School of Business, Economics and Information Systems

Chair of Management Science/Operations

and Supply Chain Management

Seminar 39736

How randomized online algorithms can mitigate worst-case scenarios for the ski-rental problem and other dynamic decision problems

Seminar leader: Prof. Dr. Alena Otto

Supervising tutor: Catherine Lorenz

Edited by: Clara Gauer

Matriculation number: 106476

Course of studies: Wirtschaftsinformatik B.Sc.

Semester: 5

Address: Innstraße 22, 94032 Passau

Telephone: +491624959481

E-mail: gauer03@ads.uni-passau.de

Place, Date: Passau

Table of Contents

1. List of symbols

Basics of competitive analysis and adversary models

Symbol	Description
σ	Request sequence
ALG	Currently viewed online algorithm
OPT	Optimal offline algorithm
$ALG(\sigma)$	Cost incurred by ALG
$OPT(\sigma)$	Cost incurred by an optimal offline algorithm
a	constant
ALG_{rand}	Randomized online algorithm
$E[ALG_{rand}(\sigma)]$	Expected costs incurred by ALG_{rand}
c	Multiple by which optimal offline algorithm and online algorithm are compared

Ski Rental Problem

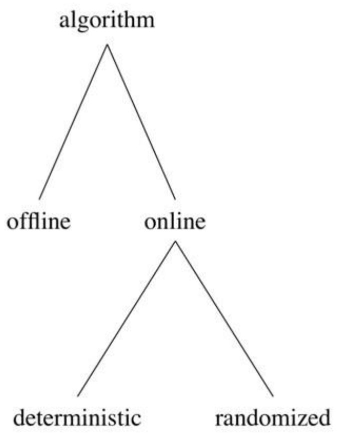
Symbol	Description
B	One time cost for buying skis
I_j	Sequence of days where the ski trip ends on day j
s	Day where the optimal strategy switches
j	Days of skiing
ALG_s	Algorithm that rents until day i and buys on day s
$OPT(I_j)$	Optimal sequence of days chosen by the optimal offline algorithm
$E[STRATEGIES]$	Expected result of each strategy
Pr_1	Probability that first strategy will be chosen
Pr_2	Probability that second strategy will be chosen
p	Probability that the randomized online algorithm decides to buy the skis on day j+1
q	Probability that the randomized online algorithm decides to continue on renting skis

Paging Problem

Symbol	Description
x	Current request
m_i	Marker bit, which protects a page from being evicted
H_k	kth harmonic number.
k	Amount of locations in the cache
$p_1, p_2, \dots, p_N$	Series of requests
i, j	Locations in the cache
s	Requests to stale pages
c	Serving c requests to clean pages costs c

2. Introduction

Algorithms can be either online or offline algorithms. In the case of online algorithms the input data is provided while the algorithm is running whereas in the case of offline algorithms all input data is provided at the start of the program.



Another way to classify algorithms is to differentiate between deterministic and randomized algorithms. The former always generates the same output given certain input data. Thus, the input data and the computation steps of the algorithm determine the result under all circumstances. The latter, however, will not always generate the same output given the same input data as certain computation steps are performed at random. As a result both the computation steps and the output will differ from run to run. (Cormen et al., 2009).

Here, we investigate whether randomization in online algorithms gives superior results compared to deterministic algorithms by achieving a more qualitative result and thus improving the performance. To this end we describe the basics of competitive analysis and compare deterministic to randomized algorithms. We then give examples from the literature that show the superiority of randomized over deterministic algorithms. Finally, the seminar thesis discusses the Ski Rental Problem and the Paging problem, which both shed some light on randomized vs. deterministic algorithms.

My contribution is the proof of the performance provided for the Ski Rental Problem, parts of the characteristics of the deterministic and randomized algorithm, the concrete behavior of the specific adversaries (p.12) as well as a discussion whether randomization is beneficial for finding a more qualitative result.

3. Basics of competitive analysis and adversary models

3.1 Performance of an online algorithm

When analyzing the information-theoretic complexity of an offline algorithm, one typically does a worst-case analysis. Such a worst-case analysis gives the maximum running time of an algorithm is for any input n. (Cormen et al, 2009). This technique assumes that the input n is available before the execution of the algorithm. However, in the case of online algorithms the input n is not known a priori but is given over time. As a consequence, worst-case analysis cannot be used to measure the quality of an online algorithm. Another reason why worst-case analysis is not useful in the context of Operations Research is that the field is interested in maximizing the quality of the output which differs from the main objective of worst-case analysis.

That's why competitive analysis, the method of choice to measure the performance of an online algorithm. Competitive analysis makes use of an "adversary", which is an entity that tries to define an input sequence that maximizes the worst-case performance of an algorithm, is being introduced in the following (Borodin & El-Yaniv, 1998) (Azar, 2005).

3.2 Competitive analysis and competitive ratio

From an Operations Research perspective, we consider an optimization problem which is a minimization problem. Competitive analysis measures the performance of online algorithms, instances of the minimization problems, relative to an optimal offline algorithm with the same input n (Borodin & El-Yaniv, 1998). Each input and corresponding actions has an associated cost. The comparison between both algorithms is made by choosing the maximum of the ratio between the cost incurred by the online algorithm and the cost incurred by an optimal offline algorithm on the same input sequence (Karp, 1992).

The competitive ratio is the key metric used in competitive analysis. Given a request sequence σ, let $ALG(\sigma)$ denote the cost incurred by ALG and let $OPT(\sigma)$ denote the cost incurred by an optimal offline algorithm OPT (Albers, 1996). A deterministic algorithm ALG is called c-competitive, if there exists a constant a for all request sequences σ such that,

$$ALG(\sigma) \leq c * OPT(\sigma) + a.$$

Given a randomized online algorithm ALG_{rand}, let $E[ALG_{rand}(\sigma)]$denote the expected costs incurred by ALG_{rand} and let $OPT(\sigma)$denote the cost incurred by an optimal offline algorithm. The expectation is over the random choice made by the online algorithm. A randomized online algorithm is called c-competitive against any oblivious adversary if there exists a constant a for all request sequences σ such that,

$$E[ALG_{rand}(\sigma)] \leq c * OPT(\sigma) + a.$$

(Reingold et al., (1994), Gupta (2020), Albers (1996)).

The online algorithm with the lowest competitive ratio is optimal (Karp, 1992). Generally, the smaller the value of c the more efficient the online algorithm.

3.3 Adversary

In the context of online algorithms, an adversary is typically modeled as an entity that generates input sequences online. The adversary wants the algorithm to perform poorly and thus wants the cost of the solution found by the online algorithm to be maximal (Karp, 1992). Since online algorithms are measured by a competitive ratio, an adversaries' aim is to design a input sequence which results in the worst-case competitive ratio. The adversary is assumed to have full knowledge of the algorithm and its execution steps (Borodin & El-Yaniv, 1998).

There are three different types of adversaries, the oblivious adversary, the adaptive online adversary as well as the adaptive offline adversary. They differ in having different information available before generating a request sequence.

The **oblivious adversary** knows the algorithm but has to state the entire request sequence before execution. The outcomes of the algorithm are unknown to the oblivious adversary.

The **adaptive online adversary** knows the algorithm and chooses the next request based on the algorithm's answer to previous one (the last online decision) but states the next request sequence immediately. Thus, the adaptive offline adversary can partially adapt its request based on the outcome of the decisions of the algorithm, because a new request element has to be stated immediately after the online decision is made.

The **adaptive offline adversary** knows the algorithm and is able to observe every action made by the online algorithm before choosing the next request. This means that a request sequence is provided, whereupon the online algorithm makes a decision. This is done until the end of the request sequence is achieved, so that the adaptive offline adversary can now place his own actions, such that it achieves the worst-case input sequence for the algorithm (Karp, 1992).

3.4 Relationship between adversaries

If the algorithm is deterministic, no adversary type has a competitive edge over the other because there is no advantage in observing the online algorithms' decisions since those can easily be computed by simulating the algorithm (Karp, 1992).

If the algorithm is randomized, there exists a hierarchy concerning the power of an algorithm between the adversary types, which is formally:

$$\text{Adaptive offline adversary} \succ \text{Adaptive online adversary} \succ \text{Oblivious adversary}$$

The adaptive offline adversary is the most powerful adversary in case of a randomized algorithm, the adaptive online adversary the second most powerful and the oblivious adversary is the least powerful adversary (Karp, 1992). The adaptive offline adversary is so strong that randomization adds no power against it, which means that it cannot guarantee a better competitive ratio than the one achieved by deterministic algorithms.

4. Comparison between online deterministic and online randomized algorithms

In the following, a comparison will be made between deterministic online algorithms and randomized online algorithms. First, a short definition of the most important characteristics is defined.

The efficiency of the algorithm is influenced by the size and complexity of the input sequence, determining the feasible number of input elements. An algorithm is called consistent if it produces the same output for the same input across multiple executions. The repeatability of an algorithm states whether an algorithm can be reproduced and validated.

Simplicity refers to the use of understandable solutions that avoid unnecessary complexity. Efficiency focuses on optimizing computational resources such as time or memory.

	Deterministic online algorithm	Randomized online algorithm
Definition	"A deterministic algorithm is one that always produces the same output for a given input. In other words, its behavior is completely determined by its input and the algorithm itself. The input arrives over time and instant response is required" (Cormen et al., 2009).	"We call an algorithm randomized if its behavior is determined not only by its input but also by values produced by a random-number- Generator while the input arrives a little at a time and instant response is required" (Cormen et. al, 2009). A randomized algorithm can be viewed as a probability distribution over a set of deterministic algorithms. (Gupta, 2020)
Size and complexity of the input sequence	Deterministic online algorithms often require a complete knowledge of the input's size or some other parameters to make decisions. They may need to maintain a data structure or use memory proportional to the input size. (Cormen et al., 2009)	Randomized online algorithms can handle large and complex input sequences better because they are often designed to perform random operations on the input sequence or to operate on the data they have seen so far. This typically requires less computational complexity and less memory (Cormen et al. (2009) , Dasgupta et al. (2006))
Consistency and repeatability	Deterministic online algorithms are used when a high consistency and repeatability of the solution must be provided. Consistency ensures that the algorithm's behavior is explainable, predictable, and stable.	The outputs of a randomized algorithms can vary each time an algorithm runs, sometimes not leading to an optimal solution. This can also lead to unstable, unexplainable, and unpredictable results.

Simplicity	In some cases, deterministic online algorithms can be easier to design because they are based on predefined rules which are consistent throughout the execution. Thus, every step of the algorithm is understandable because no unknown elements are included.	A randomized algorithm can be simpler to analyze and design than a deterministic algorithm. On the other hand, the implementation of randomization includes probabilistic reasoning and randomization techniques, which can make randomized online algorithms more difficult to design.
Efficiency	Deterministic online algorithms can be more efficient than randomized algorithms that require a large number of random samples or if the distribution of the random samples is not representative of the problem. However, deterministic algorithms may experience exponential time complexities.	Randomized online algorithms are often designed to operate on a subset of the input sequence, which can reduce the computational complexity (Motwani et al., 1995).
Adversary	The chosen input sequence by the adversary will generate the same output when run multiple times. Therefore, it is simple for an adversary to choose an input sequence, for which the costs of the algorithm are maximal (Borodin et al., 2019).	When regarding randomized online algorithms, the adversary knows the algorithm but not the order in which the execution steps are taken and which random decisions are made by the algorithm. Therefore, it is more difficult for an adversary to select an input sequence in order to maximize the costs of the algorithm. If a randomized problem has multiple deterministic strategies from which to choose, a random choice of a deterministic strategy is sufficient in order to gain competitive quality (Borodin et al., 2019)
Examples	Binary search	Quicksort algorithm

5. Examples of online algorithms which benefit from randomization

Based on literature search, we give a few examples for which randomized online algorithms perform better than their deterministic counterparts.

5.1 k-Server Problem

First defined by Manasse, McGeogh and Sleato (Manasse & McGeogh & Sleato, 1988), the k-Server problem is an online optimization problem. It involves multiple servers serving requests from a metric space, with a cost incurred for moving a server to a request. The goal is to minimize the total cost of serving all requests (Bartal, 1998).

The randomized greedy algorithm assigns incoming requests to the closest server, but randomly selects a server from the set of closest servers with the probability of selecting a server proportional to its distance to the request. (Rajaraman, et al., 2011). The closest servers are such with the minimum distance to a given request. A generalization of the k-Server problem was made by Borodin, Linial and Saks (Borodin & Linial & Saks, 1992).

According to Borodin (Borodin et al., 2019), a randomized algorithm can achieve a competitive ratio of O(log k) against oblivious adversaries. The deterministic algorithm has a competitive ratio of k.

5.2 Bipartite matching problem

"The bipartite matching problem involves finding a maximum matching in a bipartite graph, where a matching is a set of edges that do not share any vertices." (Karp & Sipser, 1981). The randomized algorithm for bipartite matching is based on augmenting paths. It works by randomly choosing an unmatched vertex on one side of the bipartite graph and following a path of alternating matched and unmatched edges until it reaches an unmatched vertex on the other side (Karp et al., 1981). The first to show that randomized online algorithms achieve the best possible performance of O(log n), if n is the amount of vertices in a graph, are Karp, Vazirani and Vazirani (Karp &Vazirani & Vazirani, 1990).

This is better than the competitive ratio of the deterministic counterparts ratio of $O(n^2)$.

The use of randomization to improve the performance of algorithms for matching problems was pioneered by Edmonds (1965) in the 1960s.

Interestingly, the first algorithm for bipartite matching, the Hungarian algorithm, was a deterministic algorithm, introduced by Kuhn (1955).

5.3 Interval coloring problem

The interval coloring problem involves assigning colors to a set of intervals such that overlapping intervals have different colors and the minimum number of colors is used.

One algorithm for interval coloring is the First Fit Decreasing (FFD) algorithm, which sorts the intervals in decreasing order of size and assigns each interval the smallest available color.

If n is the number of intervals, the deterministic First Fit Decreasing (FFD) algorithm has a worst-cast competitive ratio of O(log n). Randomization can be implemented by rearranging the order of items. This leads to an algorithm called Randomized First Fit (RFF), which results in a better competitive ratio of O(log log n). (Kleinberg & Tardos, 2006).

The interval coloring problem is a classic problem in computer science, dating back to the 1970s (Even et al., 1976). It has many real-world applications, including scheduling and frequency assignment in wireless communication systems (Tang & Li, 2011).

In addition to randomized algorithms like RFF, several other approaches have been proposed for interval coloring, including integer linear programming (Lovász & Schrijver, 1980) and greedy algorithms based on the notion of conflict graphs (Garey & Johnson, 1979).

Thus, the randomized version of the k-Server problem, the bipartite matching problem as well as the Interval coloring problem are superior to their deterministic counterpart when comparing with the competitive ratio.

6. Ski Rental Problem

After the overview based on literature research we would like to investigate the Ski Rental problem in more detail. The Ski Rental problem is a classical problem in computer science, which has a broad range of applications. In the following, we will introduce an optimal offline algorithm, a deterministic online algorithm as well as a randomized online algorithm. In the end, we will provide a competitive ratio will.

6.1 Problem Statement

The problem can be stated as follows: Is it more cost-efficient to buy or to rent skis depending on the number of days the skis are in use? (Levi & Patt-Shamir, 2015).

Option = $\begin{cases} Rent \\ Buy \end{cases}$

Costs = $\begin{cases} 1 \text{ ongoing usage cost per unit for renting} \\ B \text{ one-time cost fur buying, } \mathrm{B} > 1 \end{cases}$

Variables = $\begin{cases} I_j \text{ sequence of days where the ski trip ends on day j} \\ s \text{ day where optimal strategy switches} \\ j \text{ days of skiing} \end{cases}$

Algorithms =$\{ALG_s$ algorithm that rents until day i and buys on day s

The first option is to rent the skis. Assume that the cost for renting a pair of skis is 1 per day. In the short-term, renting is a cost-efficient strategy because the costs for renting skis are lower than the costs for buying skis. Nevertheless, as time progresses, the renting costs accumulate and eventually exceed the buying costs.

This makes the second option, buying skis, more attractive. This option requires a one-time payment, which is higher than the renting costs of 1. If the skis are used often, this option is cost-effective. Otherwise, it would have been cheaper to rent the skis for the number of days they are used.

6.2 Offline algorithm

An offline strategy can compute the days k for which a person will rent the skis.

If $j < B$, the optimal strategy is to rent the skis. If $j \geq B$, the person optimally buys the skis on the first day (Borodin & Pankratov, 2019). Therefore, the minimal costs are $min(j, B)$ and the function can be described as $ALG_s(I_j) = (s - 1 + B) * 1_{s \leq j} + j * 1_{s > j}$. This function is being minimized.

This strategy is graphically depicted, the blue line representing the accumulating costs over time and the orange dot s, so the day when the optimal strategy switches from renting to buying.

B is an integer > 1 by assumption. Now for 1 < s < B, we take $j = \left\lfloor (s - 1 + B) / \left(2 - \frac{1}{B}\right) \right\rfloor$, such that

$$\frac{ALG_s(I_j)}{OPT(I_j)} \geq 2 - \frac{1}{B}$$

6.3 Deterministic online algorithm

A deterministic online strategy implies that the adversary knows if a person bought or rented a pair of skis. The adversary also knows how a person has decided in the previous days. Thus, the adversary knows how you behave in the future. Based on this information, the adversary is able to predict how a person is going to decide for days j + 1. Recall that there is no hierarchy between adversaries in deterministic strategies, as a weak adversary can simulate a strong adversary (Borodin & Pankratov, 2019).

6.4 Randomized online algorithm

For a randomized strategy, we assume an oblivious adversary, who is not aware of the partial decisions of the person but knows the execution steps of the algorithm as well as the input. Thus, the oblivious adversary states the entire request sequence upfront. They submit the request and receive a rental plan from the algorithm, indicating which equipment to rent for each day. The adversary doesn't know the outcomes of the algorithm's decisions until they receive the rental plan. [1][2]

The assumption of an oblivious adversary is valid because the adversary does not know how an individual will decide each day influenced (Borodin & Pankratov, 2019). Randomized algorithms against an adaptive offline adversary have the same power as an oblivious adversary against deterministic algorithms, who can simulate the deterministic algorithm (Gupta, 2020), (Ramanarayana, 2011).

Let us consider a simple randomized algorithm. We have two different strategies, which are

$$STRATEGIES = \begin{cases} 1 \; buy \; on \; day \; B \\ 2 \; buy \; on \; day \; \frac{3}{4}B \end{cases}$$

[1] In this case, the adaptive online adversary knows the ski rental algorithm. The adversary starts by submitting an initial request and receiving a rental plan for the first day. Based on the algorithm's answer, the adversary adapts its next request. The adversary immediately states the modified request sequence, including the updated preferences for the next day and receiving a new rental plan.

[2] An adaptive offline adversary knows the ski rental algorithm and can observe all actions made by the online algorithm. The adversary provides a complete request upfront. The online algorithm processes each request and makes decisions accordingly. The adversary can observe the rental plans recommended by the algorithm for each day. Once the request sequence has been fully processed, the adaptive offline adversary can take its own actions and modify the rental plan for specific days based on their observations.

We flip a coin and both possible actions have the possibility to occur half of the time, because

$$E[STRATEGIES] = 1 * Pr_1 + 0 * Pr_2 = 1 * \frac{1}{2} + 0 * \frac{1}{2} = \frac{1}{2}.$$

While deterministic algorithms cannot beat 2-$\frac{1}{B}$, randomized algorithms can when using competitive ratio.

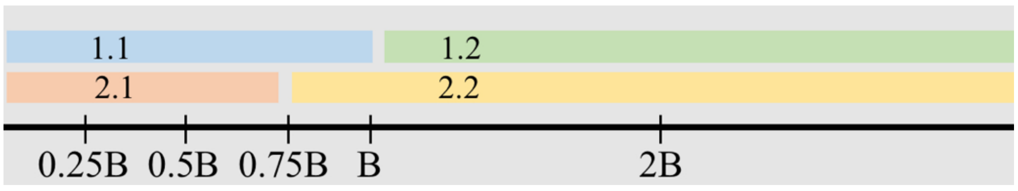
6.5 Proof of competitiveness and competitive ratio

Conjecture. The randomized counterpart is $\frac{15}{8}B$ – competitive.

Proof.

Let's discuss the strategies provided in the definition of a randomized online algorithm in more detail.

1. We proceed as before and buy the skis on day B. This means that the strategy of the deterministic online algorithm is pursued. Graphically, this strategy is represented by the blue and the green bar.
2. We purchase the skis early on day $\frac{3}{4}B$. Graphically, this strategy is represented by the orange and the yellow bar.



In the following, each partial strategy is discussed in detail. Remember that strategy 1 and 2 are equally likely to apply after flipping a coin. Thus, we assume that the sequence does not adapt to the coin flips and therefore does not depend on it.

1.1 If $j < B$, the optimal strategy for the skier is to rent the skis. $OPT(\sigma)$ is then the accumulated renting costs $< B$.

1.2 If $j \geq B$, then $OPT(\sigma) = B$. In brief, if the skier plans to ski for B days or more, then the optimal strategy is to buy the equipment on the first day. Given that the renting costs are 1, they will eventually exceed the buying costs of B if the skier decides to ski for more than B days.

2.1 If $j < \frac{3}{4}B$, we always rent skis, so $E[ALG(\sigma)] = OPT(\sigma) = j$. The optimal strategy for the skier is to rent skis for all j days. In this case, the expected cost of the randomized algorithm is equal to the cost of the offline counterpart algorithm.

2.2 If $j \geq \frac{3}{4}B$, the optimal strategy for the skier is to buy skis on the first day.

Thus, this randomized online algorithm always buys skis. In the first strategy, it rents in the time period of $[1;\ B - 1]$ and eventually buys the skis on day $j = B$. In the second strategy, renting in the period of $\left[1; \frac{3}{4}B - 1\right]$, until it buys the skis on day $\frac{3}{4}B$, depending on the result of the coin flip.

We define p as the probability that the randomized online algorithm decides to buy the equipment on day $j + 1$, and q to be the probability that the randomized online algorithm continues to rent skis for the remaining days, until the number of days exceed the buying costs. In total, $p + q = 1$ since the skier must decide between one of the two strategies.

An adversary now seeks to maximize the cost of the algorithm. Beforehand, we can assume that $p = \frac{1}{2}$ and $q = \frac{1}{2}$, since there are only two strategies. We compute the expected costs of the algorithm as follows:

$$E[ALG(\sigma)] = \frac{1}{2}(B - 1 + B) + \frac{1}{2}\left(\frac{3}{4}B - 1 + B\right) = \left(\frac{1}{2}B - \frac{1}{2} + \frac{1}{2}B\right) + \left(\frac{3}{8}B - \frac{1}{2} + \frac{1}{2}B\right) =$$

$$\left(B - \frac{1}{2}\right) + \left(\frac{3}{8}B - \frac{1}{2} + \frac{4}{8}B\right) = \frac{8}{8}B - 1 + \frac{7}{8}B \leq \frac{15}{8}B = \frac{15}{8}c(OPT(\sigma))$$

Since $\frac{15}{8}B$ is smaller than $2 - \frac{1}{B}$, a randomized algorithm does indeed beat the competitive ratio of a deterministic online algorithm and the upper bound of the randomized online algorithm is $\frac{15}{8}B$ (Kesselheim, 2018/19) (Gupta, 2020).

7. Paging problem

The paging problem focuses on effective memory management in paged memory systems. These systems partition the physical and virtual memory into equal-sized blocks. The main challenge lies in deciding which pages should be retained in the physical memory to endure efficient operations without unnecessary space consumption. The difficulty arises from the need to identify and remove pages that have a low probability of being used in the near future, while also avoiding complex implementation problems (Chu & Opderbeck, 1972).

No costs incur if a page of the physical memory (a hit) is demanded. In contrast, if a page of the virtual memory (a fault) is requested, the page is being moved to physical memory at unit cost (Irani, 1998). The algorithm is then obligated to decide which page of the physical algorithm is replaced with the requested page. Thus, this is an optimization problem which tries to minimize the occurrence of page faults.

7.1 Optimal offline algorithm

An optimal offline algorithm is the Longest Forward Distance algorithm (LFD) for the paging problem. Let x be the current request. Then, LFD is defined as:

1. If x is in the cache, do nothing.
2. If x is not in the cache, copy x into the cache, and eject that page whose next request is farthest in the future. In case two or more cache pages are never requested in the future, eject one of those arbitrarily.

If a request is farthest in the future, the page will least likely be needed, which makes it appropriate to remove this page from the cache. A page has to be removed in order to make space for the new requested page, which will be loaded in the cache.

7.2 Online deterministic algorithm

The goal of an online paging algorithm is to minimize the number of page faults that occur. They differ in the choice of a page to evict (Karp, 1992). Since the algorithm doesn´t know which pages are going to be used beforehand, alternative strategies can be pursued.

1. LRU (Least recently used): The least recently used algorithm replaces the page in the memory that has not been used for the longest period of time (Stallings, W., 2009)
2. FIFO (First-In, First-Out): The idea is to replace the oldest page in main memory. Thus, the page which has been in the main memory for the greatest period of time is being replaced (Shastri, S. & Sharma, A. & Mansotra, V., 2016).

If k is the number of cache pages, LRU and FIFO are both k-competitive, which is shown by Borodin, A. et. al (1992).

An adversary of the deterministic algorithm can simulate the paging algorithm if needed, which is why the adversary only needs to output a deterministic sequence of requests.

If the algorithm is randomized, there are two possible adversaries who choose the sequence of page requests.

An **oblivious adversary** produces a sequence of paging requests that is independent of the behavior of the paging algorithm.

An **adaptive adversary** produces page requests that depend on all the actions of the paging algorithms up to the current time, and those depend on random choices made by the algorithm which are not known ahead of time (Motwani, R. & Raghavan, P., 1996).

We consider an oblivious adversary because operating systems can not cause cache misses by only using the cache contents.

7.3 Randomized online algorithm

In the following, the pseudocode of the algorithm MARKER is provided.
The MARKER algorithm uses a single bit called the marker bit m_i to the ith location.

```
1. Loop
    a. Set marker bits $m_i = 0$ for $i = 1, \dots, k$
    b. Do until all $m_i = 1$ for $i = 1, \dots, k$
        i. Accept a page request
        ii. If requested page is in cache location i
            1. Set $m_i = 1$
        iii. Otherwise
            1. Choose an unmarked cache location j at random
            2. Evict cache location j and bring new page into j
            3. Set $m_j = 1$
```

A marker bit protects cache pages from being evicted. Initially, all the cache pages are unprotected. Thus, every cache hit causes that page to be protected and a cache miss causes an un-

protected page to be thrown out. Consequently, a new page can be inserted and protected immediately.
(Achlioptas, D. & Chrobak, M. & Noga, J., 1998) (Motwani, R. & Raghavan, P., 1996).

7.4 Proof of competitiveness and competitive ratio

Conjecture. The competitive ratio of algorithm MARKER is cH_K , where H_K is the kth harmonic number.

Proof.

The cache has k locations and there is a series of requests $p_1, p_2, \dots, p_N$.

We assume that MARKER already has a fault on the first request p_1.

MARKER divides the request sequences into phases. A phase starting with p_i ends with p_j where $j, j > i$, is the smallest integer such that

$$p_i, p_{i+1}, \dots, p_{j+1}$$

contains $k + 1$ distinct pages. At the end of a phase all pages in fast memory will be marked.

Call a page stale if it is unmarked but was marked in the previous phase. Call a page stale if it is neither stale nor marked.

Let c be the number of clean pages requested in the phase.

We will show that the expected number of faults made by MARKER is at most cH_k.

Serving c requests to clean pages costs c. There are $s = k - c \leq k - 1$ requests to stale pages. For $i = 1, \dots, s$, we compute the expected cost of the i-th request to a stale page. Let $c(i)$ be the number of clean pages that were requested in the phase immediately before the i-th request to a stale page and let $s(i)$denote the number of stale pages that remain before the i-th request to a stale page.

When MARKER serves the i-th request to a stale page, exactly $s(i) - c(i)$ of the $s(i)$pages are in fast memory, each of them with equal probability. Thus the expected cost of the request is

$$\frac{s(i)-c(i)}{s(i)} * 0 + \frac{c(i)}{s(i)} * 1 \leq \frac{c}{s(i)} = \frac{c}{k - i + 1}$$

The last equation follows because $s(i) = k-(i-1)$. The total expected cost for serving requests to stale pages is

$$\sum_{i=1}^{s} \frac{c}{k + 1 - i} \leq \sum_{i=2}^{k} \frac{c}{i} = c(H_k - 1)$$

In conclusion, MARKER's total expected costs are cH_k (Fiat, A. et al., 1991) (Manasse, M.S. & Sleator, D.D., 1991) (Motwani, 1995).

8. Conclusion

8.1 Real-life applications of randomization in production and logistics

Some problems in production and logistics need to update their information steadily or must perform one execution before retrieving the next bit of input.

One of many applications, inventory management, is the process of ordering, storing and selling a company's inventory. It depends heavily on demand, supply chain disruptions and other entities. The goal is to optimize inventory levels and make better decisions under uncertainty. Randomization typically increases the algorithm´s skill to adapt to external factors. By incorporating random elements, the algorithm becomes more flexible and capable of responding to unpredictable events (Zhang, X. & Huang, S. & Wan, Z. (2016), Davoodi, S. et al. (2016)).

Moreover, routing problems aim to optimize routes by considering travel costs and travel time. Randomization can help in the example of a „Randomized Greedy Algorithm", where one among the set of x best decisions is randomly selected This random selection is based on a random variable, introducing an element of chance into the decision. By doing so, the algorithm avoids getting trapped in local optima or consistently making the same choices, which can lead to suboptimal solutions. Randomization allows the algorithm to explore different options and increase the chances of finding better solutions that may not have been initially apparent based on deterministic criteria alone. (Tsang, M. & Shehadeh, K. (2023), Borodin, A. et al. (2019)).

Other problems like Quality Control and Scheduling problems can also benefit from randomization (Martins, L., 2021). In quality control and scheduling problems, randomization improves unbiased sampling, fairness, efficiency, and the ability to handle uncertainties.

8.2 Real-life applications of randomization on the Internet and the WWW

Randomized online algorithms have significant benefits for the Internet from an Operations Research perspective, primarily in terms of the quality of their output. The Internet generates vast amounts of data, and randomized online algorithms excel in processing this data efficiently. However, their advantage lies not only in speed but also in their ability to provide high-quality solutions in dynamic and uncertain environments. The dynamic nature of the Internet, with varying usage patterns and traffic fluctuations, can be effectively addressed by randomized online algorithms. (Lewis, H. R. & Papadimitriou, C. H., 1998).

Load Balancing is an exemplary problem in the context of the Internet. It is often necessary to distribute incoming requests to different servers to balance the load. Randomization is implemented by randomly assigning requests to servers. It benefits the algorithm by preventing overloading of specific servers. Additionally, response times are diminished (Vashistha, J. & Jayswal, A. K., 2013).

Another problem's subjective is to suggest items to users. Selecting items randomly helps reduce the popularity bias and improving the recommendation diversity. Popularity bias refers to the tendency of a recommendation system to favor popular items over less popular ones. In the end, users benefit more from the recommendations (Nadeem et al., 2022). Consider a music streaming platform, which would not only suggest the most popular hits but also newer and lesser-known artists and genres. As a result, users receive diverse music recommendations that which match their individual music taste.

8.3 Enhancing Performance of Online Algorithms through Randomization and general observations

By implementing randomization, online algorithms can adapt to changing conditions, explore different possibilities, and optimize their objectives within the constraints of limited memory and varying results.

Randomization can mitigate worst-case performance. In online settings, adversaries can strategically design inputs to exploit deterministic patterns. While worst-case inputs can occur, the probability of encountering them is reduced by randomization, since randomness adds variability and uncertainty to the algorithm's choices. This makes it harder for adversaries to predict its responses.

Sometimes, randomized algorithms achieve a better worst-case performance than their deterministic counterpart. Nonetheless, randomization can increase the complexity of analyzing the algorithm. Randomized online algorithms can provide a better performance than a deterministic algorithm, even though its performance is less predictable. Dependent on the application of the algorithm, the trade-off can be in favor of a deterministic online algorithms, where a predictable performance is mandatory, and a worse performance can be tolerated.

Randomization can be a useful tool for online algorithms if the objective is to maximize or minimize a specific function while dealing with limited memory and obtaining different results each time.

By incorporating randomness, online algorithms can explore alternative actions, strategies, or solution spaces that they might not have considered deterministically. Randomization encourages exploration by allowing the algorithm to take diverse actions and try out different strategies. This allows them to learn from different outcomes, leading to improved adaptability and better decision-making in uncertain environments, while optimizing the objective function. Furthermore, randomness can prevent the algorithm from becoming trapped in suboptimal solutions or deterministic patterns. For example, an online algorithm for inventory management can simulate many different situations by generating random values for customer demand or supplier availability. Thus, the performance of strategies under various and changing circumstances can easily be assessed.

Without randomization, an algorithm may overly rely on a single strategy or deterministic choices, limiting its ability to adapt to new situations or discover superior alternatives.

9. Bibliography

Books

Fiat, Amos & Woeginger, Gerhard J. (2005). *Online Algorithms.* Springer. https://doi.org/10.1007/BFb0029562

Borodin, Allan & El-Yaniv, Ran (1998). *Online Computation and Competitive Analysis.* Cambridge University Press (28. April 1998)

Bubley, Russ (1998). *Randomized Algorithms: Approximation, Generation and Complexity*. Springer.

Soltys, Michael (2018). *An Introduction to the Analysis of Algorithms* (3rd edition). World Scientific.

Cormen, Thomas H. & Leiserson, Charles E. & Rivest, Ronald L. & Stein, Clifford (2009). *Introduction to Algorithms* (3rd edition). The MIT Press.

Motwani, R. & Raghavan, P. (1995). *Randomized algorithms.* Cambridge University Press.

Dasgupta, S., Papadimitriou, C. H., & Vazirani, U. V. (2006). *Algorithms*. McGraw-Hill.

Rubinstein, R. Y., & Kroese, D. P. (2017). *Simulation and the Monte Carlo method* (3rd edition). John Wiley & Sons.

Rajaraman, A., & Ullman, J. D. (2011). *Mining of massive datasets*. Cambridge University Press.

Kleinberg, J., & Tardos, É. (2006). *Algorithm design*. Pearson Education India.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company.

Stallings, W. (2000). Operating systems: internals and design principles (9th edition). Pearson.

Chu, W. & Opderbeck, H. (1972). *The page fault frequency replacement algorithm. AFIPS ´72.* 10.1145/1479992.1480077

Lewis, H. R. & Papadimitriou, C. H. (1998). *Elements of the Theory of Computation*. Harvard, University of California, Berkeley.

Published Dissertation or Thesis Reference

Karp, Richard M. (1992). *On-Line Algorithms Versus Off-Line Algorithms: How Much is it Worth to Know the Future?* TR-92-044. 13 pages. https://www1.icsi.berkeley.edu/pubs/techreports/TR-92-044.pdf

Karp, R. M., & Sipser, M. (1981). *Maximum matching in sparse random graphs*. Proceedings of the 22nd Annual Symposium on Foundations of Computer Science (FOCS 1981), 364-375. doi: 10.1109/SFCS.1981.38)

Edmonds, J. (1965). Paths, Trees, and Flowers. Canadian Journal of Mathematics, 17(3), 449-467.

Lovász, L., & Schrijver, A. (1980). *Cones of matrices and set-functions and 0-1 optimization.* SIAM Journal on Optimization, 11(3), 703-727.

Kuhn, H. W. (1955). *The Hungarian Method for the assignment problem.* Naval Research Logistics Quarterly, 2(1-2), 83-97.

Journal

Reingold, Nick & Westbrook, Jeffery & Sleator, Daniel D. (1994). *Randomized Competitive Algorithms for the List Update Problem*. *Algorithmica, 11,* 15-32.

Koutsoupias, Elias & Papadimitriou, Christos H. (2000). *Beyond competitive analysis*. *Siam Journal on Computing, 30,* 300-317

Bartal, Y. (1998). On approximating arbitrary metrics by tree metrics. Journal of Computer and System Sciences, 58(3), 397-421.

Alon, N., Azar, Y., Buchbinder, N., & Naor, J. S. (2008). *The Online Strip Packing Problem*. SIAM Journal on Computing, 38(6), 2027-2045.

Tang, J., & Li, J. (2011). *Scalable influence maximization for prevalent viral marketing in large-scale social networks*. Proceedings of the 2011 IEEE 11th International Conference on Data Mining (ICDM), 1029-1034.

Shastri, S. & Sharma, A. & Manostra, V. (2016). *Study of Page Replacement Algorithms and their analysis with C#.*The International Journal of Engineering and Sciences, 5, 2319 – 180

Borodin, A. & Irani, S. & Raghavan, P. & Schieber, B. (1992). *Competitive Paging with Locality of Reference.* Journal of Computer and System Sciences, 50, 244-258

Zhang, X. & Huang, S. & Wan, Z. (2016). *Optimal pricing and ordering in global supply chain management with constraints under random demand.* Applied Mathematical Modeling, 40. 10105 – 10130.

Davoodi, S. & Jolai, F. & Mohaghar, A. & Mehregan, M. (2016). *Developing a New Algorithm for Finding the Local Minimums of the Multi-Echelon Inventory Control Systems with Random Parameters.* Procedia Economics and Finance, 36. 256 – 265.

Tsang, M. & Shehadeh, K. (2023). *Stochastic optimization models for a home service routing and appointment scheduling problem with random travel and service time*s. European Journal of Operations Research, 307. 48 – 63.

Fiat, A. & Karp, R.M. & McGeoch, L.A. & Sleator, D.D. & Young, N.E. (1991). *Competitive paging algorithms.* Jounal of algorithms, 12. 685-699.

McGeoch,L.A & Sleator, D.D. (1991). *A strongly competitive randomized paging algorithm.* Algorithmica, 6. 816 – 825.

Vashistha, J. & Jayswal, A. K. (2013). *Comparative Study of Load Balancing Algorithms*. IOSR Journal of Engineering, 3. 45-50.

Nadeem, M. & Wasid, M. & Nadeem, M. & Alam, M. et al. (2022). *Performance Comparison of Randomized and Non-Randomized Learning Algorithms based Recommender Systems.* International Journal of Next-Generation Computing, 13. 560-576.

Web Reference

Borodin, Allan & Pankratov, Denis (2019). *Online Algorithms*. University of Toronto. http://www.cs.toronto.edu/~bor/2420s19/our-online-text/chapters1-3.pdf

Kesselheim, T (2018). *Ski Rental Problem, Algorithms and Uncertainty.* Universität Bonn.

https://nerva.cs.uni-bonn.de/lib/exe/fetch.php/teaching/ws1819/vl-aau/lecturenotes01.pdf