Randomized Paging Algorithms
and Measures for their Performance

by

E. Dichterman

Technical Report #692
October 1991
Randomized Paging Algorithms
and Measures for their Performance

Eli Dichterman
Dept. of Computer Science
Technion, Haifa 32000, Israel

July 1991

Abstract

We study three randomized paging algorithms, using two different measures for their performances. One of the measures is the competitive ratio which was introduced by Sleator and Tarjan [ST85]. Although it is well-motivated, Ben-David and Borodin [BDB] have recently demonstrated some counter-intuitive features of this measure (concerning the roles of memory and finite lookahead in online algorithms), and have suggested an alternative measure based on the worst-case amortized cost, which is the second measure we use. We show a lower bound of this measure for paging algorithms and present an algorithm which achieves this bound. Since this algorithm is not even competitive, it may serve as an extreme evidence to the inherent differences between the two measures. Two other algorithms studied here are the known marking algorithm, which was introduced by Fiat et al. [FKL+88] and was proved to be $2H_k$-competitive (against an oblivious adversary, which is the one considered in this work), and the algorithm RANDOM which was introduced by Raghavan and Snir, and was proved to be $k$-competitive. We show that although the marking algorithm has a better competitive ratio, RANDOM has a better worst-case amortized cost (at least for the two cases analyzed in this work). Two related aspects considered here are the memory size of the algorithms and the number of random bits they use.

1 Introduction

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 system [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, which is an abstract model to many online problems.

*e-mail: eli@cs.technion.ac.il
The k-server problem consists of a metric space, on which k mobile servers are located. An online algorithm is presented with a sequence of requests, where each request is a point in the space. The algorithm serves a request by moving the servers in such a way that one of them covers the requested point. Online algorithms serve each request without any knowledge on the subsequent requests. There is a cost attached to each run of the algorithm - the total distance incurred by the servers while being moved to serve sequences of requests. The ultimate goal of the algorithm is to minimize this cost.

Sleator and Tarjan [ST85] introduced the competitive ratio as a measure for the performance of online algorithms. An online algorithm is said to be c-competitive if the cost incurred by using it on any request sequence is within a factor of c of the minimal possible cost of serving that sequence. The minimal cost may be attributed to an optimal offline algorithm which serves the requests knowing the entire sequence in advance.

Ben-David and Borodin pointed out two significant counter-intuitive aspects of the competitive ratio. One such aspect concerns the role of memory in competitive algorithms. They gave some lower bounds on the amount of memory needed for running competitive online algorithms, which imply that there are online tasks for which any competitive algorithm should use unbounded memory space. Another aspect is the role of finite lookahead in online algorithms. They show that the definition of the competitive ratio reflects no improvement on the performance of an online algorithm due to a finite amount of lookahead. As an alternative measure they suggested the max/max ratio, which is based on worst-case analysis, and which exhibits, in respect to the above aspects, a more intuitive behavior. The max/max ratio compares the worst-case amortized behavior of an algorithm with that of optimal offline algorithm. An algorithm is said to have max/max ratio \( M \) if the cost incurred by using it on any request sequence is within a factor of \( M \) of the maximal cost that an optimal offline algorithm pays on a sequence of the same length. Except for the normalizing by the best that can be done using an optimal offline algorithm, this measure actually reflects the worst-case amortized cost of an online algorithm. In this work we use both the competitive ratio and the max/max ratio (or rather the amortized cost) as measures for the performances of the online algorithms we analyze.

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 prove 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. The cost of serving a sequence of requests in this model is defined as the expected (total) distance incurred by the servers while serving it according to the algorithm's randomized strategy.

The paper is organized as follows. In Section 2 we summarize in two tables the results known on the three randomized algorithms discussed in this work (including the results of this work). The first table compare their performances relative to both the competitive ratio and the worst-case amortized cost. The second table presents their resources needs, namely their memory size and the amount of random bits they use for each move. The tables reflects that the precedences we assign to the algorithms may depend on our concerns and needs. In Section 3 we formally define the notions of the k-server problem, randomized online algorithms and the measures for
their performance, namely the competitive ratio and the worst-case amortized cost. In Section 4 we show a lower bound for the worst-case amortized cost in a uniform metric space, and show an algorithm which achieves this bound. Although the algorithm has an optimal worst-case amortized cost it is not even competitive, thus serving as an extreme example for the inherent differences between the two measures. A significant drawback of the demonstrated algorithm is that it needs $O(n \log n)$ memory bits, where $n$ is the number of points in the space. In Section 5 we analyze the simple and memoryless RANDOM algorithm, which was proved to be $k$-competitive, even for stronger adversary than the oblivious one [RS89]. The analysis of the algorithm is based on asymptotic theory, which is a novel technique in the context of online algorithms, and may find further uses. The analysis of RANDOM introduced also a new type of adversary - a greedy one, which chooses always the next request to be the one that maximize the cost of serving that request by the online algorithm. This adversary is proved to be optimal against the RANDOM algorithm, and is conjectured to be optimal against any memoryless adversary. Although this notion is quiet superficial, it may help analyzing memoryless randomized algorithms, since not all the request sequences have to be considered, but only those which are greedy. In Section 6 we analyze the marking algorithm which was proved to be $O(\log k)$-competitive [FKL+88]. We show that although this algorithm has a better competitive ratio than that of RANDOM, its worst-case amortized cost is worse than that of RANDOM. Thus, while RANDOM has a lower worst-case cost per request, MARK has a relative lower cost per request, when comparing to the optimal offline algorithm.

2 Summary of Results

Let us first summarize the results shown in this paper. The next table presents the performance of each randomized paging algorithm, relative to both the competitive ratio, and the worst-case amortized cost.

<table>
<thead>
<tr>
<th>Paging Algorithm</th>
<th>Competitive Ratio</th>
<th>Worst-Case Amortized Cost $k = 2$</th>
<th>$2 &lt; k &lt; n - 1$</th>
<th>$k = n - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIFORM</td>
<td>$\infty$</td>
<td>$1 - \frac{1}{n-1}$</td>
<td>$\frac{n-k}{n-1}$</td>
<td>$\frac{1}{k}$</td>
</tr>
<tr>
<td>RANDOM</td>
<td>$k$</td>
<td>$1 - \frac{1}{n-1}$</td>
<td>?</td>
<td>$\frac{2}{k+1}$</td>
</tr>
<tr>
<td>MARK</td>
<td>$2H_k$</td>
<td>$1$</td>
<td>$\frac{n-k(H_k - H_{n-k} + 1)}{k}$</td>
<td>$\frac{H_k}{k}$</td>
</tr>
<tr>
<td>Lower Bound</td>
<td>$H_k$</td>
<td>$1 - \frac{1}{n-1}$</td>
<td>$\frac{n-k}{n-1}$</td>
<td>$\frac{1}{k}$</td>
</tr>
</tbody>
</table>
Remarks

- The k-competitiveness of RANDOM was proved in [RS89] (even for a stronger adversary than the oblivious one considered here), and the $2H_k$-competitiveness of MARK was proved in [FKL*88].
- The competitive ratio of MARK in the case $k = n - 1$ is $H_k$.
- The worst-case amortized cost of MARK in the case $k = 2$, $n = 3$ is $\frac{3}{4}$ (and not 1).

Note that while UNIFORM has an optimal worst-case amortized cost it is not even competitive, and while MARK is better than RANDOM relative to the competitive ratio, this is not the case relative to the worst-case amortized cost (we show that RANDOM is better than MARK for the cases $k = 2$ and $k = n - 1$, and conjecture that this true for the other cases as well). Thus, different measures induce different precedences among the algorithms, as they actually reflect different concerns.

Except for the performance of the algorithm, there are two more aspects concerning the implementation of the algorithms, namely their memory size and the amount of random bits they use for each move. The resources needs (memory and random bits) of the algorithms are summarized in the following table.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Memory Size</th>
<th>Random Bits Per Move</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIFORM</td>
<td>$n \log n$</td>
<td>log $n$</td>
</tr>
<tr>
<td>RANDOM</td>
<td>0</td>
<td>log $k$</td>
</tr>
<tr>
<td>MARK</td>
<td>$k \log k$</td>
<td>log $k$</td>
</tr>
</tbody>
</table>

Remarks

- Raghavan and Snir [RS89] show variants of the algorithm RANDOM in which the number of random bits per move is traded with the memory size, such that their sum remains log $k$.
- Ben-David and Dichterman [BDD] show that the algorithm MARK can be transformed into an algorithm which use a bounded number of bits ($O(k \log k)$), regardless of the length of its input.

Note that with respect to these aspects the algorithm UNIFORM has a significant drawback, because its memory size as well as the amount of random bits it uses for each move, depend on the size of metric space, while the needs of RANDOM and MARK depend only on $k$ (and RANDOM is even memoryless).

3 Definitions and Notations

In this section we define formally the basic notions of the $k$-server problem, randomized online algorithm, the competitive ratio and the worst-case amortized cost.
The Metric Space 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 
\(d(u,v) = 0\) for every \(v \in V\). The space is called uniform iff \(d(u,v) = 1\) for every \(u \neq v\). (In 
this case the problem is called the uniform \(k\)-server problem).

Location A location is a set of \(k\) points in the space, representing the locations of the servers. 
We denote the set of all the possible locations of \(k\) servers in the space \(V\) by \(L^k_v\), that is 
\[L^k_v = \{l \subseteq V : |l| = k\}\].

Request A request is a point \(v \in V\). It represents a request for a service at that point, which can 
be satisfied by occupying the point with a server (i.e. the request is satisfied by moving the 
servers to a location \(l\) which satisfies \(v \in l\)).

Request Sequence 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 \(\sigma(i), \ldots, \sigma(j - 1)\) is denoted by \(\sigma(i,j)\) (if \(i \neq j\) then the subsequence is empty).

Service Sequence A service sequence \(\rho\) for the request sequence \(\sigma\) is a sequence of \(|\sigma| + 1\) many 
locations, satisfying \(\rho(0) = \{\sigma(-k + 1), \ldots, \sigma(0)\}\) (the initial location), and \(\sigma(t) \in \rho(t)\) for 
every \(1 \leq t \leq |\sigma|\). The set of all the service sequence for the the request sequence \(\sigma\) is denoted 
by \(S(\sigma)\).

Cost of Service Sequence We define the cost of moving from location \(l_1\) to location \(l_2\) by 
\[
\text{Cost}(l_1, l_2) = \min_{f : l_1 \rightarrow l_2} \sum_{v \in l_1} d(v, f(v))
\]
which is the minimal distance that the servers have to move from location \(l_1\) to location \(l_2\). 
The cost of the service sequence \(\rho\) is defined by:
\[
\text{Cost}(\rho) = \sum_{i=1}^{|\sigma|} \text{Cost}(\rho(i-1), \rho(i))
\]

Offline Cost 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 sequence 
and its cost to an optimal offline algorithm.

Online Algorithm While offline algorithm may base its response to a request sequence on the 
knowledge of the entire request sequence, (and therefore can choose the optimal service se­
quence for every 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 servers) and a request, to the next 
configuration.
Randomized Online Algorithm  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).

Response Sequence We denote by $A_r(\sigma)$ the response sequence of the algorithm $A$ to the request sequence $\sigma$, using the random number $r$. The set of all the response sequences of $A$ to $\sigma$ is denoted by $S_A(\sigma)$. That is:

$$S_A(\sigma) \triangleq \{ \rho : \exists r \in \mathbb{R}, A_r(\sigma) = \rho \}$$

Probability of a Response Sequence For every $\rho \in S_A(\sigma)$ we define the probability that $\rho$ will be the response sequence of $A$ to $\sigma$ to be:

$$p_A(\sigma, \rho) \triangleq \mu \left( \{ r \in \mathbb{R}[0,1) : A_r(\sigma) = \rho \} \right)$$

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 $\rho$.

The Cost of a Request Sequence We define the average cost of the request sequence $\sigma$ for the online algorithm $A$ by:

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

The Competitive Ratio A common measure for the performance of online algorithms, which was introduced by Sleator and Tarjan [ST85], is the competitive ratio, defined for the algorithm $A$ to be:

$$w_C(A) \triangleq \limsup_{n \to \infty} \max_{\text{Cost}(\sigma) \geq n} \frac{C_A(\sigma)}{C_{\text{OPT}}(\sigma)}$$

If $w_C(A) = C$ for some $C \in \mathbb{N}$ then we say that $A$ is $C$-competitive. An alternative to define competitiveness is to say that $A$ is $C$-competitive if there is a constant $a$ such that $C_A(\sigma) \leq C \cdot C_{\text{OPT}}(\sigma) + a$ for every request sequence $\sigma$.

Worst-Case Amortized Cost The worst-case amortized cost of the algorithm $A$ is defined by:

$$M(A) \triangleq \limsup_{m \to \infty} \max_{\text{Cost}(\sigma) \geq m} \frac{C_A(\sigma)}{m}$$

The Max/Max Ratio A related measure which was defined by Ben-David and Borodin [BDB], is the max/max ratio, which is defined for the algorithm $A$ as:

$$w_M(A) \triangleq \frac{M(A)}{M(\text{OPT})}$$

where $\text{OPT}$ is the optimal offline algorithm. Thus, the worst-case amortized cost of the optimal offline algorithm serves as a normalizing factor to the worst-case amortized cost of the online algorithm.
Uncover Probability: Given a request sequence $\sigma$ for the randomized online algorithm $A$, we define the probability that a point $v \in V$ will not be occupied with a server at time $t$, as:

$$p_v(t) = \sum_{\sigma \in \mathcal{S}(\sigma)} p_A(\sigma, v)$$

We call this probability the uncover probability of the point $v$ at time $t$. At any time $0 \leq t \leq |\sigma|$ we define an increasing order on the uncovered probabilities of the points in $V$:

$$q_i(t) = \min\{|p_v(t) : v \in V\} - \{q_j(t) : 1 \leq j < i\}$$

Greedy Request Sequence: Given an algorithm $A$, the request sequence $\sigma$ is greedy if $q_v(t-1) = p_v(t)(t-1)$ for every $1 \leq t \leq |\sigma|$. That is, $\sigma$ is greedy if it always requests the point which has the highest uncover probability. Note that for the uniform $k$-server problem, if $\sigma$ is a greedy request sequence for the algorithm $A$, then $C_A(\sigma) = \sum_{t=1}^{n-1} q_v(t-1)$.

We now relate our model to other known models of online problems. The abstract model of the $k$-server, which is a special case of task systems [BLS87], was formulated in [MMS90]. They also formulated the competitive analysis for deterministic online algorithms, and suggested a definition of the competitive ratio for randomized online algorithms. This definition was indeed used in [FKL+88], achieving a $O(\log k)$-competitive randomized algorithm for paging. Ben-David et al. [BDBK+90] formulated a general framework for studying online algorithms. They defined a game between an online algorithm and an adversary, in which the adversary makes a sequence of requests, which are served one at a time by the online algorithm. They defined three types of adversaries and studied the power of randomization against each one of them. The type of adversary attributed here is an oblivious one, which must construct the request sequence in advance (based only on the description of the online algorithm but before any moves are made), and pays for it optimally. Note that this type of adversary is defined implicitly by the definition of the competitive ratio.

The one-way scanning of an infinite string of random bits appears in many works on random computation, such as [Nis90a, Nis90b, BNS89].

4 The Algorithm UNIFORM

In this section we prove that the lower bound of the worst-case amortized cost for the uniform $k$-server problem is $\frac{n-k}{n-1}$. Since this is also the worst-case amortized cost of the optimal offline algorithm (see [BDB]), it follows that the lower bound of the max/max ratio for the uniform $k$-server problem is 1. We present a randomized algorithm which we call UNIFORM that achieves this lower bound. We also show that this algorithm is not competitive, thus supplying an extreme evident to the inherent differences between the competitive ratio and the worst-case amortized cost (or the max/max ratio). A significant drawback of the algorithm UNIFORM is that its memory size depends on the space size (it needs $n \log n$ bits where $n$ is the number of points in the space).

Let us first prove the lower bound of the worst-case amortized cost for the uniform $k$-server problem.

Theorem 1: Every randomized online algorithm for the uniform $k$-server problem satisfies $M(A) \geq \frac{n-k}{n-1}$.
The proof of Theorem 2 is as follows:

1. If \( a \in W \) then a server is moved from \( u \) to \( v \).
2. Else a server is moved from \( u \) to \( v \) with probability \( 1 - \frac{k}{|W|} \), and a server from any of the other occupied points is moved to \( v \) with a probability \( \frac{k}{|W|} \).

We show that \( UNIFORM \) achieves the lower bound of the worst-case amortized cost for the uniform \( k \)-server problem.

**Theorem 2** For the uniform \( k \)-server problem \( M(UNIFORM) = \frac{n-k}{n-1} \).

**Proof** Let \( W_t \) denotes the content of the set \( W \) at time \( t \). Given a request sequence \( \sigma \), it suffices to prove that \( p_v(t) = \frac{|W_t| - k}{|W_t| - 1} \) for every \( v \in W_t - \{\sigma(t)\} \), and for every \( 0 \leq t < |\sigma| \). We prove this claim by induction on \( t \). Initially, \( W_0 \) is the initial location, so \( |W_0| = k \), and indeed \( p_v(0) = 0 \) for every \( v \in W_0 \). Assume that \( W_t \) satisfies the claim, and consider the following two cases:

1. \( \sigma(t + 1) = \sigma(t) \). In this case nothing changes so the claim remains true.
2. \( \sigma(t + 1) \in W_t - \{\sigma(t)\} \). In this case we get for every \( v \in W_{t+1} \):

\[
p_v(t + 1) = \begin{cases} p_{\sigma(t+1)}(t) & v = \sigma(t) \\ p_v(t) & v \in W_t - \{\sigma(t)\} \end{cases}
\]

\[
= \frac{|W_t| - k}{|W_t| - 1} = \frac{|W_{t+1}| - k}{|W_{t+1}| - 1}
\]

3. \( \sigma(t + 1) \not\in W_t \). In this case we get:

\[
p_{\sigma(t)}(t + 1) = \frac{|W_t| + 1 - k}{|W_t|} = \frac{|W_{t+1}| - k}{|W_{t+1}| - 1}
\]
and for every $v \in W_{t+1} - \{\sigma(t)\}$ we get:

$$p_v(t+1) = p_v(t) + (1 - p_v(t)) \cdot \frac{1}{|W_t|}$$

$$= \frac{|W_t| - k}{|W_t| - 1} + \left(1 - \frac{|W_t| - k}{|W_t| - 1}\right) \cdot \frac{1}{|W_t|}$$

$$= \frac{|W_{t+1}| - k}{|W_{t+1}| - 1}$$

Although the algorithm $UNIFORM$ has an optimal worst-case amortized cost, it is not even competitive, as the next theorem proves.

**Theorem 3** The algorithm is not competitive for any $k$-server problem.

**Proof** Consider a request sequence $\sigma$ consists of alternating requests to the points $u$ and $v$, where $v$ is included in the initial location, but $u$ is not. Obviously, the optimal offline algorithm can fix two of its servers at the points $u$ and $v$, so the cost of $\sigma$ is bounded by some constant (for any length of $\sigma$). Nevertheless, there is a probability of $\frac{1}{k}$ that $UNIFORM$ will move successively a server between $u$ and $v$, so the average cost $\sigma$ incurred depends on its length. This implies that $UNIFORM$ is not competitive.

5 The Algorithm RANDOM

In this section we analyze the simple and memoryless algorithm called $RANDOM$ [RS89]. First show its worst-case amortized cost for the cases $k = 2$ and $k = n - 1$ of the uniform $k$-server problem (where $n$ is the number of points in the space).

We start with the case $k = 2$, and analyze the behavior of the algorithm on greedy sequences. The following lemma shows how the uncover probabilities behave in this case.

**Lemma 4** Let $\sigma$ be a greedy request sequence for the algorithm $RANDOM$ and for the uniform 2-server problem. Then for every $0 \leq t \leq |\sigma|:

$$q(t) = 0$$

$$\left|q_i(t) - \left(1 - \frac{2^{n-i}}{2^{n-1} - 1}\right)\right| < 2^{-t} \quad \text{for} \quad 2 \leq i \leq n$$

**Proof** We prove that the claim is true for every $t$, by induction on $t$. The claim is true for $t = 0$, because all the probabilities are in the range $[0, 1]$, and there it at least one point which is not in the initial location (and thus has an uncover probability of 0). Assuming the claim is true for $t$, $0 \leq t < |\sigma|$, we prove it for $t + 1$. Let $\rho \in S_k(\sigma)$. Given the location $\rho(t)$ we claim that the location $\rho(t + 1)$ satisfies:

- $\sigma(t + 1) \in \rho(t + 1)$. 

• \( \sigma(t) \not\in \rho(t+1) \) iff \( \sigma(t+1) \not\in \rho(t) \) and \( \text{RANDOM} \) has chosen to move the server from \( \sigma(t) \) to \( \sigma(t+1) \).

• \( v \not\in \rho(t+1) \) (for \( v \not\in \{ \sigma(t), \sigma(t+1) \} \)) iff either \( v \not\in \rho(t) \) or \( v \in \rho(t) \) and \( \text{RANDOM} \) has chosen to move the server from \( v \) to \( \sigma(t+1) \).

Thus, the uncover probabilities at \( t+1 \) satisfy:

\[
p_v(t+1) = \begin{cases} 
0 & v = \sigma(t+1) \\
\frac{1}{2} p_v(t+1)(t) & v = \sigma(t) \\
\frac{1}{2} (1 + p_v(t)) & v \not\in \{ \sigma(t), \sigma(t+1) \}
\end{cases}
\]

Using the greediness of \( \sigma \) we get the following order of the uncover probabilities at \( t+1 \):

\[
q_v(t+1) = \begin{cases} 
0 & i = 1 \\
\frac{1}{2} q_v(t) & i = 2 \\
\frac{1}{2} (1 + q_{v-1}(t)) & 3 \leq i \leq \pi
\end{cases}
\]

Assuming that the order at \( t \) satisfies the claim, it can be verified that the order at \( t+1 \) satisfies the claim. \( \Box \)

**Lemma 5** Let \( \sigma \) be any request sequence for the uniform 2-server problem. Then there is a greedy request sequence \( \sigma' \) for the algorithm \( \text{RANDOM} \) which satisfies:

1. \( |\sigma'| = |\sigma| \).
2. \( C_{\text{RANDOM}}(\sigma) < C_{\text{RANDOM}}(\sigma') + 5 \).

**Proof** Define the request sequence \( \sigma \) to be \( t \)-greedy, if the subsequence \( \sigma(1,t+1) \) is a greedy request sequence. Given a request sequence \( \sigma \), we prove by induction on \( t \), that for every \( 0 \leq t \leq |\sigma| \) we can get a \( t \)-greedy request sequence \( \sigma_t \) of length \( |\sigma_t| \), satisfying

\[
C_{\text{RANDOM}}(\sigma) < C_{\text{RANDOM}}(\sigma_t) + 5 \sum_{i=0}^{t-1} 2^{-(i+1)}
\]

The base is obvious for \( \sigma_0 \equiv \sigma \). Assuming we have constructed \( \sigma_t \) we construct \( \sigma_{t+1} \). Consider the following two cases:

1. \( \sigma(t+1) \) satisfies \( p_v(t+1)(t) = q_v(t) \) in \( \sigma_t \). Then \( \sigma_{t+1} \equiv \sigma_t \) is \( (t+1) \)-greedy and satisfies inequality 3.
2. Let \( v \) be the point that satisfies \( p_v(t) = q_v(t) \) in \( \sigma_t \). There are two cases:
   (a) There is no subsequent request to the point \( v \). In this case we can simply replace the request \( \sigma(t+1) \) by \( v \) and get a \( (t+1) \)-greedy request sequence \( \sigma_{t+1} \) which satisfies \( C_{\text{RANDOM}}(\sigma_t) < C_{\text{RANDOM}}(\sigma_{t+1}) \).
   (b) Let \( i > 1 \) be the minimal index such that \( \sigma(t+i) = v \), and let removing the request to \( v \) at \( t+i+1 \) and inserting it at \( t+1 \). That is, \( \sigma_{t+i+1} = \sigma_t(t+1)\sigma(t+i+1, t+1)\sigma(t+i+1, |\sigma|+1) \). Now we can use the previous lemma in order to compare the average costs of \( \sigma_{t+i+1} \) to that of \( \sigma_t \):
Theorem 6

Theorem 6
Thus, for \( t = |\sigma| \) we get a greedy request sequence \( \sigma' = \sigma_{|\sigma|} \) which satisfies the conditions of the lemma.

Using the two previous lemmas we get the following theorem.

Theorem 6

For the uniform 2-server problem, \( M(RANDOM) = 1 - \frac{1}{2^{n-1}} \).

The second case we analyze is \( k = n - 1 \).

Lemma 7

Let \( \sigma \) be a greedy sequence for the algorithm RANDOM and for the uniform \((n-1)\)-server problem. Then for every \( 0 \leq t \leq |\sigma| \):

\[
\left| q_n(t) - \frac{2}{k+1} \right| < \frac{(k-1)}{k} \cdot \left\lfloor \frac{1}{k} \right\rfloor
\]

Proof

We prove that the claim is true for every \( t \), by induction on \( t \). The claim is true for \( t = 0 \), because all the probabilities are in the range \([0, 1] \), and there is one point which is not in the initial location (and thus has an uncover probability of 0). Assuming the claim is true for \( t, 0 \leq t < |\sigma| \), we prove it for \( t + 1 \). Let \( \rho \in S_A(v) \). Given the location \( \rho(t) \) we claim that the location \( \rho(t + 1) \) satisfies:

- \( \sigma(t + 1) \in \rho(t + 1) \).
- \( \sigma(t) \notin \rho(t + 1) \) if \( \sigma(t + 1) \notin \rho(t) \) and RANDOM has chosen to move the server from \( \sigma(t) \) to \( \sigma(t + 1) \).
- \( v \notin \rho(t + 1) \) (for \( v \notin \{\sigma(t), \sigma(t + 1)\} \)) iff either \( v \notin \rho(t) \) or \( \sigma(t + 1) \notin \rho(t) \) (and thus \( v \notin \rho(t) \)) and RANDOM has chosen to move the server from \( v \) to \( \sigma(t + 1) \).

Thus, the uncover probabilities at \( t + 1 \) satisfy:

\[
p_n(t + 1) = \begin{cases} 
0 & v = \sigma(t + 1) \\
\frac{1}{2}p_{\sigma(t+1)}(t) & v = \sigma(t) \\
p_n(t) + \frac{1}{2}p_{(t+1)}(t) & v \notin \{\sigma(t), \sigma(t + 1)\} 
\end{cases}
\]

Using the greediness of \( \sigma \) we get the following order of the uncover probabilities at \( t + 1 \):

\[
q_i(t + 1) = \begin{cases} 
0 & i = 1 \\
\frac{1}{2}q_n(t) & i = 2 \\
qu_{i-1}(t) + \frac{1}{2}q_n(t) & 3 \leq i \leq n
\end{cases}
\]
This implies that for \( t \geq k \), and for every \( 1 \leq i \leq n \):

\[
q_i(t) = \frac{1}{k} \sum_{j=1}^{i-1} q_n(t-j)
\]  

(4)

In particular, we get for \( i = n \):

\[
q_n(t) = \frac{1}{k} \sum_{j=1}^{k} q_n(t-j)
\]  

(5)

Summing equation 4 over \( 1 \leq i \leq n \) we get:

\[
\sum_{i=1}^{n} q_i(t) = \frac{1}{k} \sum_{j=1}^{k} (k-j+1)q_n(t-j)
\]

Equations 5 and 6 hold for \( t \geq k \). To make them hold for \( t < 0 \) we may assume the following initial conditions:

\[
q_n(t) = \begin{cases} 
0 & -k < t < 0 \\
1 & t = 0 
\end{cases}
\]

Now we can use equations 5 and 6 and the initial conditions to prove by induction on \( t \) that for every \(-k < t < \sigma\) the lemma holds. The base is obvious for every \(-k < t < 0\). Assuming the correctness of the claim for every \( t' \), \(-k \leq t' < t - 1\), where \( 0 < t < \sigma \), we prove its correctness for \( t \). Using equation 6 we get for every \( 0 < t < \sigma \):

\[
\sum_{j=1}^{k} (k-j+1)q_n(t-j) \approx k
\]

Equations 5 and 6 hold for \( t \geq k \). To make them hold for \( t \geq 0 \) we may assume the following initial conditions:

\[
q_n(t) = \begin{cases} 
0 & -k < t < 0 \\
1 & t = 0 
\end{cases}
\]

Now we can use equations 5 and 6 and the initial conditions to prove by induction on \( t \) that for every \(-k < t < \sigma\) the lemma holds. The base is obvious for every \(-k < t < 0\). Assuming the correctness of the claim for every \( t' \), \(-k \leq t' < t - 1\), where \( 0 < t < \sigma \), we prove its correctness for \( t \). Using equation 6 we get for every \( 0 < t < \sigma \):

\[
\sum_{j=1}^{k} (k-j+1) \left[ q_n(t-j) - \frac{2}{k+1} \right] = 0
\]

This implies that either all the elements of the sequence \( \{q_n(t-j) - \frac{2}{k+1}\}_{j=1}^{k} \) equal 0, or that at least two of them have opposite signs. We can use this fact, equation 5 and the induction hypothesis for \( t \leq t' \leq t - 1 \) to get:

\[
\left| q_n(t) - \frac{2}{k+1} \right| = \frac{1}{k} \sum_{j=1}^{k} \left| q_n(t-j) - \frac{2}{k+1} \right| < \frac{1}{k} (k-1) \left( \frac{k-1}{k} \right)^{\frac{t-1}{2}} \leq \left( \frac{k-1}{k} \right)^{\frac{t}{2}}
\]

\(\square\)

**Lemma 8.** Let \( \sigma \) be any request sequence for the uniform \((n-1)\)-server problem. Then there is a greedy request sequence \( \sigma' \) for the algorithm RANDOM which satisfies:

1. \( |\sigma'| = |\sigma| \).
2. $C_{\text{RANDOM}}(\sigma) < C_{\text{RANDOM}}(\sigma') + 3k^2$.

Proof Sketch The proof of the previous theorem can be extended such that for every $2 \leq i \leq n$ and every $0 \leq t \leq |\sigma|$:
\[
q_i(t) - \frac{i-2}{k} \cdot \frac{2}{k+1} < \left( \frac{k-1}{k} \right)^{t/2} \cdot \left( \frac{k}{k-1} \right)^{t/2}.
\]
Using this set of inequalities, we can apply the same technique used in the proof of lemma 4 for the case $k = 2$, to prove by induction on $t$ that for every $0 \leq t \leq |\sigma|$ there is a $t$-greedy request sequence of length $|\sigma|$ that satisfies:
\[
C_{\text{RANDOM}}(\sigma) < C_{\text{RANDOM}}(\sigma_0) + \sum_{i=0}^{t-1} \left( \frac{k-1}{k} \right)^{t/2} \cdot \left( \frac{k}{k-1} \right)^{t/2}.
\]
Thus, for $t = |\sigma|$ we get a greedy sequence $\sigma' = \sigma_{|\sigma|}$ which satisfies the condition of the lemma. □

Using the two previous lemmas we get the following theorem.

Theorem 9 For the uniform $(n-1)$-server problem, $M(\text{RANDOM}) = \frac{2}{k+1}$.

Remark The worst-case amortized analysis of the algorithm $\text{RANDOM}$ for the case $k = n-1$ can be used to show that in this case $w_C(\text{RANDOM}) = k$ (which implies that $w_C(\text{RANDOM}) \geq k$ for all the other cases). Nevertheless, Raghavan and Snir [RS89] have already show that $\text{RANDOM}$ is $k$-competitive for any $k$ (and even when considering a stronger adversary than the oblivious one considered in this work).

6 The Algorithm $\text{MARK}$

Fiat et al. [FKL+88] suggested the marking algorithm for the uniform $k$-server problem. The algorithm which needs $O(k \log k)$ memory bits, is $2H_k$-competitive in general, and $H_k$-competitive for the case $k = n-1$ (where $H_k = \sum_{i=1}^{k} \frac{1}{i}$ satisfies $\ln k < H_k < \ln k + 1$). We show that although this algorithm is better than the previous algorithms relative to the competitive ratio (recall that $\text{UNIFORM}$ is not even competitive, while $\text{RANDOM}$ is only $k$-competitive), it is worse than them relative to the worst-case amortized cost.

The algorithm, which we denote by $\text{MARK}$ manages a set of marked points, enforcing a partition of the request sequence into phases. A phase starts with no marked points and ends with $k$ marked points. During the phase every requested point is marked, and then its server is fixed till the end of the phase. If a point not occupied with server is requested, the algorithm chooses uniformly at random one of the unmarked points which are occupied with server, and move its server to the requested point.

Theorem 10 For a uniform space where $k \leq \frac{n}{2}$, $M(\text{MARK})=1$.

Proof Since $k \leq \frac{n}{2}$ we can construct a request sequence consists of phases of length $k$, such that there is no point which is requested in two successive phases. Thus every phase of this request sequence includes only requests to points which were not marked at the end of the previous phase. This implies that every request of this sequence costs exactly 1 to $\text{MARK}$. □
Thus, in the case $k = 2n \geq 4$, the worst-case amortized cost of $MARK$ is 1 while that of $UNIFORM$ is $1 - \frac{1}{n-1}$, and that of $RANDOM$ is $1 - \frac{1}{2n-2}$. Obviously, in this case and relative to the worst-case amortized cost $UNIFORM$ is better than $RANDOM$, and $RANDOM$ is better than $MARK$.

**Theorem 11** For a uniform space where $k < n < 2k$, $M(MARK) \approx \frac{n-k}{k}(H_k - H_{n-k} + 1)$.

**Proof** Fiat et al. [FKL+88] proved that if $l$ is the number of new points in a phase (points that were not marked in the previous phase) then the average cost of the phase is $l(H_k - H_l + 1)$. Since $k < n < 2k$, $l$ can be maximize to $l = n - k$, which implies an average cost of $(n - k)(H_k - H_{n-k} + 1)$ for each phase, and a worst-case amortized cost of $\frac{n-k}{k}(H_k - H_{n-k} + 1)$.

**Corollary 12** For the uniform $(n - 1)$-server problem, $M(MARK) = \frac{H_k}{k}$.

Thus, in the uniform space where $k = n - 1$, the worst-case amortized cost of $MARK$ is $O\left(\frac{\log k}{k}\right)$ while the worst-case amortized cost of $UNIFORM$ is $\frac{1}{k}$, and that of $RANDOM$ is $\frac{n}{k+1}$.

**Acknowledgment**

I want to thank my supervisor Shai Ben-David for introducing me the subject of online algorithms and for many helpful discussions.

**References**


