sequences of any length). We call such algorithms almost-derandomized (and will appreciate any suggestion for a better name). Our main results show how to transform certain types of randomized algorithms into almost-derandomized ones without losing much in their competitive ratio.

The paper is organized as follows. In Section 2 we formally define the notions of the k-server problem, randomized online algorithms and the competitive ratio. In Section 3 we present two
parameters quantifying the amount of nondeterminism associated with a randomized algorithm. The first parameter is roughly the number of random bits the algorithm uses. The second parameter counts the number of response sequences available to an algorithm for serving a given request sequence. We analyze the relationship between these two parameters. In Section 4 we introduce a technique of approximating one algorithm by another and show the existence of a tradeoff between the loss of competitive ratio and the reduction in needed randomness, obtained through replacing an algorithm by such an approximation. Finally, in Section 5 we offer a second line of attack. We introduce a notion of phased algorithms and show that such algorithms can be easily transformed into pseudo-randomized algorithms, which perform just as well as the original ones, by recycling the random bits they are using. We show that the known marking algorithm \cite{FKL+88} is a phased algorithm.

2 Definitions and Notations

In this section we define formally the basic notions of the k-server problem, randomized online algorithm, and the competitive ratio. The model consists of a metric space \((V, d)\), where \(V\) is a set of points and \(d\) is a metric on \(V\). \(d : V \times V \rightarrow \mathbb{R}^+\) is symmetric, satisfies the triangle inequality, and \(d(v, v) = 0\) for every \(v \in V\). The space is called uniform iff \(d(u, v) = 1\) for every \(u \neq v\). \(k\) mobile and identical servers are located in \(k\) points of the space. We call a set of points occupied with servers a location. A request is a point \(v \in V\). A request \(v\) can be satisfied by occupying \(v\) with a server. A request sequence \(\sigma\) is a finite sequence of requests. The length of the sequence is denoted by \(|\sigma|\), the \(i\)-th request in the sequence is denoted by \(\sigma(i)\) and the subsequence \(\sigma(i), \ldots, \sigma(j)\) is denoted by \(\sigma(i, j)\). In order to simplify the notations, we also let the request sequence to set the initial location of the servers. A service sequence \(p\) for the request sequence \(\sigma\) is a sequence of \(|\sigma|\) many locations which satisfies \(\sigma(i) \in p(i)\) for every \(1 \leq i \leq |\sigma|\). The set of all the service sequences for the the request sequence \(\sigma\) is denoted by \(S(\sigma)\). The cost of moving from location \(l_1\) to location \(l_2\) is the minimal distances that the servers have to move from \(l_1\) to \(l_2\). The cost of a service sequence \(p\), denoted by \(\text{Cost}(p)\), is the sum of costs of moving from each location to the next.

The optimal cost of serving the request sequence \(\sigma\) is \(C_{OPT}(\sigma) = \min_{\rho \in S(\sigma)} \text{Cost}(\rho)\). Obviously, a service sequence \(\rho\) with the optimal cost for the request sequence \(\sigma\) can be computed once \(\sigma\) is known entirely, and this is the reason for attributing the optimal service sequence and its cost to an optimal offline algorithm.

While offline algorithm may base its responses on knowing of the entire request sequence, an online algorithm is introduced with the requests one at a time, and has to satisfy them online before seeing subsequent requests. We allow an online algorithm to use memory in order to record information while serving the requests. It follows that an online algorithm is a function that maps a configuration (consists of the memory state of the algorithm and the location of the serves) and a request, to the next configuration.

An online algorithm is randomized if it can choose its responses randomly. We model these random choices using an infinite string of random bits, represented by a real number \(r \in \mathbb{R}[0, 1)\). Given the current configuration (defined as for deterministic online algorithm) and a request, the algorithm may use a finite number of random bits in order to choose the next configuration. Once used, these bits are discarded from the random input (although they may be stored in the memory). (Similar approaches of modeling randomness through a one-way scanning of an infinite string of random bits appear in \cite{Nis90a, Nis90b, BNS89}).
We denote by $A_{m,r}(\sigma)$ the response sequence of the algorithm $A$ to the request sequence $\sigma$, starting with memory state $m$, and using the random number $r$, by $A_r(\sigma)$. The set of all the response sequences of $A$ to $\sigma$, starting in state $m$, is denoted by $S^A_m(\sigma)$. For every $\rho \in S^A_m(\sigma)$ we define the probability that $\rho$ will be the response sequence of $A$ to $\sigma$ (starting with memory state $m$) to be:

$$p^A_m(\sigma, \rho) \triangleq \mu \left( \{ r \in [0,1) : A_{m,r}(\sigma) = \rho \} \right)$$

where $\mu$ is Lebesgue measure. Note that this definition coincides with the intuitive approach of coin tosses. Namely, we assign a probability of $\frac{1}{2}$ to each side of a coin (0 or 1), and measure the probability to get a sequence of coin tosses for which the algorithm $A$ will respond to the request sequence $\sigma$ with the response sequence $\rho$.

If $m$ is the initial memory state of $A$, then we drop the superscript $m$ from the above definitions.

We define the average cost of the request sequence $\sigma$ for the online algorithm $A$ by:

$$C_A(\sigma) \triangleq \sum_{\rho \in S_A(\sigma)} \text{Cost}(\rho) \cdot p_A(\sigma, \rho)$$

A common measure for the performance of online algorithms is Sleator and Tarjan's competitive ratio [ST85]. The competitive ratio of an algorithm $A$ is:

$$w_C(A) \triangleq \lim_{n \to \infty} \sup \max_{C_{OPT}(\sigma) \geq n} \frac{C_A(\sigma)}{C_{OPT}(\sigma)}$$

If $w_C(A) = C$ for some finite number $C$ then we say that $A$ is $C$-competitive.

### 3 Nondeterminism versus Randomness

In this section we introduce two parameters for quantifying the amount of randomization used by an online algorithm. The first one counts the number of binary random bits the algorithm needs. We call it the randomness order of the algorithm. The other one is the number of response sequences an algorithm may use to serve a given request sequence. We call this parameter the nondeterminism order of the algorithm. We say that an algorithm is pseudo-randomized if its randomness order is bounded by some constant (for any request sequence), and it is almost deterministic if its nondeterminism order is bounded by some constant.

Naturally, these two notions are closely related. We shall see that there are algorithms for which they are different and show some conditions under which they are equivalent.

Recall that our aim is to construct efficient pseudo-randomized algorithms, we shall see that this task is achievable if we start off with an almost deterministic algorithm. Namely, for the $k$-server problem on a bounded metric space, any $C$-competitive online algorithm which is almost deterministic can be effectively transformed into a $C$-competitive algorithm which is pseudo-randomized.

**Definition 1 (Randomness Order)** The randomness order of a randomized online algorithm $A$ is a function $R_A$ from the set of request sequence to $\mathbb{N}$. $R_A(\sigma)$ is the maximum number of random bits used by $A$ on any (positive probability) run serving the request sequence $\sigma$. 

Given a configuration and a request there is a finite amount of random bits that an algorithm uses in order to choose the next configuration, so the randomness order is well defined for every randomized online algorithm.

**Definition 2 (Nondeterminism Order)** The nondeterminism order of a randomized algorithm $A$ is a function $N_A$ from the set of request sequences to $\mathbb{N}$. $N_A(\sigma) = |S_A(\sigma)|$ (the number of response sequences $A$ has to the request sequence $\sigma$).

**Definition 3 (Almost Derandomized)** An online algorithm $A$ is pseudo-randomized of order $r_A$, for some $r_A \in \mathbb{N}$, if $R_A(\sigma) \leq r_A$ for every request sequence $\sigma$.

**Definition 4 (Almost Deterministic)** An online algorithm $A$ is almost deterministic of order $d_A$, for some $d_A \in \mathbb{N}$, if $N_A(\sigma) \leq d_A$ for every request sequence $\sigma$.

It can be easily seen that $A$ is pseudo-randomized iff there exists a finite bound $g$, such that on every request sequence $A$ makes at most $g$ many nondeterministic decisions.

A natural example for a competitive online algorithm which is pseudo-randomized (and also almost deterministic), is the discrete version of Karp's algorithm for the continuous circle (which may be seen as a special case of the general algorithm presented in [AKW]). The algorithm chooses a point on the circle at random. This point breaks the circle into a line segment, on which any $k$-competitive algorithm can be applied (such as [CKPV90]). This algorithm was proved to be $2k$-competitive. We can apply this algorithm to a finite polygon (or a finite set of points on a circle). Since the algorithm depends only on the relative position of the breaking point with respect to the nodes of the graph, it can be implemented using finitely many random bits.

One more example for a competitive algorithm which is pseudo-randomized is achieved by applying the technique shown in Section 5 to the marking algorithm [FKL+88].

The following theorem shows that any competitive algorithm which is pseudo-randomized can be transformed into a deterministic competitive algorithm. The relation between the competitive ratios of the two algorithms depends on the randomness order of the pseudo-randomized algorithm. Thus, complete derandomisation of an online algorithm can be achieved by first transforming into a pseudo-randomized algorithm (using the techniques shown in the following sections), then, using the constructive proof of the following theorem, transforming the pseudo-randomized algorithm into a deterministic one.

**Theorem 1** If $A$ is a $C$-competitive algorithm which is pseudo-randomized of order $r_A$, then there is a deterministic algorithm $B$ which is $(2^{r_A} \cdot C)$-competitive.

**Proof** Let $A_r$ be the deterministic online algorithm which simulates $A$ using the string $r \in \{0,1\}^{r_A}$ instead of random bits ($r$ is fixed for $A_r$). For any request sequence $\sigma$ we have:

$$C_A(\sigma) = \sum_r \frac{1}{2^{r_A}} C_{A_r}(\sigma)$$

Each $A_r$ has to be at most $(2^{r_A} \cdot C)$-competitive, otherwise $A$ is not $C$-competitive. Thus, we can choose $B$ to be any of the $A_r$'s. \qed
Theorem 2 If an online algorithm $A$ is pseudo-randomized of order $r_A$ then it is also almost deterministic of order $2^{r_A}$.

Proof Just note that, for every request sequence $\sigma$, $N_A(\sigma) \leq 2^{r_A(\sigma)}$.  \hfill $\Box$

Claim 3 There exists an almost deterministic online algorithm $A$ which is not pseudo-randomized.

Proof Let $|V| = 3, k = 2$, where $A$ is a limited probabilistic version of the LRU (Least Recently Used) algorithm. Most of the time $A$ will serve any request deterministically by moving the server from the point that was least recently requested. Let the initial location be $\{u, v\}$. If $v$ is requested $n$ successive times just before $w \notin \{u, v\}$ is requested for the first time then, with probability $1 - \frac{1}{2^n}$, $A$ serves the first request to $w$ by the server located on $u$. After $w$ was requested for the first time, $A$ will continue to serve deterministically according to the LRU.

Since the algorithm chooses among two different responses only once, it is almost deterministic. Nevertheless, for every $n \in \mathbb{N}$ there is a sequence, namely $v^n w$, for which $A$ must use $n$ random bits to choose its response. It follows that $A$ is not pseudo-randomized.  \hfill $\Box$

Remark Although the above algorithm is not pseudo-randomized, it can be approximated by an almost derandomized algorithm. The notion of an approximation of an online algorithm by another online algorithm is defined in Section 4, and a technique for approximating an almost deterministic algorithm by an almost derandomized algorithm is demonstrated.

The algorithm of the above example needs infinite memory (to count occurrences of requests to the node $v$). A similar example can be constructed with a bounded memory if the underlying metric space is infinite (e.g. if $V = \mathbb{N}$ then an online algorithm may need $n$ random bits to choose how to serve the first request to the point $n \in \mathbb{N}$). There is yet a third possible difficulty in trying to derandomized almost deterministic algorithms - the algorithm may be just wasting random bits. Clearly, an online algorithm can scan random bits even when it decides on the next configuration deterministically. Or it may be the case that random bits are used in order to choose between different memory states, where the next location is decided deterministically. This leads us to the next definition.

Definition 5 (Thrifty Algorithms) A randomized online algorithm $A$ is thrifty iff it uses random bits only when it has to choose between at least two different (next state) locations.

Thrifty is not a restriction on the power of randomized online algorithms. That is, we can transform any randomized online algorithm $A$ to a thrifty algorithm $B$ which responds to each request sequence in the same way as $A$ does (and thus has also the same performance). The reason the next claim is not as self evident as it may seem is that an algorithm may fail to be thrifty due to randomly choosing between different memory states that may later effect its behavior. The remedy to this problem is through extending the memory to compensate for the loss in randomness. This is yet another manifestation of the close relationship between these resources (Raghavan and Snir [RS89] discuss other such phenomena).

Claim 4 For every randomized online algorithm $A$ there is a thrifty online algorithm $B$ such that $p_A(\sigma, p) = p_B(\sigma, p)$ for every request sequence $\sigma$ and every response sequence $p \in S(\sigma)$.

6
Proof Given any randomized algorithm \( A \), we can transform it to a thrifty algorithm \( B \) in two steps. In the first step every redundant use of random bits can be simply eliminated, thus ensuring that \( A \) does not use random bits when it actually decides on the next configuration deterministically.

The second step will ensure that the algorithm won't use random bits just to change its memory state. In this step we may have to enlarge the set of memory states of the algorithm. Let \( M \) be the collection of \( A \)'s memory states. The memory states of \( B \) will be pairs \((m, \tau)\) where \( m \in M \) and \( \tau \in \{0, 1\}^* \). \( B \) will imitate \( A \), but whenever \( A \) uses random bits to choose among configurations which have the same location but different memory states, \( B \), rather than scanning any random bits, writes into its memory all possible next-memory-states of \( A \), each paired with the sequence(s) of random bits \( \tau \) that could have led to it. Upon having to choose between at least two different locations (of the servers), \( B \) scans all the random bits that have been scanned by \( A \) since the last time \( B \) accessed the random input string, finds out which string \( \tau^* \) is the result of the bits just read, discards all the pairs whose \( \tau \) part is inconsistent with it, and behaves according to \( A \)'s program on the basis of the memory states whose binary string component equals \( \tau^* \). Clearly \( B \) manages to simulate \( A \) using random bits only when they are needed for choosing between different locations.

The following theorem shows that the conditions mentioned along the above discussion suffice to guarantee equivalence between almost determinism and almost derandomization of algorithms.

**Theorem 5** Let \( A \) be an online algorithm for the \( k \)-server problem on a finite metric space. If \( A \) is thrifty, almost deterministic, and has a finite set of memory states, then it is pseudo-randomized.

**Proof** Assume \( V \) is finite and let \( A \) be a thrifty algorithm which is almost deterministic of order \( d_A \), and has a finite set of memory states. Since the space \( V \) is finite there is a finite number of possible locations of the servers, which together with the finiteness of the set of memory states of \( A \) implies that there is a finite number of possible configurations. Thus, there is an upper bound \( c_A \) on the number of random bits used by \( A \) to choose the next configuration, given the current configuration and the current request. For any request sequence \( A \) has no more than \( d_A \) response sequences, so there are no more than \( d_A - 1 \) requests for which \( A \) chooses its response randomly (for all the other requests \( A \) chooses its response deterministically). Since \( A \) uses no more than \( c_A \) random bits for each choice. This implies that \( R_A(\sigma) \leq d_A \cdot c_A \) for every request sequence \( \sigma \). Since \( c_A \) and \( d_A \) are finite, \( A \) is pseudo-randomized. \( \square \)

We conclude this section with a constructive transformation of almost deterministic algorithms into pseudo-randomized one. The transformation preserves the competitive ratio.

**Theorem 6** Let \( A \) be a \( C \)-competitive online for a \( k \)-server on a bounded metric space. If \( A \) is almost deterministic then there is a \( C \)-competitive algorithm for this problem which is pseudo-randomized.

**Proof** Assume \( A \) is a \( C \)-competitive algorithm which is almost deterministic of order \( d \), and let \( \sigma^* \) be a request sequence for which \( A \) has \( d \)-many possible response sequences. We may consider the response sequences of \( A \) to request sequences which start with \( \sigma^* \) as a distribution over \( d \) deterministic algorithms \( A_1, \ldots, A_d \), where each \( A_i \) is assigned a probability \( p_i \) - the probability of choosing \( A_i(\sigma^*) \) as the response of \( A \) on \( \sigma^* \). Since \( A \) is \( C \)-competitive, there is a constant \( a \) such that for every request sequence \( \sigma \) which starts with \( \sigma^* : \)

\[
C_A(\sigma) = \sum_{i=1}^{d} p_i \cdot C_{A_i}(\sigma) \leq C \cdot C_{OPT}(\sigma) + a
\]

7
This implies that each algorithm $A_i$ is at most $(\frac{d}{d-1})$-competitive on request sequences which starts with $\sigma^*_i$. Any competitive algorithm can be forced to locate its serves in any given location $l$ by a long enough request sequence $\sigma_l$ consisting of repeated requests to the points in that location (because the optimal offline algorithm can locate its serves in $l$ and pays nothing for serving $\sigma_l$). Obviously, for every location $l$ there is a sequence which is long enough for every $A_i$, $1 \leq i \leq d$. Furthermore, since the metric space is bounded, the cost of serving $\sigma_l$ (by any algorithm) is bounded.

Now we define the algorithm $B$ as follows. Given a request sequence $\sigma$, let $l$ be the initial location it dictates. $B$ will simulate $A$ on the request sequence $\sigma^*_l \sigma^*_l \sigma^*_l$. Since the cost of $\sigma^*_l$ and $\sigma_l$ for $A$ is finite, $B$ is also $C$-competitive. The number of random bits used by $B$ is just the (finite) number of random bits used by $A$ to serve $\sigma^*_l$.

4 Trading Competitiveness for Derandomization

In Section 3 we have seen that there exist almost deterministic algorithms which are not pseudo-randomized. That is, although the number of random choices they make is bounded by a constant (regardless of the length of the request sequence), there is no a priory bound to the number of random bits they may need. We have stated some necessary conditions under which algorithms are guaranteed to be pseudo-randomized, or can be constructively transformed into pseudo-randomized algorithms without any loss in their performance. So far we cared mainly about whether the randomness order of an algorithm is finite or infinite and did not worry about its exact value. Recall that by Theorem 1, once we wish to transform a randomized algorithm into a deterministic one, the randomness order becomes significant - it sets the upper bound on the loss in competitiveness caused by the transformation.

In this section we focus on the relationship between the degree of nondeterminism of the algorithm we start off and the randomness degree of its transformed version. We introduce a notion of $\epsilon$-approximating one algorithm by another. We show that, on one hand, the smaller $\epsilon$ is, the closer the competitive ratios of the algorithms are kept. On the other hand, if we wish to $\epsilon$-approximate an almost deterministic algorithm by a pseudo-randomized one then, the order of randomness of the approximating algorithm we construct grows as $\epsilon$ is decreased.

**Definition 6 (Approximation)** The online algorithm $B$ $\epsilon$-approximates the online algorithm $A$, for some $0 \leq \epsilon \leq 1$, iff for every request sequence $\sigma$ the following two conditions are satisfied:

1. $S_B(\sigma) = S_B(\sigma)$.
2. $|p_A(\sigma, \rho) - p_B(\sigma, \rho)| < \epsilon$ for every $\rho \in S_A(\sigma)$.

The definition of approximation is rather delicate. We could have strengthened demand 2 to: $|p_A(\sigma, \rho) - p_B(\sigma, \rho)| < \epsilon \cdot p_A(\sigma, \rho)$, thus ensuring that if $B$ $\epsilon$-approximates $A$ then $w_C(B) < (1 + \epsilon) \cdot w_C(A)$, for every $A$ and $B$. But then one couldn't get a significant reduction in the number of random bits used by $A$. In particular, one can verify that, with regard to this definition, if $A$ is not pseudo-randomized then neither is $B$. The fact that $B$ $\epsilon$-approximates $A$ does not ensures that the performances of $A$ and $B$ are almost the same. The relation between their performances depends on a further property of $A$, which we call the approximation ratio, and is defined as follows:

8
Definition 7 (The approximation ratio) Given an online algorithm $A$ and a request sequence $\sigma$, let the uniform cost of serving $\sigma$ by $A$ be:

$$C^*_A(\sigma) = \frac{1}{|S_A(\sigma)|} \sum_{\rho \in S_A(\sigma)} \text{Cost}(\rho)$$

$C^*_A(\sigma)$ is the expected cost of serving $\sigma$ when all the response sequences of $A$ to $\sigma$ are assigned equal probabilities.

The approximation ratio of an online algorithm $A$ is defined as:

$$X_A = \limsup_{n \to \infty} \max_{\alpha(\sigma) \geq n} \frac{C^*_A(\sigma)}{C^*_A(\sigma)}$$

Roughly, this ratio is large if $A$ allows (even with very small probability) the use of noncompetitive response sequences.

The following lemma shows that if $A$ is almost deterministic and it has a finite approximation ratio, then its efficiency is preserved by its approximations.

Lemma 7 Let $A$ be an almost deterministic algorithm of order $d_A$, such that $X_A$ is finite, and let $B$ be an algorithm which $\epsilon$-approximates $A$. Then:

$$w_C(B) \leq (1 + \epsilon \cdot d_A \cdot X_A) w_C(A)$$

Proof For every request sequence $\sigma$ we have:

$$C_B(\sigma) = \sum_{\rho \in S_B(\sigma)} p_B(\sigma, \rho) \cdot \text{Cost}(\rho)$$

$$\leq \sum_{\rho \in S_B(\sigma)} (p_A(\sigma, \rho) + \epsilon) \cdot \text{Cost}(\rho)$$

$$= C_A(\sigma) + \epsilon \cdot |S_A(\sigma)| \cdot C^*_A(\sigma)$$

$$\leq C_A(\sigma) \left(1 + \epsilon \cdot d_A \cdot \frac{C^*_A(\sigma)}{C^*_A(\sigma)} \right)$$

Using the definitions of the competitive ratio and the approximation ratio we get that $w_C(B) \leq w_C(A)(1 + \epsilon \cdot d_A \cdot X_A)$. \qed

Which algorithms have finite approximation ratio? It can be easily verified that an algorithm which is pseudo-randomized has a finite approximation ratio. Using theorems 3 and 5 from Section 3 we therefore have:

Corollary 8 Let $A$ be an online algorithm for the $k$-server problem on a finite metric space. If $A$ is thrifty, almost deterministic and has a finite set of memory states, then $A$ has a finite approximation ratio.

Corollary 9 Let $A$ be a $C$-competitive online algorithm for the $k$-server problem on a bounded metric space. If $A$ is almost deterministic then it can be transformed into a $C$-competitive algorithm which has a finite approximation ratio.
We now show how to construct an almost derandomized algorithm which $\epsilon$-approximates a given almost deterministic algorithm.

Lemma 10 Let $A$ be an online algorithm which is almost deterministic of order $d$. For every $0 < \epsilon \leq 1$ there exists an online algorithm $B$ which $\epsilon$-approximates $A$ and is pseudo-randomized of order $(d - 1) \cdot \lceil \log_2 \frac{d}{\epsilon} \rceil$.

Proof See Appendix A.

Corollary 11 For every almost deterministic online algorithm of order $d$ which has a finite approximation ratio $X$, and every $0 < \epsilon \leq 1$, there is a pseudo-randomized online algorithm of order $(d - 1) \cdot \lceil \log_2 \frac{d}{\epsilon} X \rceil$, which has a competitive ratio of $w_C(B) \leq w_C(A) \cdot (1 + \epsilon)$.

By invoking Theorem 1 from Section 3, and using the previous corollary for $\epsilon = 1$, we can now fully derandomize any randomized algorithm that has a bound on the number of different responses it may use on any request sequence, and has a finite approximation ratio.

Corollary 12 Let $A$ be an online algorithm which is almost deterministic of order $d$ and has approximation ratio $X$. Then there exists a deterministic online algorithm $B$ such that $w_C(B) \leq 2 \cdot (d^2 \cdot X)^{d-1} \cdot w_C(A)$.

5 Phased Algorithms

Not being able to guarantee efficient derandomization of all randomized online algorithms, we have so far listed some sufficient conditions for the existence of such constructions. Common to all these conditions was the assumption that the algorithms are almost deterministic. As this is a very restricting demand, one would like to find further derandomization techniques that apply to other classes of algorithms. In this section we offer such a technique. It is based upon the idea of recycling random bits and we can apply it to a certain family of algorithms, the class of phased algorithms.

Roughly, $A$ is phased if any request sequence $\sigma$ for $A$ can be divided into phases, such that the responses of $A$ to one phase are independent of its responses to another phase.

Definition 8 A randomized online algorithm $A$ is phased of order $r$ if for every request sequence $\sigma$ there is a partition $\sigma = \sigma_1 \ldots \sigma_n$ (each $\sigma_i$ is called a phase), which satisfies the following two conditions:

1. There is a sequence of configuration $c_1, \ldots, c_n$ such that, with probability 1, $A$ is in configuration $c_i$ as it concludes serving the phase $\sigma_i$, for every $1 \leq i < n$.
2. $A$ uses no more than $r$ random bits in order to serve any phase $\sigma_i$ (for every $1 \leq i \leq n$).

A typical example of a phased algorithm is the marking algorithm of Fiat et al. [FKL+88]. This algorithm, which we denote by MARK, stores in its memory a set of marked points. Initially, all the points in the initial location (and only them) are marked. Whenever an unmarked node is requested, it is added to the set of marked points. Whenever the set of marked points contains
many points, all the marks are erased. To decided which server should be moved to cover a
next request, the algorithm chooses a server uniformly at random from the set of servers sitting
on unmarked points. Obviously, a request sequence can be divided into phases, such that a phase
ends when exactly $k$ different points are requested (since the beginning of that phase).

Fiat et al. prove that, over a uniform metric space, $w_C(MARK) = 2 \cdot H_k$, where $H_k = \sum_{i=1}^{k} \frac{1}{i}$
($H_k$ satisfies $\ln k < H_k < \ln k + 1$). They also prove that $w_C(MARK) = H_k$ for the case $k = |V| - 1$,
and that this is the lower bound for all $k$.

It can be easily verified that $MARK$ is neither almost deterministic nor pseudo-randomized.
Nevertheless, $MARK$ is a phased algorithm of order $k \cdot \lceil \log_2 k \rceil$. Using the following constructive
theorem we can transform it into a pseudo-randomized algorithm of order $k \cdot \lceil \log_2 k \rceil$ (i.e. this
many random bits suffice for implementing it on any request sequence).

Let us first prove that if the algorithm $A$ is phased then the average cost of any request sequence
$\sigma$ to $A$ is the sum of the average costs of each of $\sigma'$s phases to $A$.

Lemma 13 Let $A$ be a phased algorithm and let $\sigma$ be a request sequence. Assume that $\sigma_1, \ldots, \sigma_n$
is the partition of $\sigma$ into phases, and let $m_i$ be the memory state of $A$ at the end of the phase $\sigma_i$,
for every $1 \leq i \leq n$. Then:

$$C_A(\sigma) = \sum_{\rho \in S_A(\sigma)} p_A(\sigma, \rho) \cdot \text{Cost}(\rho) = \sum_{i=1}^{n} \sum_{\rho_i \in S_A^m(\sigma_i)} p_A^m(\sigma_i, \rho_i) \cdot \text{Cost}(\rho_i)$$

Proof By the definition of a phased algorithm:

$$S_A(\sigma) = \{ \rho_1 \cdots \rho_n : \rho_i \in S_A^m(\sigma_i), 1 \leq i \leq n \}$$

and for every $\rho \in S_A(\sigma)$:

$$p_A(\sigma, \rho) = \prod_{i=1}^{n} p_A^m(\sigma_i, \rho(t_i, t_{i+1}))$$

Thus we $C_A(\sigma)$ can be calculated as follows:

$$C_A(\sigma) = \sum_{\rho \in S_A(\sigma)} p_A(\sigma, \rho) \cdot \text{Cost}(\rho)$$

$$= \sum_{\rho_i \in S_A^m(\sigma_i)} \cdots \sum_{\rho_n \in S_A^m(\sigma_n)} \left[ \prod_{i=1}^{n} p_A^m(\sigma_i, \rho_i) \right] \left( \sum_{i=1}^{n} \text{Cost}(\rho_i) \right)$$

The order of the sum can be changed as follows:

$$C_A(\sigma) = \sum_{i=1}^{n} \sum_{\rho_i \in S_A^m(\sigma_i)} \text{Cost}(\rho_i) \cdot p_A^m(\sigma_i, \rho_i) \cdot \sum_{\rho_j \in S_A^m(\sigma_j) \neq \rho_i} \prod_{j \neq i} p_A^m(\sigma_j, \rho_j)$$

For all $1 \leq i \leq n$:

$$\sum_{\rho_j \in S_A^m(\sigma_j) \neq \rho_i} \prod_{j \neq i} p_A(\sigma_j, \rho_j) = 1$$
because the sum is for all the responses to all of the phases except $\sigma_i$. Thus we get:

$$C_A(\sigma) = \sum_{i=1}^{n} \sum_{\rho \in S_A^m(\sigma_i)} \text{Cost}(\rho_i) \cdot p_A^m(\sigma_i, \rho_i)$$

\[\square\]

**Theorem 14** If $A$ is a phased algorithm of order $r$ then there is a pseudo-randomized algorithm $B$ of order $r$ which satisfies $w_C(A) = w_C(B)$.

**Proof** Given a phased algorithm $A$ of order $r$, we construct the algorithm $B$ as follows. Initially, $B$ simulates the algorithm $A$, and stores the random bits it uses, until it has $r$ bits in its memory. Then $B$ simulates the algorithm $A$ using these bits as random bits. Each time $A$ uses random bits, the algorithm $B$ uses the bits stored in its memory (in a cyclic way).

Obviously, since $B$ uses the same $r$ bits for all phases, we can't have $S_B(\sigma) = S_A(\sigma)$ for every request sequence $\sigma$. Nevertheless, we show that $C_B(\sigma) = C_A(\sigma)$ for every request sequence $\sigma$.

Let $\sigma$ be a request sequence, $\sigma_1, \ldots, \sigma_n$ its partition into phases, and $c_i = (l_i, m_i)$ be the configuration of $A$ at the end of the phase $\sigma_i$. Assume further that for every $1 \leq i \leq n$, the phase $\sigma_i$ sets $l_i$ to be the initial location of the servers, and requests the subsequence $\sigma(t_i, t_{i+1})$ ($t_i$ marks the beginning of the $i$-th phase, where $t_{n+1} = |\sigma| + 1$).

By lemma 12 we get:

$$C_A(\sigma) = \sum_{\rho \in S_A(\sigma)} p_A(\sigma, \rho) \cdot \text{Cost}(\rho) = \sum_{\rho \in S_A(\sigma)} \sum_{\rho \in S_A(\sigma)}^n p_A^m(\sigma_i, \rho_i) \cdot \text{Cost}(\rho_i)$$

$B$ serves each phase using $r$ independent random bits. Thus, for every $1 \leq i \leq n$ and every $\rho_i \in S_A^m(\sigma_i)$ there exists $\rho \in S_B(\sigma)$ such that $\rho_i = \rho(t_i, t_{i+1})$. That is, for every $1 \leq i \leq n$:

$$S_A^m(\sigma_i) = \{ \rho_i : \exists \rho \in S_B(\sigma) \rho(t_i, t_{i+1}) = \rho_i \}$$

Further more, since for every $1 \leq i \leq n$:

$$\mu(\{ \rho \in \mathbb{R}[0,1) : B_\rho(\sigma) = \rho, \rho(t_i, t_{i+1}) = \rho_i \}) = \mu(\{ \rho \in \mathbb{R}[0,1) : A_{m_i, r}(\sigma_i) = \rho_i \})$$

Then, for every $1 \leq i \leq n$:

$$p_A^m(\sigma_i, \rho_i) = \sum_{\rho \in S_B(\sigma)} p_B(\sigma, \rho)$$

Thus we get:

$$C_A(\sigma) = \sum_{i=1}^{n} \sum_{\rho \in S_B(\sigma)} \text{Cost}(\rho) \sum_{\rho \in S_B(\sigma)} p_B(\sigma, \rho)$$

And by changing the sum order we get:

$$C_A(\sigma) = \sum_{\rho \in S_B(\sigma)} \rho \sum_{i=1}^{n} \text{Cost}(\rho(t_i, t_{i+1})) = C_B(\sigma)$$

Since this is true for every request sequence $\sigma$, we get $w_C(B) = w_C(A)$. Clearly, by the definition of $B$, it is pseudo-randomized of order $r$. \[\square\]
Corollary 15 For every phased algorithm $A$ there exists a deterministic algorithm $B$ such that $w_C(B) \leq w_C(A) \cdot 2^r$, where $r$ is the order of $A$'s phase.

Note that $w_C(A)$ is the weakly randomized competitive ratio (i.e., its ratio against an oblivious adversary) while $w_C(B)$ is a deterministic competitive ratio.

References


A Approximation of Online Algorithms

We prove that any almost deterministic algorithm $A$ can be approximated by a pseudo-randomized algorithm $B$, such that the randomness order of $B$ depends on the nondeterminism order of $A$ and the quality of the approximation.

**Lemma 16** Let $A$ be an online algorithm which is almost deterministic of order $d_A$, and let $0 < \epsilon < 1$. Then there exist an online algorithm $B$ which $\epsilon$-approximates $A$ and is pseudo-randomized of order $(d_A - 1) \cdot \lceil \log_2 \frac{d_A}{\epsilon} \rceil$.

**Proof** Given an online algorithm $A$ for which $N_A(n) \leq d_A$ for every $n \in \mathbb{N}$ and $0 < \epsilon < 1$, we define an algorithm $B$ which responses to every request sequence almost like $A$ does. Whenever $A$ chooses its response using random bits, $B$ chooses its response using no more than $\alpha = \lceil \log_2 \frac{d_A}{\epsilon} \rceil$ random bits, while ensuring that the probability of each of its response sequence is close enough to the probability of that response sequence to be chosen by $A$. For every configuration $c$ and for every request $v$ for which $A$ uses more than $\alpha$ random bits in order to choose the next configuration, $B$ will respond exactly as $A$ does. Let $c$ and $v$ be such that $A$ uses more than $\alpha$ bits in order to choose the next configuration. Assume further that the configuration $C_i$ is chosen with probability $P_i$, for $1 \leq i \leq t$. We show that $B$ can approximate this behavior using no more than $\alpha$ random bits. Make a partition of $[0, 1)$ into $t$ sets $B_1, \ldots, B_t$ in the following way:

- Let $q_0 = 0$, and $q_i = \sum_{j=1}^{i} P_j$, for every $1 \leq i \leq t$.
- Define $B_i = \{ r \in [0, 1) : 2^{q_{i-1}} \leq 2^r < 2^{q_i} \}$.  

It is easy to verify that $|\mu(B_i) - P_i| < \frac{1}{2^t} < \frac{1}{d_A}$. Thus, for every $1 \leq i \leq t$ we can define $B$, such that if the current configuration is $C$ and $v$ is requested, then $B$ will choose the configuration $C_i$ as the next configuration if the current random number $r$ satisfies $r \in B_i$.

Since $A$ uses random bits to choose its next response no more than $d_A - 1$ times, so does $B$, and since $B$ uses at most $\alpha$ bits each time, we get $R_B(\sigma) < (d_A - 1) \cdot \alpha$, for every request sequence $\sigma$. 

14
It remains to show that $B$ $\epsilon$-approximates $A$. We prove by induction on $i$, $1 \leq i \leq d_A$, that for every request sequence which satisfy $|S_A(\sigma)| = i$ and for every $\rho \in S_A(\sigma)$, $B$ satisfies $|p_A(\sigma, \rho) - p_B(\sigma, \rho)| < (i - 1) \cdot \frac{\epsilon}{d_A}$. For $i = 1$ $S_A(\sigma) = \{\rho\}$, and $p_A(\sigma, \rho) = p_B(\sigma, \rho) = 1$. Assuming the claim is true for every $1 \leq j < i$ we prove it for $i$. Given a request sequence $\sigma$ for which $|S_A(\sigma)| = i$, let $\sigma_1$ be the longest prefix of $\sigma$ which satisfies $|S_A(\sigma_1)| < i$, and let $\sigma = \sigma_1 \sigma_2$. Let $\rho_1 \in S_A(\sigma_1)$, we assume

$$\{\rho_2 : \rho_1 \rho_2 \in R_A(\sigma)\} = \{\rho_2^1, \ldots, \rho_2^t\}$$

Let $p_A^i = \frac{p_A(\sigma_1, \rho_1)}{p_A(\sigma_1, \rho_1)}$ and $p_B^i = \frac{p_B(\sigma_1, \rho_1)}{p_B(\sigma_1, \rho_1)}$ for every $1 \leq j \leq t$. By the induction hypothesis we have:

$$|p_A(\sigma_1, \rho_1) - p_B(\sigma_1, \rho_1)| < (i - 2) \cdot \frac{\epsilon}{d_A}$$

We have defined $B$ such that for every $1 \leq j \leq t$:

$$|p_A^i - p_B^i| < \frac{\epsilon}{d_A}$$

Thus we get:

$$|p_A(\sigma, \rho_1) - p_B(\sigma, \rho_1)|$$

$$= |p_A(\sigma_1, \rho_1) \cdot p_A^i - p_B(\sigma_1, \rho_1) \cdot p_B^i|$$

$$= |p_A(\sigma_1, \rho_1) - p_B(\sigma_1, \rho_1)| |p_A^i - p_B^i| + |p_B(\sigma_1, \rho_1)| |p_B^i| - |p_A(\sigma_1, \rho_1)| |p_A^i|$$

$$< (i - 2) \cdot \frac{\epsilon}{d_A} + \frac{\epsilon}{d_A}$$

$$= (i - 1) \cdot \frac{\epsilon}{d_A}$$

Thus we have shown that $B$ $\epsilon$-approximates $A$. □