Average-Case Analysis

Lecture Notes, Winter Term 08/09

University of Freiburg

Alexander Souza

Contents

Chapter 1

Introduction

Generally speaking, *average-case analysis* asks how some given algorithm behaves "typically". The main motivation for this is that for some (practically relevant) algorithms there is a gap between the worst-possible and the usually observed mannerism.

For example, the deterministic sorting algorithm QUICKSORT that always chooses the first element of an n-element array as its pivot requires $O\left(n^2\right)$ comparisons in the worst case. Thus, from that perspective one has to dismiss QUICKSORT (since there are algorithms that guarantee $O\left(n \log n\right)$ comparisons). On the other hand, QUICKSORT is often observed to outperform other algorithms in terms of number of comparisons. We will show very soon that the expected number of comparisons of QUICKSORT is also $O\left(n \log n\right)$ and even that this is achieved with high probability.

Now we explain the unspecified notions of the first two sentences in little more detail. First of all, an *algorithm* receives an input and produces an output after a finite number of elementary operations. Within this lecture we will mostly be concerned with *optimization algorithms*. The outputs these algorithms produce are valuated with an objective function and the goal is to minimize (respectively maximize) the valuation.

The second undetermined notion from above is *behave*: We will either be interested in the *running time* or the *approximation guarantee* of a particular algorithm. The number of elementary operations the algorithm executes before returning its output is called the running time on that particular instance. For example, many algorithms that solve NP-hard optimization problems exactly require exponential running time on some instances. Exponential running time is in general not desired since it only allows treatment of relatively small inputs. On the other hand, if we resort to polynomial running time we may have to surrender on solution quality (unless $\mathsf{P} = \mathsf{NP}$), i.e., the algorithm does not always produce solutions with best-possible objective value. We are mainly interested in algorithms where one can prove an a priori bound on the ratio of the objective value achieved by the algorithm compared with the best-possible value – the approximation guarantee.

The third notion, "*typically*" is intended to reflect the following gap that some practically relevant algorithms feature. From the *worst-case* perspective, unless $\mathsf{P} = \mathsf{NP}$, there is no way that we can avoid exponential running time (if we are interested in exact solutions) or suboptimal solutions (if we insist on polynomial running time) in general. So, what is the point in asking for "typical" running time, respectively approximation guarantee? The simple answer is: observed behaviour. There are several algorithms where the worst-case bounds on running time, respectively approximation guarantee, do not adequately reflect the behaviour one usually comes across – these bounds seem to be too pessimistic. Among these algorithms there are several ones that play an important role in applications, e.g., QUICKSORT for sorting and LEAST RECENTLY USED for memory management. It is thus

primarily of practical interest to give answers why those algorithms are often observed to work well. After this glimpse into the field we have to make all these notions more precise.

1.1 Optimization Problems and Algorithms

An *instance* I of a (combinatorial) optimization problem P can formally be defined as a tuple $I = (U, S, \text{val}, \text{extr})$ with the following meaning:

U	the *solution space* (on which val and S are defined),
S	the set of *feasible solutions* $S \subseteq U$,
val	the *objective function* $\text{val} : U \to \mathbb{R}$,
extr	the *extremum* (usually max or min).

U and S are usually not given explicitly. Instead, S is often defined with a *feasibility predicate* F by $S = \{X \in U : \ X \text{ satisfies } F\}$. Our goal is to find a feasible solution where the desired extremum of val is attained. It will be convenient to write a problem P in the format:

Problem 1.1 P

extremize	$\text{val}(X)$
subject to	$X \in U$ satisfies F

Any extremizing solution is called an *optimum solution*, or simply an *optimum*. We usually denote by $\text{OPT} = \text{OPT}(I)$ the optimal objective value of an instance I. The objective value an algorithm ALG achieves on the instance I is denoted $\text{ALG} = \text{ALG}(I)$.

Example 1.1 (KNAPSACK)**.** We have a knapsack that can carry a total weight of at most c. There are n items and each such item i has weight w_i and yields profit p_i at a market. The goal is to choose the items that maximize our total profit at a market without violating the capacity of our knapsack. Consider $c = 100$ and the following weight-profit table:

i	w_i	p_i
1	100	150
2	1	2
3	50	55
4	50	100

The feasibility predicate is F : "total weight of chosen items is at most c". For any set $X \subseteq \{1, 2, 3, 4\}$ define $p(X) = \sum_{i \in X} p_i$ and $w(X) = \sum_{i \in X} w_i$. Now we formulate the problem in the introduced notation.

Problem 1.2 KNAPSACK

$$\begin{aligned} \text{maximize} \quad & p(X) \\ \text{subject to} \quad & w(X) \leq c, \\ & X \subseteq \{1, 2, 3, 4\}. \end{aligned}$$

Which is the best solution? We evaluate all possibilities and find that $X^* = \{3, 4\}$ gives $w(X^*) = 155$ altogether which maximizes our profit.

A central problem around combinatorial optimization is that it is often in principle possible to find an optimum solution by enumerating the set of feasible solutions, but this set mostly contains "too many" elements. This phenomenon is called *combinatorial explosion*.

An *optimization algorithm* is given a description of an instance I, called the *input*, and has to produce a feasible solution O, called the *output*, after a finite number of elementary operations, where each such operation is effective and well-defined. We do not further specify what an elementary operation means as this depends on the context. Operations of interest include additions, multiplications, memory accesses, comparisons, and so on. The number of elementary operations an algorithm executes on an input I before producing an output O is called its *running time* on that input. The number of bits of the input is denoted $|I|$ and called the *input-length*. An algorithm whose running time is a polynomial *in* $|I|$ is called a *polynomial time* (or *efficient*) algorithm.

The central problem we are facing is that (unless $\mathsf{P} = \mathsf{NP}$), there are (many) optimization problems for which no efficient algorithm exists. This means that we only have the following choices: If we insist on the optimum solution, then we have to accept exponential running time (at least on some instances). Otherwise, if we want polynomial running time, then we must accept that any algorithm may produce suboptimal solutions (at least on some instances).

This fundamental problem yields that we have to reason more about how to judge optimization algorithms. It is convenient to see optimization as a game played between a (malicious) adversary and an algorithm. Such a game is played with an *optimization problem* P, which is simply a set of instances $P = \{I_1, I_2, \dots\}$. In the most basic form, the adversary chooses an instance among the members of P and the algorithm has to find a feasible solution. Throughout, we shall assume that our algorithms are deterministic.

1.2 Adversarial Models and Performance Measures

The variants of the game differ mainly in two ways:

(1) The way the adversary is restricted in the choice of the instance (the *adversarial model*), and

(2) the manner in which the solution produced by the algorithm is assessed (the *performance measure*).

We will present several popular choices in the paragraphs below.

Worst-Case Model

In the classical *worst-case* perspective, the adversary may choose an arbitrary instance I of the members of a given problem P.

For any instance I, let $t_I(\text{ALG})$ denote the running time of an algorithm ALG on it. In terms of running time we valuate the algorithm with the quantity

$$t(\text{ALG}) = \sup_{I \in P} t_I(\text{ALG}).$$

In terms of solution quality on a given instance I, we judge an algorithm with the *approximation ratio* $r_I(\text{ALG}) = \text{ALG}(I)/\text{OPT}(I)$ produced. Since the adversary is assumed to be malicious, we have to be pessimistic. If P is a minimization problem, the adversary

tries to maximize the approximation ratio and the overall quality of the algorithm is measured with

$$r(\text{ALG}) = \sup_{I \in P} r_I(\text{ALG}) = \sup_{I \in P} \frac{\text{ALG}(I)}{\text{OPT}(I)}.$$

An algorithm with $r(\text{ALG}) \leq \varrho$ is called a *ϱ-approximation*. If P is a maximization problem, he tries to minimize the ratio and we judge

$$r(\text{ALG}) = \inf_{I \in P} r_I(\text{ALG}) = \inf_{I \in P} \frac{\text{ALG}(I)}{\text{OPT}(I)}$$

and call it a ϱ-approximation if $r(\text{ALG}) \geq \varrho$.

Average-Case Models

For reasons sketched above, the worst-case model is sometimes too pessimistic to give satisfactory answers from the perspective of the practitioner. So, apparently, the adversary is too powerful, and one has to seek models that try to capture "typical" behaviour.

One attempt in that direction is the *diffuse adversary*. The adversary is given the problem, i.e., the set P of instances and along with that a set of available probability distributions $\Delta = \{D_1, D_2, \dots\}$. The set Δ is assumed to be chosen from an outside entity. Each distribution $D \in \Delta$ assigns to each instance $I \in P$ a certain probability $\Pr[I]$. Notice that the input-instance I is now a random variable. Instead of allowing the adversary to choose a particular instance we only allow him to choose any probability distribution from the set Δ.

In terms of running time, one is usually interested in

$$t(\text{ALG}) = \sup_{D \in \Delta} \mathbb{E}[t_I(\text{ALG})] = \sup_{D \in \Delta} \sum_{I \in P} t_I(\text{ALG}) \Pr[I].$$

Notice that I is a random variable and hence $t_I(\text{ALG})$ is also a random variable.

It is debatable how the quality of an algorithm should be judged in such a model. There are at least two alternatives. Let P be a minimization problem without loss of generality. To transfer the definitions below to maximization we only have to replace sup with inf.

One point of view is: Let D be the any distribution for P. Then taking the expectation with respect to D yields

$$e_D(\text{ALG}) = \mathbb{E}\left[\frac{\text{ALG}(I)}{\text{OPT}(I)}\right] = \sum_{I \in P} \frac{\text{ALG}(I)}{\text{OPT}(I)} \Pr[I]$$

which judges the algorithm through the expected value of the approximation ratio. Consequently, as the adversary is malicious, the overall judgement is given through

$$e(\text{ALG}) = \sup_{D \in \Delta} e_D(\text{ALG}) = \sup_{D \in \Delta} \mathbb{E}\left[\frac{\text{ALG}(I)}{\text{OPT}(I)}\right].$$

Another way is this: For a distribution D, and taking expectations with respect to it, define

$$e'_D(\text{ALG}) = \frac{\mathbb{E}[\text{ALG}(I)]}{\mathbb{E}[\text{OPT}(I)]} = \frac{\sum_{I \in P} \text{ALG}(I) \Pr[I]}{\sum_{I \in P} \text{OPT}(I) \Pr[I]},$$

i.e., the ratio of the expectations, which yields the overall measure

$$e'(\text{ALG}) = \sup_{D \in \Delta} e_D(\text{ALG}) = \sup_{D \in \Delta} \frac{\mathbb{E}[\text{ALG}(I)]}{\mathbb{E}[\text{OPT}(I)]}.$$

If $e(\text{ALG}) \leq \varrho$, respectively $e'(\text{ALG}) \leq \varrho$ holds, we speak of a ϱ-expected-approximation (with the measure used in the context).

Both measures have their advantages and drawbacks. The perspective of $e(\text{ALG})$ is that *one* instance is drawn at random and the algorithm is compared directly against the optimal algorithm *on that instance* (in expectation). Thus the measure prefers algorithms that give good approximation ratios "often". By contrast, from the point of view of $e'(\text{ALG})$ *many* instances are drawn and the *total objective value* achieved is compared with the total optimal value (respectively in expectation). This measure thus prefers algorithms that give good approximation ratios when the optimum itself yields "large" objective values. It certainly depends on the intended application which perspective is the advisable one.

Example 1.2. Consider a hypothetical minimization problem with two instances $P = \{A, B\}$. Associate the following probability distribution with the problem

$$\Pr[x \in A] = 1 - \frac{1}{n} \quad \text{and} \quad \Pr[x \in B] = \frac{1}{n}.$$

where n is arbitrarily large. Let OPT denote the value of the optimal objective value and define $\text{OPT}(A) = 1$ and $\text{OPT}(B) = n$. Let ALG_1 and ALG_2 denote the values achieved by two certain algorithms where these have the properties $\text{ALG}_1(A) = 1$, $\text{ALG}_1(B) = n^2$, $\text{ALG}_2(A) = n$, and $\text{ALG}_2(B) = n$.

The following truth-table depicts the relative qualities of the algorithms measured with $r(\text{ALG})$, $e(\text{ALG})$, and $e'(\text{ALG})$ respectively.

I	A	B	$r(\text{ALG})$	$e(\text{ALG})$	$e'(\text{ALG})$
$\text{OPT}(I)$	1	n	1	1	1
$\text{ALG}_1(I)$	1	n^2	n	2	$(n+1)/2$
$\text{ALG}_2(I)$	n	n	n	n	$n/2$
$\Pr[I]$	$1-1/n$	$1/n$			

Observe that ALG_1 achieves the optimum value with probability $1 - 1/n$ and with probability $1/n$ it has worst-case performance $\text{ALG}_1(I)/\text{OPT}(I) = n$. As n grows, the algorithm performs optimal with increasing probability.

Further observe that ALG_2 achieves the worst-case performance with probability $1-1/n$ and the optimum value on instances where the optimum value is n, i.e., large. But these occur only with probability $1/n$.

The measure $e(\text{ALG})$ favours ALG_1, i.e., the one that performs well with high probability. The larger n is, the more it favours ALG_1, as its probability for being optimal increases. By contrast $e'(\text{ALG})$ favours ALG_2, which works well on instances where the optimum value is large. Finally, with the approximation ratio $r(\text{ALG})$, these algorithms are even indistinguishable.

Decision-Making Under Uncertainty

In the above paragraphs we always assumed implicitly that the algorithm is given the whole input at the outset. This assumption is not always justified. Sometimes we are facing situations where the instance becomes known only gradually. Nonetheless, an algorithm has to make decisions.

For example, due to the halting problem, it is impossible to find out how long a processor will be executing a program before it terminates (or whether it terminates at all). However, the operating system has to decide which program to run and how much computation time it will be granted.

A classical worst-case notion is *competitive analysis* where an algorithm receives a sequence of inputs and has to react upon each one without any foresight. Analogously to the approximation ratio, the performance is judged by comparing the objective value achieved with the optimal value. In such situations, a malicious adversary is even more powerful than before, since an algorithm that is not even aware of the whole instance is compared with an optimal algorithm that knows the whole instance in advance. Thus the problem that the results in this model tend to be overly pessimistic is true even more.

Hence average-case models for decision-making under uncertainty are interesting both from a theoretical and practical perspective. However, up to now, there are several reasonable approaches, each one with advantages and drawbacks. We will not go into the details here, but introduce the models in the chapters when discussed.

1.3 Basic Probability Theory

This section is intended as a refresher of the basic notions of probability theory. We will restrict ourselves to discrete probability spaces here. This will mostly be sufficient for our purposes, but in case needed, we will also allow continuous probability spaces with analogous definitions.

Fundamentals

A set $\Omega = \{\omega_1, \omega_2, \dots\}$ is called a *probability space* and its elements $\omega \in \Omega$ are called the *elementary events*. A *probability distribution* D assigns to each elementary event ω a number, called its *probability* $\Pr[\omega]$ such that:

(1) $0 \leq \Pr[\omega] \leq 1$ for all $\omega \in \Omega$, and

(2) $\sum_{\omega \in \Omega} \Pr[\omega] = 1$.

Any subset $A \subseteq \Omega$ is called an *event* and the corresponding probability is given through $\Pr[A] = \sum_{\omega \in A} \Pr[\omega]$. For each event A define the *complementary event* $\overline{A} = \Omega - A$. Observe that $A \subseteq B$ implies $\Pr[A] \leq \Pr[B]$. Notice the special probabilities $\Pr[\emptyset] = 0$ and $\Pr[\Omega] = 1$.

Theorem 1.3. *Let $A_1, \dots, A_n$ be pairwise disjoint events, i.e., $A_i \cap A_j = \emptyset$ for all $i \neq j$, then $\Pr[\cup_{i=1}^n A_i] = \sum_{i=1}^n \Pr[A_i]$. If the events are not pairwise disjoint, then we have $\Pr[\cup_{i=1}^n A_i] \leq \sum_{i=1}^n \Pr[A_i]$.*

Let A and B be two events where $B \neq \emptyset$. Then the *conditional probability of A given B* is defined by $\Pr[A \mid B] = \Pr[A \cap B] / \Pr[B]$. Two events A and B are called *independent* if $\Pr[A \cap B] = \Pr[A] \Pr[B]$.

Theorem 1.4. *If $\Pr[A_1 \cap \dots \cap A_n] > 0$ then*

$$\Pr[A_1 \cap \dots \cap A_n] = \Pr[A_1] \Pr[A_2 \mid A_1] \cdot \dots \cdot \Pr[A_n \mid A_1 \cap \dots \cap A_{n-1}].$$

Theorem 1.5. *Let $A_1, \dots, A_n$ be pairwise disjoint events and let $B \subseteq A_1 \cup \dots \cup A_n$. Then we have*

$$\Pr[B] = \sum_{i=1}^{n} \Pr[B \mid A_i] \Pr[A_i].$$

If $\Pr[B] > 0$ we additionally have

$$\Pr[A_i \mid B] = \frac{\Pr[A_i \cap B]}{\Pr[B]} = \frac{\Pr[A_i \cap B]}{\sum_{i=1}^n \Pr[B \mid A_i] \Pr[A_i]}.$$

For some given probability space any mapping $X : \Omega \to \mathbb{Z}$ is called a (numerical) *random variable*. The *expected value* of X is given through

$$\mathbb{E}[X] = \sum_{\omega \in \Omega} X(\omega)\Pr[\omega] = \sum_{x \in \mathbb{Z}} x\Pr[X = x]$$

provided that $\sum_{x \in \mathbb{Z}} |x|\Pr[X = x]$ converges.

Theorem 1.6. *Let X and Y be two random variables such that $X(\omega) \leq Y(\omega)$ for all $\omega \in \Omega$ then $\mathbb{E}[X] \leq \mathbb{E}[Y]$.*

Theorem 1.7 (Linearity of Expectation)**.** *For random variables $X_1, \ldots, X_n$ and constants $a_1, \ldots, a_n, b$ we have that*

$$\mathbb{E}[a_1X_1 + \cdots + a_nX_n + b] = a_1\mathbb{E}[X_1] + \cdots + a_n\mathbb{E}[X_n] + b.$$

The *variance* of a random variable X is $\mathrm{Var}[X] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2$. Random variables X and Y are called *independent* if $\Pr[X = x, Y = y] = \Pr[X = x]\Pr[Y = y]$ holds for all x and y.

Theorem 1.8. *For* independent *random variables $X_1, \ldots, X_n$ and constants $a_1, \ldots, a_n, b$ we have that*

$$\mathbb{E}[X_1 \cdot \cdots \cdot X_n] = \mathbb{E}[X_1] \cdot \cdots \cdot \mathbb{E}[X_n]$$

and

$$\mathrm{Var}[a_1X_1 + \cdots + a_nX_n + b] = a_1^2\mathrm{Var}[X_1] + \cdots + a_n^2\mathrm{Var}[X_n].$$

Bounds on Distributions

Two elementary bounds on the distribution of any random variable are the theorems of Markov and Chebyshev.

Theorem 1.9 (Markov)**.** *Let $X \geq 0$ be a random variable. Then*

$$\Pr[X \geq t] \leq \frac{\mathbb{E}[X]}{t}$$

holds for all $t > 0$.

Theorem 1.10 (Chebyshev)**.** *Let X be a random variable. Then*

$$\Pr[|X - \mathbb{E}[X]| \geq t] \leq \frac{\mathrm{Var}[X]}{t^2}$$

holds for all $t > 0$.

Important Distributions

A *Bernoulli* distributed random variable $X \in \{0, 1\}$ indicates if a certain event that has probability p occurs. We denote $X \sim \mathrm{Ber}(p)$ and it has the distribution:

$$\Pr[X = 1] = p \quad \text{and} \quad \Pr[X = 0] = 1 - p.$$

We further have $\mathbb{E}[X] = p$ and $\mathrm{Var}[X] = p(1 - p)$.

A sum of n independent Bernoulli variables $X = X_1 + \cdots + X_n$ with respective parameter p has *Binomial* distribution, denoted $X \sim \mathrm{Bin}\,(n, p)$. The distribution is for any $k \in \{0, \ldots, n\}$:

$$\Pr[X = k] = \binom{n}{k} p^k (1-p)^{n-k}.$$

We clearly have $\mathbb{E}[X] = np$ and $\mathrm{Var}[X] = np(1-p)$.

A random variable $X \in \{1, 2, \ldots\}$ that counts the number of trials until some event that has probability p occurs for the first time has *geometric distribution*. It is denoted $X \sim \mathrm{Geo}\,(p)$ and has the distribution

$$\Pr[X = k] = (1-p)^{k-1} p$$

and $\mathbb{E}[X] = 1/p$ and $\mathrm{Var}[X] = (1-p)/p^2$.

A *uniform distributed* random variable X is used to assign the same probability to each element of a set $U = \{a, a+1, \ldots, b-1, b\}$. We write $X \sim \mathrm{Uni}\,\{a, \ldots, b\}$ and it has the distribution

$$\Pr[X = k] = \frac{1}{b-a+1}$$

and $\mathbb{E}[X] = (a+b)/2$ and $\mathrm{Var}[X] = ((b-a+1)^2 - 1)/12$.

Chapter 2

Sorting and Selecting

In this chapter we give average-case analyses of the well-known sorting and searching algorithms QUICKSORT and QUICKSELECT.

2.1 Quicksort

The problem SORT is the following: We are given a sequence $a = (a_1, a_2, \ldots, a_n)$ of pairwise distinct numbers and are asked to find a permutation π of $(1, 2, \ldots, n)$ such that the sequence $b = (b_1, b_2, \ldots, b_n) = (a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)})$ satisfies $b_1 \leq b_2 \leq \cdots \leq b_n$. The elementary operations we are interested in are the comparisons "$\leq$" an algorithm asks. Let $t_a(\text{ALG})$ be the number of comparisons of an algorithm ALG given a sequence a.

The assumption that the numbers are pairwise distinct can easily be removed, but we consider it for clarity of exposition.

The idea of the algorithm QUICKSORT is to choose some element p from a, called the *pivot*, and divide a into two subsequences ℓ and r. The sequence ℓ contains the elements $a_i \leq p$ and r those $a_i > p$. QUICKSORT is then called recursively until the sequence is empty. The sequence $(\text{QUICKSORT}(\ell), p, \text{QUICKSORT}(r))$ is finally returned.

Algorithm 2.1 QUICKSORT

Input. Sequence $(a_1, a_2, \ldots, a_n)$

Output. Sequence $(b_1, b_2, \ldots, b_n)$

(1) If $n = 0$ return.

(2) Otherwise let $p = a_1$. Let ℓ and r be two empty sequences.

(3) For $i = 2, \ldots, n$, if $a_i \leq p$ append a_i to ℓ otherwise append a_i to r.

(4) Return $(\text{QUICKSORT}(\ell), p, \text{QUICKSORT}(r))$

Worst-Case Analysis

It is well-known that this variant of QUICKSORT has the weakness that it may require $\Omega\left(n^2\right)$ comparisons.

Observation 2.1. *There is a sequence a such that $t_a(\text{QUICKSORT}) = n(n-1)/2$.*

Proof. Consider $a = (1, 2, \ldots, n)$. Then, in step (3), ℓ remains empty while r contains $n-1$ elements. This step requires $n-1$ comparisons. By induction, the recursive calls QUICKSORT(ℓ) and QUICKSORT(r) require 0, respectively $(n-1)(n-2)/2$ comparisons "$\leq$". Thus, the whole algorithm needs $n-1+(n-1)(n-2)/2 = n(n-1)/2$ comparisons. □

Average-Case Analysis

Here we study QUICKSORT in the following probabilistic model. Let π be any permutation of $(1, 2, \ldots, n)$ and let the probability of π be $1/n!$, i.e., $\Pr[\pi] = 1/n!$. We draw a permutation π at random according to this probability distribution and the input of QUICKSORT is the sequence $A = (a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)})$. (We use the capital letter A to emphasize that it is a random variable.)

Then the expected number of comparisons QUICKSORT needs is $O(n \log n)$. More specifically let $H_n = \sum_{k=1}^{n} 1/k$ denote the n-th *Harmonic number* and recall that $\log n \leq H_n \leq \log n + 1$.

Theorem 2.2. *We have that* $\mathbb{E}[t_A(\text{QUICKSORT})] = 2(n+1)H_n - 4n$.

We will give two proofs of this theorem. In the first one we derive a recursion, in the second one we use linearity of expectation. It causes no loss of generality to assume that the original sequence is itself a permutation of $(1, 2, \ldots, n)$ and will be assumed in the sequel. Also, let $c_n = \mathbb{E}[t_A(\text{QUICKSORT})]$ be the expected number of comparisons QUICKSORT carries out on a uniformly distributed permutation A of length n.

First proof of Theorem 2.2. The idea of this proof is to give a recursion formula and then solve it (by induction). Before we actually prove this theorem we need a lemma that is needed due to the recursive structure of the algorithm. The original input A is a random permutation with uniform distribution. Hence the pivot P is a random variable. Thus also the sequences L and R constructed by QUICKSORT are random permutations of subsets of $\{1, 2, \ldots, n\}$. However, for the proof to work, we have to show that these sequences are also uniform distributed.

Lemma 2.3. *Let the outcome of the pivot element be $P = p$. Let $L = \ell$ and $R = r$ be two possible outcomes of the sequences* QUICKSORT *constructs given pivot p. Then we have*

$$\Pr[L = \ell \mid P = p] = \frac{1}{|\ell|!} \quad \textit{and} \quad \Pr[R = r \mid P = p] = \frac{1}{|r|!}.$$

Proof. We give a proof for L; for R it is analogous. Conditioning on a pivot element $p \in \{1, 2, \ldots, n\}$ yields that *each* outcome ℓ of L has the same elements, namely, the numbers $i < p$. Let ℓ be any outcome of L and let $k = |\ell|$ be the cardinality of that sequence. There are $\binom{n-1}{k}(n-1-k)!$ many permutations of the sequence $(1, 2, \ldots, n)$ that contain p as first element and ℓ as a subsequence. Thus, the probability that the outcome of L is exactly ℓ (given pivot p) is

$$\Pr[L = \ell \mid P = p] = \binom{n-1}{k} \frac{(n-1-k)!}{(n-1)!} = \frac{(n-1)!}{k!(n-1-k)!} \cdot \frac{(n-1-k)!}{(n-1)!} = \frac{1}{k!}$$

which is what we had to show. □

Since A is uniformly distributed the probability that the outcome of the pivot $P = p$ equals $1/n$ for every $p \in \{1, 2, \ldots, n\}$. For $n = 0$ we have $c_0 = 0$ and for $n > 0$ we we obtain the recursion

$$\begin{aligned}
c_n = \mathbb{E}\left[t_A(\textsc{Quicksort})\right] &= n - 1 + \sum_{p=1}^{n} \mathbb{E}\left[t_A(\textsc{Quicksort}) \mid P = p\right] \Pr\left[P = p\right] \\
&= n - 1 + \sum_{p=1}^{n} (\mathbb{E}\left[t_L(\textsc{Quicksort}) \mid P = p\right] + \mathbb{E}\left[t_R(\textsc{Quicksort}) \mid P = p\right]) \Pr\left[P = p\right] \\
&= n - 1 + \sum_{p=1}^{n} (c_{p-1} + c_{n-p}) \frac{1}{n} \\
&= n - 1 + \frac{2}{n} \sum_{k=0}^{n-1} c_k
\end{aligned}$$

where the equality $\mathbb{E}\left[t_L(\textsc{Quicksort}) \mid P = p\right] = c_{p-1}$ for L (and analogously for R) depends critically on Lemma 2.3.

We "guess" a solution for c_n and verify by induction that it satisfies the recursion $c_n = n - 1 + 2/n \sum_{k=0}^{n-1} c_k$. Our guess is $c_k = 2(k+1)H_k - 4k$ for any $k \in \{0, 1, \ldots, n\}$.

The base case is $k = 0$. There, Quicksort does not perform any comparison at all, i.e., $c_0 = 0$. Using $H_0 = 0$ yields the base case.

For the inductive case we have that the claim is true for any $k \leq n - 1$ and we will prove it for $k = n$. Using the identity $\sum_{k=0}^{n-1}(k+1)H_k = n/2(nH_n - n/2 - 3/2 + H_n)$ we find

$$\begin{aligned}
c_n = n - 1 + \frac{2}{n} \sum_{k=0}^{n-1} c_k &= n - 1 + \frac{2}{n} \sum_{k=0}^{n-1} (2(k+1)H_k - 4k) \\
&= n - 1 + 2\left(nH_n - \frac{n}{2} - \frac{3}{2} + H_n\right) - 4(n-1) = 2(n+1)H_n - n - 3 - 3(n-1) \\
&= 2(n+1)H_n - 4n
\end{aligned}$$

and the result is established. □

Second proof of Theorem 2.2. We assumed that the original sequence is a permutation of $(1, 2, \ldots, n)$. So, for any $i < j \in \{1, 2, \ldots, n\}$ let the random variable X_{ij} be equal to one if i and j are compared during the course of the algorithm and zero otherwise. The total number of comparisons is hence $X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$. Thus, by

$$c_n = \mathbb{E}\left[X\right] = \mathbb{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{E}\left[X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\left[X_{ij} = 1\right]$$

we have to derive the probability that i and j are compared.

First observe that *each* element will be pivot element in the course of the algorithm *exactly* once. Thus the input A induces a unique sequence $P = (P_1, P_2, \ldots, P_n)$ of pivots.

Now fix i and j arbitrarily. When will these elements be compared? We claim that it will be the case if and only if either i or j is the *first* pivot in the sequence P from the set $\{i, \ldots, j\}$. If i and j are compared, then either one must be the pivot and they must be in the same sequence. Thus all previous pivots (if any) must be smaller than i or larger than j, since i and j would end up in different subsequences, otherwise. Hence, either i or

j is the first pivot in the set $\{i, \ldots, j\}$. The converse direction is trivial: if one of i or j is the first pivot from the set $\{i, \ldots, j\}$, then i and j are still in the same sequence and will hence be compared.

What is the probability that, say, i is the first pivot of $\{i, \ldots, j\}$? Consider the subsequence S induced by the elements $\{i, \ldots, j\}$ on A. The crucial observation is that S is also a subsequence of P. Hence i is the first pivot from the set $\{i, \ldots, j\}$ if and only if i is the first element from that set in A. Since A is uniformly distributed the probability for that event is exactly $1/(j-i+1)$. Analogous reasoning for j yields the overall probability $\Pr[X_{ij} = 1] = 2/(j-i+1)$.

This allows us to calculate

$$\begin{aligned} c_n &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr[X_{ij} = 1] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k} \\ &= \sum_{k=2}^{n} \sum_{i=1}^{n+1-k} \frac{2}{k} = \sum_{k=2}^{n} (n+1-k)\frac{2}{k} = (n+1)\sum_{k=2}^{n} \frac{2}{k} - 2(n-1) \\ &= 2(n+1)H_n - 4n \end{aligned}$$

and the proof is complete. □

So we have shown that the expected number of comparisons of QUICKSORT is $O(n \log n)$. We can even strengthen the statement: The number of comparisons is $O(n \log n)$ *with high probability*, i.e., with probability that tends to one as n tends to infinity.

Theorem 2.4. *For any $c > 1$ we have that*

$$\Pr[t_A(\text{QUICKSORT}) \leq c\mathbb{E}[t_A(\text{QUICKSORT})]] \geq 1 - \frac{4}{(c-1)^2 H_n}.$$

Proof. Our goal is to apply Chebyshev's inequality: $\Pr[|X - \mathbb{E}[X]| \geq t] \leq \operatorname{Var}[X]/t^2$, where, as before X denotes the number of comparisons.

Suppose that we can prove $\operatorname{Var}[X] = 2n\mathbb{E}[X] + 8n^2$ then we have

$$\begin{aligned} \Pr[X > c\mathbb{E}[X]] &\leq \Pr[|X - \mathbb{E}[X]| > (c-1)\mathbb{E}[X]] \leq \frac{\operatorname{Var}[X]}{(c-1)^2\mathbb{E}[X]^2} \\ &= \frac{2}{2(c-1)^2 H_n} + \frac{2}{(c-1)^2 H_n^2} \leq \frac{4}{(c-1)^2 H_n} \end{aligned}$$

which is what we wanted to show.

As before, X_{ij} is equal to one if the elements i and j are compared during the course of the algorithm, zero otherwise.

$$\begin{aligned} \operatorname{Var}[X] &= \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2 = \mathbb{E}\left[\left(\sum_{i<j} X_{ij}\right)\left(\sum_{k<\ell} X_{k\ell}\right)\right] - \mathbb{E}[X]^2 \\ &= \sum_{i<j} \mathbb{E}[X_{ij}] + 2\sum_{i<j,k} \mathbb{E}[X_{ij}X_{ik}] + \sum_{\substack{i<j,\ k<\ell \\ k\neq i,\ \ell\neq j}} \mathbb{E}[X_{ij}X_{k\ell}] - \mathbb{E}[X]^2 \\ &\leq \mathbb{E}[X] + 2\sum_{i<j,k} \mathbb{E}[X_{ij}X_{ik}] + \sum_{\substack{i<j,\ k<\ell \\ k\neq i,\ \ell\neq j}} \mathbb{E}[X_{ij}]\,\mathbb{E}[X_{k\ell}] - \mathbb{E}[X]^2 \\ &\leq \mathbb{E}[X] + 2\sum_{i<j,k} \mathbb{E}[X_{ij}X_{ik}]. \end{aligned}$$

Now we have to bound $\mathbb{E}[X_{ij}X_{ik}]$ for any combination of $i < j$ and a free k. For any of the cases $i < j < k$, $i < k < j$, and $k < i < j$ we have

$$\mathbb{E}[X_{ij}X_{ik}] = \mathbb{E}[X_{ik} \mid X_{ij}] \Pr[X_{ij} = 1] \leq \frac{2}{j - i + 1}.$$

This yields

$$\sum_{i<j,k} \mathbb{E}[X_{ij}X_{ik}] \leq (n-1) \sum_{i<j} \frac{2}{j-i+1} \leq 2n(n-1)H_n$$

and thus in total $\mathrm{Var}[X] \leq \mathbb{E}[X] + 4(n-1)nH_n \leq 2n\mathbb{E}[X] + 8n^2$ which gives the result. □

2.2 Quickselect

In the problem SELECT we are given a sequence $a = (a_1, a_2, \ldots, a_n)$ of pairwise distinct numbers and are asked to find the k-th smallest element, for some given k. We are again interested in the number of comparisons "$\leq$". Let $t_a(\text{ALG}, k)$ be the number of comparisons of an algorithm ALG given a sequence a and parameter k.

The idea of the algorithm QUICKSELECT is, similarly to QUICKSORT, to choose some element p from a, called the *pivot*, and divide a into two subsequences ℓ and r. The sequence ℓ contains the elements $a_i \leq p$ and r those $a_i > p$. Then QUICKSELECT determines which subsequence contains the desired element and is then called recursively on that subsequence until the element is found.

Algorithm 2.2 QUICKSELECT

Input. Sequence $(a_1, a_2, \ldots, a_n)$, integer k

Output. Element b

(1) If $n = 0$ return.

(2) Otherwise let $p = a_1$. Let ℓ and r be two empty sequences.

(3) For $i = 2, \ldots, n$, if $a_i \leq p$ append a_i to ℓ otherwise append a_i to r.

(4) If $|\ell| = k - 1$ return $b = p$. Else, if $|\ell| > k$ return QUICKSELECT(ℓ, k), otherwise return QUICKSELECT$(r, k - |\ell|)$

Worst-Case Analysis

Analogously to QUICKSORT, this variant of QUICKSELECT may require $\Omega(n^2)$ comparisons.

Observation 2.5. *There is a sequence a such that $t_a(\text{QUICKSELECT}, n) = n(n-1)/2$.*

Proof. Consider $a = (1, 2, \ldots, n)$ and $k = n$. Then, in step (3), ℓ remains empty while r contains $n - 1$ elements. This step requires $n - 1$ comparisons. By induction, the recursive call to QUICKSELECT$(r, n-1)$ requires $(n-1)(n-2)/2$ comparisons "$\leq$". Thus, the whole algorithm needs $n - 1 + (n-1)(n-2)/2 = n(n-1)/2$ comparisons. □

Average-Case Analysis

We study QUICKSELECT also in the probabilistic model that each permutation of the original sequence is equally likely. Then the expected number of comparisons of QUICKSELECT is $O(n)$.

Theorem 2.6. *For any k we have that $\mathbb{E}[t_A(\text{QUICKSELECT}, k)] \leq 4n$.*

Proof. Again, without loss of generality we assume that the original sequence is itself a permutation of $(1, 2, \ldots, n)$. Let $c_{n,k}$ be the expected number of comparisons of QUICKSELECT for the parameter k on a uniformly distributed permutation A of length n.

We give a recursion formula and then give an upper bound on the solution (by induction). Notice that Lemma 2.3 holds here, too, since QUICKSELECT uses the same pivot-rule as QUICKSORT. Since A is uniformly distributed the probability that the outcome of the pivot $P = p$ equals $1/n$ for every $p \in \{1, 2, \ldots, n\}$. For $n = 0$ we have $c_0 = 0$ and for $n > 0$ we we obtain the recursion

$$\begin{aligned}
c_{n,k} = \mathbb{E}[t_A(\text{QUICKSELECT}, k)] &= n - 1 + \sum_{p=1}^{n} \mathbb{E}[t_A(\text{QUICKSELECT}, k) \mid P = p] \Pr[P = p] \\
&= n - 1 + \left(\sum_{p=1}^{k-1} \mathbb{E}[t_R(\text{QUICKSELECT}, k-p) \mid P = p] \right. \\
&\qquad \left. + \sum_{p=k+1}^{n} \mathbb{E}[t_L(\text{QUICKSELECT}, k) \mid P = p] \right) \Pr[P = p] \\
&= n - 1 + \frac{1}{n} \left(\sum_{p=1}^{k-1} c_{n-p,k-p} + \sum_{p=k+1}^{n} c_{p,k} \right)
\end{aligned}$$

where the equality $\mathbb{E}[t_L(\text{QUICKSELECT}) \mid P = p] = c_{p,k}$ for L (and analogously for R) depends critically on Lemma 2.3.

We guess that $c_{n,k} \leq 4n$ for any k and verify

$$\begin{aligned}
c_{n,k} = n - 1 + \frac{1}{n} \left(\sum_{p=1}^{k-1} c_{n-p,k-p} + \sum_{p=k+1}^{n} c_{p,k} \right) &\leq n - 1 + \frac{1}{n} \left(\sum_{p=1}^{k-1} 4(n-p) + \sum_{p=k+1}^{n} 4p \right) \\
&= n - 1 + \frac{4}{n} \left(n(n-1) - \frac{(n-k)(n-k+1)}{2} - \frac{k(k+1)}{2} \right) \\
&\leq n - 1 + \frac{4}{n} \left(n(n-1) - \frac{n^2}{4} \right) = n - 1 + 4(n-1) - n \\
&\leq 4n
\end{aligned}$$

and the result is established. □

Chapter 3

Knapsack

This chapter is concerned with the KNAPSACK problem. This problem is of interest in its own right because it formalizes the natural problem of selecting items so that a given budget is not exceeded but profit is as large as possible. Questions like that often also arise as subproblems of other problems. Typical applications include: option-selection in finance, cutting, and packing problems. See Example 1.1 for an illustration.

In the KNAPSACK problem we are given a *capacity* c and n items. Each item i comes along with a *profit* p_i and a *weight* w_i. We are asked to choose a subset of the items as to maximize total profit but the total weight not exceeding c. Let $x_j \in \{0,1\}$ indicate if a certain item j is included in the solution or not.

In the sequel let $p = (p_1, \ldots, p_n), w = (w_1, \ldots, w_n)$, and $x = (x_1, \ldots, x_n)$ denote the respective vectors. The profit of a vector $x \in \{0,1\}^n$ is $\text{val}(x) = \sum_{j=1}^{n} p_j x_j$. The weight of a vector $x \in \{0,1\}^n$ is given by $\text{weight}(x) = \sum_{j=1}^{n} w_j x_j$. In order to obtain a non-trivial problem we assume $w_j \leq c$ for all $j = 1, \ldots, n$ and $\sum_{j=1}^{n} w_j > c$ throughout.

Problem 3.1 KNAPSACK

$$\begin{aligned} \text{maximize} \quad & \text{val}(x) = \sum_j p_j x_j \\ \text{subject to} \quad & \sum_j w_j x_j \leq c, \\ & x_j \in \{0,1\} \quad j = 1, \ldots, n. \end{aligned}$$

KNAPSACK is NP-hard which means that "most probably", there is no polynomial time optimization algorithm for it. We will consider two stochastic variants of the problem: In Section 3.1 the profits of the items are random, in Section 3.2 the weights are.

3.1 Stochastic Profits

3.1.1 Greedy Algorithm

The idea behind the following GREEDY algorithm is to successively choose items that yield most profit per unit weight. The ratio p_j/w_j is called the *efficiency* of item j. We may assume that the items are given in non-increasing order of efficiency. The smallest item number k such that $\sum_{j=1}^{k} p_j > c$ is called the *break item*, i.e., the first one that does not

fit into the knapsack anymore. Common sense suggests the following simple algorithm: $x_j = 1$ for $j = 1, \ldots, k-1$, $x_j = 0$ for $j = k, \ldots, n$.

Unfortunately, the approximation ratio of this algorithm can be arbitrarily bad as the example below shows. The problem is that more efficient items can "block" more profitable ones.

Example 3.1. Consider the following instance, where c is a sufficiently large integer.

j	p_j	w_j	p_j/w_j
1	1	1	1
2	$c-1$	c	$1-1/c$

The algorithm chooses item 1, i.e., the solution $x = (1, 0)$ and hence $\text{val}(x) = 1$. The optimum solution is $x^* = (0, 1)$ and thus $\text{val}(x^*) = c - 1$. The approximation ratio of the algorithm is $1/(c-1)$, i.e., arbitrarily bad (small). However, this natural algorithm can be turned into a 1/2-approximation from a worst-case perspective and even better from an average-case point of view.

Algorithm 3.1 GREEDY

Input. Integer c, vectors $p, w \in \mathbb{N}^n$ with $w_j \leq c$, $\sum_j w_j > c$, and $p_1/w_1 \geq \cdots \geq p_n/w_n$.

Output. Vector $x \in \{0,1\}^n$ such that $\text{weight}(x) \leq c$.

(1) Define $k = \min\{j \in \{1, \ldots, n\} : \sum_{i=1}^{j} w_i > c\}$.

(2) Let $x = (1^{k-1}, 0^{n-k+1})$ and $y = (0^{k-1}, 1, 0^{n-k})$.

(3) Return x if $\text{val}(x) \geq \text{val}(y)$; otherwise y.

Worst-Case Analysis

The result below states that this modified algorithm is a 1/2-approximation. It is an exercise to show that this bound is tight.

Theorem 3.2. *We have that* $r(\text{GREEDY}) \geq 1/2$.

Proof. In Problem 3.1, let us replace the constraints $x_j \in \{0,1\}$ with $x_j \in [0,1]$. This yields that the optimum solution of the modified problem can only be larger than the optimum of the original problem. Such a modification is called *relaxation* of the constraints.

Problem 3.2 FRACTIONAL KNAPSACK

$$\begin{aligned} \text{maximize} \quad & \text{val}(x) = \sum_j p_j x_j \\ \text{subject to} \quad & \sum_j w_j x_j \leq c, \\ & x_j \in [0,1] \quad j = 1, \ldots, n. \end{aligned}$$

The optimal solution of this problem has the following structure. The proof of the observation below is an exercise.

Observation 3.3. *Let $p, w, \in \mathbb{N}^n$ be non-negative integral vectors with*

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_n}{w_n}.$$

The break index is defined as

$$k = \min \left\{ j \in \{1, \ldots, n\} : \sum_{i=1}^{j} w_i > c \right\}.$$

Then an optimum solution for the Fractional Knapsack *problem is given by*

$$\begin{aligned} z_j^* &= 1 \quad \textit{for } j = 1, \ldots, k-1, \\ z_j^* &= \frac{c - \sum_{i=1}^{k-1} w_i}{w_k} \quad \textit{for } j = k, \textit{ and} \\ z_j^* &= 0 \quad \textit{for } j = k+1, \ldots, n. \end{aligned}$$

Notice that z^* has the form $z^* = (1, \ldots, 1, \alpha, 0, \ldots, 0)$ for some value $0 \leq \alpha < 1$ at the break index. The Greedy algorithm constructs two solutions x and y. Observe that x has the form $x = (1, \ldots, 1, 0, 0, \ldots, 0)$, i.e., equal to z^* except it has $\alpha = 0$ at the break index. Furthermore $y = (0, \ldots, 0, 1, 0, \ldots, 0)$ with only the break item taken. The value obtained by the Greedy algorithm is equal to $\max\{\text{val}(x), \text{val}(y)\}$.

Let x^* be an optimum solution for the Knapsack instance. Since every solution that is feasible for the Knapsack instance is also feasible for the respective Fractional Knapsack instance we have that $\text{val}(x^*) \leq \text{val}(z^*)$.

In total we have

$$\text{val}(x^*) \leq \text{val}(z^*) = \text{val}(x) + \alpha p_k \leq \text{val}(x) + \text{val}(y) \leq 2 \max\{\text{val}(x), \text{val}(y)\}$$

which implies the approximation ratio of $1/2$. □

Average-Case Analysis

On random instances (and sufficiently large capacity c), Greedy is usually much better than 1/2-approximate. Indeed, the algorithm approximates the better, the larger the capacity is. In the sequel let the weights w_j be arbitrary, i.e., adversarial and the profits $P_j \sim \text{Uni}\,(0, 1)$ be uniform distributed. It is important to note that the adversary specifies the weights *before* the profits materialize.

Theorem 3.4. *We have that* $e(\textsc{Greedy}) \geq 1 - O\left(c^{-1}\right)$.

Proof. Let p be an outcome of the random profits and let them be ordered according to efficiency. Consider the solution-candidate x of Greedy and z^* the optimal solution for Fractional Knapsack. Analogously to the deterministic case we have

$$\frac{\text{val}(x)}{\text{val}(x^*)} \geq \frac{\text{val}(x)}{\text{val}(z^*)} \geq \frac{\text{val}(x)}{\text{val}(x) + \alpha p_k} = 1 - \frac{\alpha p_k}{\text{val}(x)} \geq 1 - \frac{1}{\text{val}(x)}.$$

Taking expectations yields $e(\textsc{Greedy}) \geq 1 - \mathbb{E}\left[1/\text{val}(X)\right]$. Thus it suffices to show that $\mathbb{E}\left[1/\text{val}(X)\right] = O\left(c^{-1}\right)$. Consider the following solution s: Sort the items according to weight and include exactly the c smallest items. It is important to note that this solution is constructed *without* looking at the profits. We prove:

(1) For any outcome of the profits we have $\mathrm{val}(s) \leq \mathrm{val}(x)$.

(2) $\mathbb{E}\left[1/\mathrm{val}(X)\right] \leq \mathbb{E}\left[1/\mathrm{val}(s)\right] = O\left(c^{-1}\right)$.

These properties immediately imply the result.

Consider an outcome of the p_j and let S and X denote the *sets* associated with the vectors s and x. Let $p(S)$ and $p(X)$ denote the respective profits. For (1) we have to show $p(S) \leq p(X)$. First of all observe that $S \supseteq X$ is impossible. Why? The set S consists of exactly c items, while X consists of at least c items as all the weights are at most one.

Case 1. If $S \subseteq X$ then x has all the items s has (and possibly some more). Since the profits are non-negative this yields $p(S) \leq p(X)$.

Case 2. Otherwise, there are i and j such that $s_i = 1$ and $x_i = 0$ and $s_j = 0$ and $x_j = 1$. Since

$$w_j \geq \max\{w_\ell : \ell \in S\} \geq w_i$$

and

$$\frac{p_j}{w_j} \geq \min\left\{\frac{p_\ell}{w_\ell} : \ell \in X\right\} \geq \frac{p_i}{w_i}$$

we have that $p_j \geq p_i w_j / w_i \geq p_i$. This means that the item j GREEDY has chosen is more profitable than the item i it has not. Now remove j from X and i from S and repeat the argument until Case 1 holds.

The main tool to prove (2) is the Chernoff bound: Let $X_1, \ldots, X_n \in \{0, 1\}$ be independent indicator variables. For $X = \sum_i X_i$ and any $\delta \geq 0$ it holds that

$$\Pr\left[X \geq (1+\delta)\mathbb{E}\left[X\right]\right] \leq \exp\left(-\frac{\delta^2 \mathbb{E}\left[X\right]}{3}\right).$$

Let the c items in the set S be numbered $1, 2, \ldots, c$. Fix some number k and define that $X_{j,k}$ equals one if $P_j \leq 1/6k$ and zero otherwise. Further define $X_k = \sum_{j=1}^{c} X_{j,k}$. We obviously have that $p(S) \leq c/12k$ implies $X_k \geq c/2$.

$$\begin{aligned}
\mathbb{E}\left[\frac{1}{\mathrm{val}(X)}\right] &\leq \mathbb{E}\left[\frac{1}{\mathrm{val}(s)}\right] \leq \mathbb{E}\left[\frac{1}{p(S)} \;\middle|\; p(S) \geq \frac{c}{12}\right] \Pr\left[p(S) \geq \frac{c}{12}\right] \\
&\quad + \sum_{k=2}^{\infty} \mathbb{E}\left[\frac{1}{p(S)} \;\middle|\; \frac{c}{12k} \leq p(S) \leq \frac{c}{12(k-1)}\right] \Pr\left[\frac{c}{12k} \leq p(S) \leq \frac{c}{12(k-1)}\right] \\
&\leq \frac{12}{c} + \sum_{k=2}^{\infty} \frac{12k}{c} \Pr\left[p(S) \leq \frac{c}{12(k-1)}\right] \leq \frac{12}{c}\left(1 + \sum_{k=2}^{\infty} k \Pr\left[X_k \geq \frac{c}{2}\right]\right) \\
&\leq \frac{12}{c}\left(1 + \sum_{k=1}^{\infty} 2k \Pr\left[X_k \geq \frac{c}{2}\right]\right).
\end{aligned}$$

Here we apply the Chernoff bound: Using $\mathbb{E}\left[X_k\right] = c/6k$ we find

$$\Pr\left[X_k \geq \frac{c}{2}\right] = \Pr\left[X_k \geq (1 + 3k - 1)\mathbb{E}\left[X_k\right]\right] \leq \exp\left(-\frac{(c/6k)(3k-1)^2}{3}\right) \leq \exp\left(-\frac{ck}{2}\right).$$

With this bound we have that

$$\mathbb{E}\left[\frac{1}{\mathrm{val}(X)}\right] \leq \frac{12}{c}\left(1 + \sum_{k=1}^{\infty} 2k \exp\left(-\frac{ck}{2}\right)\right) = O\left(c^{-1}\right)$$

and the result is proved. □

3.1.2 Nemhauser-Ullmann Algorithm

In this section we will give an average-case analysis of an elegant enumeration algorithm, named after its authors, NEMHAUSER-ULLMANN. This algorithm computes an *optimal* solution for the KNAPSACK problem. Not surprisingly, since KNAPSACK is NP-hard, the algorithm has exponential running time in the worst case. However, under some rather weak assumptions on profit distributions, the algorithm has polynomial expected running time. For sake of exposition, we restrict our attention to the (important) special case that profits are uniformly distributed.

One (stupid) way of solving KNAPSACK is to enumerate all subsets of the set of items, check the feasibility of the current candidate, and keep track of the most profitable subset. However, this approach has always exponential running time and enumerates a lot of "uninteresting" subsets. Why? If we know (for some reason) that a certain subset is not optimal, then all its subsets will also not be optimal. Hence it makes no sense enumerating them. A more clever way will be introduced now.

The algorithm uses the following *dominance criterion*. First of all, we may assume without loss of generality that for two solutions $x \neq y$ we have $\text{val}(x) \neq \text{val}(y)$ or $\text{weight}(x) \neq \text{weight}(y)$. Why? Since these solutions are equivalent otherwise, we lose nothing by droping one of them. Let x and y be two solutions. We say that y is *dominated* by x if

$$\text{val}(x) \geq \text{val}(y) \quad \text{and} \quad \text{weight}(x) \leq \text{weight}(y).$$

A solution x is called *dominating* if there is *no* solution $y \neq x$ that dominates x. The intuition is that the solutions that are dominated are redundant: We already have a solution that is not heavier and at least as profitable. The following observation is the basis of the algorithm.

Observation 3.5. *An optimal solution is to be sought among the dominating solutions.*

Let S_j be the set of all dominating solutions only considering the items $\{1, \ldots, j\}$, i.e., the solutions are of the form $x \in \{0,1\}^j 0^{n-j}$. For any two solutions x and y define the operation $z = x \oplus y$ that sets $z_j = 1$ if $x_j = 1$ or $y_j = 1$, and $z_j = 0$ otherwise. Further define the special solution 1_j which is everywhere zero except at item j, where it has the value one.

Algorithm 3.2 NEMHAUSER-ULLMANN

Input. Integer c, vectors $p, w \in \mathbb{N}^n$ with $w_j \leq c$, and $\sum_j w_j > c$.

Output. Vector $x \in \{0,1\}^n$ such that $\text{weight}(x) \leq c$.

(1) Let $S_0 = \{0\}$.

(2) For $j = 1, \ldots, n$ let $T_j = \{x \oplus 1_j : x \in S_{j-1}\}$ and

$$S_j = S_{j-1} \cup T_j - \{y \in S_{j-1} \cup T_j : \ y \text{ is dominated by some } x \in S_{j-1} \cup T_j\}.$$

(3) Return the most profitable solution $x^* \in S_0 \cup \cdots \cup S_n$ with $\text{weight}(x^*) \leq c$.

The running time obviously depends on the cardinalities of the sets $S_0, S_1, \ldots, S_n$. For any j let q_j be an upper bound on S_j. We may assume that $q_0 \leq q_1 \leq \cdots \leq q_n$. It is an exercise to show that the computation of S_j can be implemented to run in time $O(q_j)$. This yields the following bound.

Lemma 3.6. *Let I be an instance of* Knapsack *with n items. We have that*

$$t_I(\text{Nemhauser-Ullmann}) = O\left(nq_n\right),$$

where q_n is any upper bound on $|S_n|$ of the instance I.

Worst-Case Analysis

In the worst case, the running time of Nemhauser-Ullmann is exponential. The proof is left as an exercise.

Theorem 3.7. *We have that $t(\text{Nemhauser-Ullmann}) = \Omega\left(2^n\right)$.*

Average-Case Analysis

Here we give an average-case analysis using the following probabilistic model. It causes no loss of generality to assume that the weights and profits are chosen between zero and one. Specifically, an adversary chooses weights $w_j \in [0,1]$ arbitrarily and the profits are independent and uniformly distributed $P_j \sim \text{Uni}\,(0,1)$. It is important to note that the adversary specifies the weights without knowing the outcomes of the P_j. Then the expected running time of Nemhauser-Ullmann is polynomial.

Theorem 3.8. *We have that $t(\text{Nemhauser-Ullmann}) = O\left(n^4\right)$.*

Let the random variable Q_j denote the number of dominating sets using the elements $\{1,\ldots,j\}$, i.e., the cardinality of the set S_j. The idea of the proof below is to show $\mathbb{E}\left[Q_n\right] = O\left(n^3\right)$.

Proof of Theorem 3.8. Let $m = 2^n$ and define $x_1, x_2, \ldots, x_m$ as the sequence of all vectors $x \in \{0,1\}^n$ according to *non-decreasing weight*, i.e., $\text{weight}(x_1) \leq \cdots \leq \text{weight}(x_m)$, and those with equal weight, with *decreasing profit*.

Let $A_0 = \emptyset$ and for any $k \geq 1$ define the set $A_k = \{x_1, \ldots, x_k\}$ and the number

$$\delta_k = \max_{x \in A_k} \text{val}(x) - \max_{x \in A_{k-1}} \text{val}(x).$$

The crucial observation is that x_k is dominating if and only if $\delta_k > 0$. Why? First of all, since $A_k \supseteq A_{k-1}$ we have $\delta_k \geq 0$. For $\delta_k > 0$ the solution x_k is not dominated by any $y \in A_{k-1}$ since $\text{val}(x_k) > \text{val}(y)$. It is also not dominated by any $y \in A_m - A_k$ since at least $\text{weight}(x_k) < \text{weight}(y)$ or $\text{val}(x_k) > \text{val}(y)$ is true. Thus, if $\delta_k > 0$, then x_k is dominating. Conversely, if $\delta_k = 0$ there is a solution $y \in A_{k-1}$ with $\text{val}(x_k) = \text{val}(y)$ and $\text{weight}(x_k) \geq \text{weight}(y)$, i.e., x_k is dominated by y.

Furthermore observe that we can write $\text{val}(x_k) = \sum_{\ell=1}^{k} \delta_\ell$ for any k. For random profits, δ_k is actually a random variable Δ_k. The following intermediate result (proof below) is the key: It essentially states that *if* a solution is dominating, i.e., $\Delta_k > 0$, then its increase in profit is reasonably large.

Lemma 3.9. *It holds that $\mathbb{E}\left[\Delta_k \mid \Delta_k > 0\right] \geq 1/(32n^2)$.*

Observe that $\mathbb{E}[\mathrm{val}(x_m)] = n/2$. Using Lemma 3.9 we can furthermore write

$$\begin{aligned}\mathbb{E}[\mathrm{val}(x_m)] &= \mathbb{E}\left[\sum_{k=1}^{m} \Delta_k\right] = \sum_{k=1}^{m} \mathbb{E}[\Delta_k] \\ &= \sum_{k=1}^{m} \mathbb{E}[\Delta_k \mid \Delta_k > 0] \Pr[\Delta_k > 0] \\ &\geq \sum_{k=1}^{m} \frac{1}{32n^2} \Pr[\Delta_k > 0]\end{aligned}$$

so we have obtained

$$\sum_{k=1}^{m} \Pr[\Delta_k > 0] \leq 32n^2 \mathbb{E}[\mathrm{val}(x_m)] = 16n^3.$$

We are interested in $\mathbb{E}[Q_n]$. Let D_k be an indicator variable for the event $\Delta_k > 0$. Thus we can write $Q_n = \sum_{k=1}^{m} D_k$ and hence

$$\mathbb{E}[Q_n] = \mathbb{E}\left[\sum_{k=1}^{m} D_k\right] = \sum_{k=1}^{m} \Pr[\Delta_k > 0] \leq 16n^3$$

which yields the theorem (provided that Lemma 3.9 is true). □

Proof of Lemma 3.9. For ease of notation we will drop unnecessary subscripts: Let $x = x_k$ be some solution in the sequence $x_1, x_2, \ldots, x_m$ as defined above and let $\Delta = \Delta_k$ be the associated random variable to $\delta = \delta_k$. We will use this notation below: For two solutions x and y define $z = x \ominus y$ so that $z_j = 1$ if $x_j = 1$ and $y_j = 0$, and $z_j = 0$ otherwise.

We bound $\mathbb{E}[\Delta \mid \Delta > 0]$ from below by

$$\mathbb{E}[\Delta \mid \Delta > 0] \geq \Pr\left[\Delta \geq \frac{1}{16n^2} \;\middle|\; \Delta > 0\right] \cdot \frac{1}{16n^2}$$

and thus it suffices to show that

$$\Pr\left[\Delta \geq \frac{1}{16n^2} \;\middle|\; \Delta > 0\right] \geq \frac{1}{2}.$$

Fix a solution $x = x_k$ and let $x_1, \ldots, x_{k-1}$ be its predecessors in the sequence of solutions. Further define $a_\ell = x \ominus x_\ell$ and $b_\ell = x_\ell \ominus x$. Now we look at the probability we are interested in:

$$\begin{aligned}\Pr\left[\Delta \geq \frac{1}{16n^2} \;\middle|\; \Delta > 0\right] &= \Pr\left[\forall \ell < k\ \mathrm{val}(x) - \mathrm{val}(x_\ell) \geq \frac{1}{16n^2} \;\middle|\; \forall \ell < k\ \mathrm{val}(x) > \mathrm{val}(x_\ell)\right] \\ &= \Pr\left[\forall \ell < k\ \mathrm{val}(a_\ell) \geq \mathrm{val}(b_\ell) + \frac{1}{16n^2} \;\middle|\; \forall \ell < k\ \mathrm{val}(a_\ell) \geq \mathrm{val}(b_\ell)\right] \\ &= \Pr\left[\forall \ell < k\ \mathrm{val}(b_\ell) \leq \mathrm{val}(a_\ell) - \frac{1}{16n^2} \;\middle|\; \forall \ell < k\ \mathrm{val}(b_\ell) \leq \mathrm{val}(a_\ell)\right]\end{aligned}$$

Without loss of generality, the items are indexed such that $x \in \{0,1\}^d 0^{n-d}$ for some d. This yields that $a_\ell \in \{0,1\}^d 0^{n-d}$ and $b_\ell \in 0^d \{0,1\}^{n-d}$.

Now fix the realizations of the profits $p_1, \ldots, p_d$, i.e., we have fixed the values of x and a_ℓ for any $\ell < k$. The remaining profits $p_{d+1}, \ldots, p_n$ may still be varied. Recall the

definition of conditional probabilities $\Pr[A \mid B] = \Pr[A \cap B]/\Pr[B]$ for any two events A and B with $\Pr[B] > 0$. Consider the following sets (that will correspond to events later):

$$A = \left\{(p_{d+1}, \ldots, p_n) \in [0,1]^{n-d} : \forall \ell < k \ \mathrm{val}(b_\ell) \leq \mathrm{val}(a_\ell) - \frac{1}{16n^2}\right\}$$
$$B = \left\{(p_{d+1}, \ldots, p_n) \in [0,1]^{n-d} : \forall \ell < k \ \mathrm{val}(b_\ell) \leq \mathrm{val}(a_\ell)\right\}$$

We obviously have $A \subseteq B$ and thus for each fixed realization of $p_1, \ldots, p_d$ that

$$\Pr\left[\Delta \geq \frac{1}{16n^2} \;\middle|\; \Delta > 0, p_1, \ldots, p_d\right] = \frac{\Pr[A \cap B]}{\Pr[B]} = \frac{\mathrm{vol}(A)}{\mathrm{vol}(B)}.$$

Let $0 \leq \varepsilon \leq 1$ and define the B_ε as the set B scaled by $1 - \varepsilon$

$$B_\varepsilon = \left\{(p_{d+1}, \ldots, p_n) \in [0, 1-\varepsilon]^{n-d} : \forall \ell < k \ \mathrm{val}(b_\ell) \leq (1-\varepsilon)\mathrm{val}(a_\ell)\right\}.$$

Now choose ε such that $B_\varepsilon \subseteq A$ and we find

$$\frac{\mathrm{vol}(A)}{\mathrm{vol}(B)} \geq \frac{\mathrm{vol}(B_\varepsilon)}{\mathrm{vol}(B)} = \frac{(1-\varepsilon)^{n-d}\mathrm{vol}(B)}{\mathrm{vol}(B)} = (1-\varepsilon)^{n-d} \geq 1 - \varepsilon(n-d).$$

In the above reasoning the outcomes $p_1, \ldots, p_d$ where fixed. We derived a bound on the desired probability conditional on these values and a fixed value for ε. Now we have to treat the $P_1, \ldots, P_d$ as random variables but we still keep ε fixed. Hence, then, B_ε and A are random sets. Assume for the moment that the choice $\varepsilon = 1/(4n)$ yields $B_\varepsilon \subseteq A$ with probability at least $3/4$. Then we have

$$\begin{aligned}
\Pr\left[\Delta \geq \frac{1}{16n^2} \;\middle|\; \Delta > 0\right] &= \mathbb{E}\left[\Pr\left[\Delta \geq \frac{1}{16n^2} \;\middle|\; \Delta > 0, P_1, \ldots, P_d\right]\right] \\
&= \mathbb{E}\left[\frac{\mathrm{vol}(A)}{\mathrm{vol}(B)}\right] \geq \mathbb{E}\left[\frac{\mathrm{vol}(B_\varepsilon)}{\mathrm{vol}(B)} \;\middle|\; A \subseteq B_\varepsilon\right] \Pr[A \subseteq B_\varepsilon] \\
&\geq \left(1 - \frac{n-d}{4n}\right) \cdot \frac{3}{4} \geq \frac{3}{4} \cdot \frac{3}{4} \geq \frac{1}{2}
\end{aligned}$$

and it only remains to show that the choice of $\varepsilon = 1/(4n)$ has the desired property.

For any a_ℓ with $\mathrm{val}(a_\ell) \geq 1/(4n)$ and this ε we have

$$\mathrm{val}(a_\ell)(1-\varepsilon) = \mathrm{val}(a_\ell) - \frac{\mathrm{val}(a_\ell)}{4n} \leq \mathrm{val}(a_\ell) - \frac{1}{16n^2}.$$

Thus, whenever $\mathrm{val}(a_\ell) \geq 1/(4n)$, any member of B_ε is also a member of A, which means

that $B_\varepsilon \subseteq A$. This property allows us to finalize the proof:

$$\begin{aligned}
\Pr[B_\varepsilon \subseteq A] &\geq \Pr\left[\mathrm{val}(a_\ell) \geq \frac{1}{4n} \;\middle|\; \forall \ell < k\ \mathrm{val}(b_\ell) \leq \mathrm{val}(a_\ell)\right] \\
&= 1 - \Pr\left[\mathrm{val}(a_\ell) \leq \frac{1}{4n} \;\middle|\; \forall \ell < k\ \mathrm{val}(b_\ell) \leq \mathrm{val}(a_\ell)\right] \\
&\geq 1 - \Pr\left[\exists j \in \{1,\ldots,d\} : P_j \leq \frac{1}{4n} \;\middle|\; \forall \ell < k\ \mathrm{val}(b_\ell) \leq \mathrm{val}(a_\ell)\right] \\
&\geq 1 - \sum_{j=1}^{d} \Pr\left[P_j \leq \frac{1}{4n} \;\middle|\; \forall \ell < k\ \mathrm{val}(b_\ell) \leq \mathrm{val}(a_\ell)\right] \\
&\geq 1 - \sum_{j=1}^{d} \Pr\left[P_j \leq \frac{1}{4n}\right] = 1 - \frac{d}{4n} \\
&\geq \frac{3}{4}.
\end{aligned}$$

Above, $\Pr[P_j \leq 1/(4n) \mid \forall \ell < k\ \mathrm{val}(b_\ell) \leq \mathrm{val}(a_\ell)] \leq \Pr[P_j \leq 1/(4n)]$ is the only location in the whole proof where we use that the profits are uniform distributed. □

3.2 Stochastic Weights

In this section we consider a variant of the problem where the weights W_j are random variables. The only assumption is that the W_j are independent (and of course non-negative).

In order to obtain a truly different problem than before, we consider the following *non-clairvoyant* setting: We learn the actual outcome w_j of the random variable W_j only upon placing it into the knapsack. If we attempt to insert j and the items fits within our current residual capacity, it is accepted irrevocably, otherwise it is rejected. The process ends when the knapsack overflows. The profit p_j of each item is known beforehand and the objective is still to maximize the total profit of accepted items.

It causes no loss of generality to assume that the capacity of the knapsack is equal to one. The set $J = \{1,\ldots,n\}$ denotes the set of items. An actual instance of the problem is the vector p of profits and the vector of probability distributions W for the weights.

Below, adaptive algorithms are allowed to adjust the choice of the next candidate, while non-adaptive algorithms have to specify their ordering in advance. This makes some difference as the following example illustrates.

Example 3.10. The capacity is $c = 1$, $n = 3$, and the profits are $p = (1, 1, 1)$. The weight distributions are $W = (W_1, W_2, W_3)$ with

$$W_1 = \begin{cases} 1/5 & \text{probability } 1/2 \\ 3/5 & \text{probability } 1/2 \end{cases},\ W_2 = 4/5 \text{ probability } 1,\ W_3 = \begin{cases} 2/5 & \text{probability } 1/2 \\ 9/10 & \text{probability } 1/2 \end{cases}.$$

If we have to specify the ordering at which we try the items in advance, then the best ordering is $(1, 2, 3)$. This yields an expected profit of $3/2$.

If we are allowed to choose depending on the current situation in our knapsack, we might want to do as follows: We first insert item 1. In case $W_1 = 1/5$ we insert item 2. Otherwise we insert item 3. Looking at the respective decision tree yields that our expected profit is $7/4$.

We further want to motivate that there are actually applications (in scheduling) for such a setting. Suppose we are given a set of jobs and we can obtain a certain profit p_j from each job j. The actual duration of the job w_j is not entirely known. We rather know the probability distribution for the duration. There is a fixed deadline (i.e., the capacity of the knapsack) and we seek to maximize the profit of the jobs finished before the deadline.

We will consider two types of algorithms, depending on the point in time they have to choose the item that is tried next. The more powerful ones are *adaptive* algorithms: Given the set $I \subseteq J$ that is currently in the knapsack and its residual capacity $0 \leq c \leq 1$, an algorithm Π determines an item $j = \Pi(I, c)$ as the next candidate. We can thus think of such an algorithm as a mapping $\Pi : 2^J \times [0, 1] \to J$ specifying the next item to insert. Notice that the profit of such an algorithm, denoted $\mathrm{val}(\Pi)$ is a random variable. We denote by

$$\textsc{adapt}(p, W) = \max_{\Pi} \mathbb{E}\left[\mathrm{val}(\Pi)\right]$$

the maximal expected profit achievable by adaptive algorithms for the instance (p, W).

By contrast, a *non-adaptive* algorithm π has to specify the ordering in which items are tried in advance, i.e., the algorithm chooses a permutation π of J. We denote by

$$\textsc{non-adapt}(p, W) = \max_{\pi} \mathbb{E}\left[\mathrm{val}(\pi)\right]$$

the maximal expected profit achievable by non-adaptive algorithms for the instance (p, W).

Since non-adaptive are special cases of adaptive algorithms we always have the inequality

$$\textsc{adapt}(p, W) \geq \textsc{non-adapt}(p, W).$$

For any non-adaptive algorithm Alg, that chooses the permutation π, say, and achieves $\textsc{alg}(p, W) = \mathbb{E}\left[\mathrm{val}(\pi)\right]$, the quantity

$$\alpha(\textsc{Alg}) = \sup_{(p,W)} \frac{\textsc{alg}(p, W)}{\textsc{adapt}(p, W)}$$

is called the *adaptivity gap* of the algorithm.

We begin the exposition in Section 3.2.1 with a bound on the expected profit of optimal adaptive algorithms. We continue in Section 3.2.2 and give a non-adaptive algorithm that achieves $7/32 \approx 1/5$ of the expected profit of any optimal *adaptive* algorithm.

We need some additional definitions: The *truncated weight* of an item j is $\tilde{W}_j = \min\{1, W_j\}$ and denote $\mu_j = \mathbb{E}\left[\tilde{W}_j\right]$. This quantity is more useful than the actual weight of an item, since any outcome $w_j > 1$ yields rejection of the item j.

As usual denote for any $x \in \{0, 1\}^n$ the profit $\mathrm{val}(x) = \sum_j p_j x_j$, the weight (now a random variable) $\mathrm{weight}(x) = \sum_j W_j x_j$, and the mean truncated weight $\mathrm{mass}(x) = \sum_j \mu_j x_j$, also called the *mass* of x.

It is an exercise to prove the following bound.

Lemma 3.11. *For any $x \in \{0, 1\}^n$ we have* $\Pr\left[\mathrm{weight}(x) \leq 1\right] \geq 1 - \mathrm{mass}(x)$.

3.2.1 Adaptive Algorithms

Since adaptive algorithms Π can be seen as mappings $\Pi : 2^J \times [0, 1] \to J$ it is not clear that an optimal adaptive policy can be described using polynomial space only. However, a simple relaxation similar to the Fractional Knapsack problem can be used to obtain an upper bound on the expected profit attainable by *any* adaptive algorithm.

Fix an adaptive policy Π and let X denote the (random) items that are inserted successfully. If the weights are deterministic it is clear that $\text{mass}(X) \leq 1$. Perhaps surprisingly, in a stochastic setting we can have $\mathbb{E}[\text{mass}(X)] \geq 1$.

Example 3.12. Suppose we have an infinite number of items, where each profit is one and the weight is one with probability p and zero otherwise. Then the expected number of items we insert successfully is $2/p-1$ since we can continue until two of them instantiate to unit weight. Each item j has mean truncated weight $\mu_j = p$ and thus $\mathbb{E}[\text{mass}(X)] = 2-p$, which can be arbitrarily close to 2.

However, the mass we try to insert can never be more than two in expectation. The idea of proof of the following lemma is the observation that each $\tilde{W}_j$ is bounded by one and we can count at most one item beyond the capacity of one.

Lemma 3.13. *For any instance (with knapsack capacity equal to one) and any adaptive policy, let X denote the (random) vector of items the algorithm attempts to insert. Then $\mathbb{E}[\text{mass}(X)] \leq 2$.*

With this key-lemma, we can bound the expected profit of an optimal adaptive algorithm. Define by $v_j = p_j \Pr[W_j \leq 1]$ the *effective value* of item j which is an upper bound on the expected profit an algorithm can gain if it attempts to insert j. Now we parametrize the Fractional Knapsack relaxation we already know with a value t for the capacity of the knapsack:

$$\Phi(t) = \max\left\{\sum_j v_j x_j : \sum_j \mu_j x_j \leq t, x_j \in [0,1]\right\}.$$

With this formulation we can prove a bound on the expected profit of any optimal adaptive algorithm.

Theorem 3.14. *For any instance (p, W) it holds that $\text{ADAPT}(p, W) \leq \Phi(2)$.*

Proof. Fix an adaptive policy Π and let x denote the (random) vector of items that Π *attempts* to insert into the knapsack. Further let the vector y be $y_j = \Pr[x_j = 1]$.

Given y, the expected mass that Π attempts to insert is $\mathbb{E}[\text{mass}(x)] = \sum_j \mu_j y_j$. On the other hand, Lemma 3.13 yields that $\mathbb{E}[\text{mass}(x)] \leq 2$. But this in turn means that the vector y is feasible for the problem $\Phi(2)$ and we find the bounds

$$\Phi(2) \geq \sum_j v_j y_j \geq \mathbb{E}[\text{val}(\Pi)]$$

which yields the claim. □

3.2.2 Non-Adaptive Algorithms

In the previous section we gave a bound on the largest possible value of any adaptive algorithm, i.e., including an optimal one. However, the question within this section is if also a much simpler non-adaptive algorithm can obtain "good" solutions.

Indeed, we show that a natural greedy approach yields a solution with expected profit $7/32 \approx 1/5$ of that of an optimal adaptive algorithm. We give a randomized algorithm for sake of exposition, but it can easily be derandomized.

Consider the function $\Phi(t)$, which can be seen as the FRACTIONAL KNAPSACK relaxation with capacity t. The usual GREEDY algorithm considers the items in decreasing order of *expected effective efficiency* v_j/μ_j:

$$\frac{v_1}{\mu_1} \geq \cdots \geq \frac{v_n}{\mu_n}.$$

Define $m_k = \sum_{j=1}^{k} \mu_j$ and observe that we have for any $t = m_{k-1} + \xi \in [m_{k-1}, m_k]$

$$\Phi(t) = \sum_{j=1}^{k-1} v_j + \frac{v_k}{\mu_k}\xi.$$

We may assume that $\Phi(1) = 1$ by scaling the v_j by an appropriate factor. And we may assume that there are enough items such that $\sum_{j=1}^{n} \mu_j \geq 1$ (which can be achieved by adding items with profit zero and positive weight).

Now we define the randomized greedy RGREEDY algorithm: Let r be the break index, i.e., the smallest index with $\sum_{j=1}^{r} \mu_j \geq 1$ and let $\mu'_r = 1 - \sum_{j=1}^{r-1} \mu_j$, i.e., the part of item r that fits within capacity one. Let $p' = \mu'_r/\mu_r$, $w'_r = pw_r$. For $j = 1, \ldots, r-1$ set $w'_j = w_j$ and $\mu'_r = \mu_r$. We assume $\Phi(1) = \sum_{j=1}^{r} w'_j = 1$.

Algorithm 3.3 RGREEDY

Input. Integer r, vectors v', μ', real $p' \in [0, 1]$

Output. Vector $x \in \{0, 1\}^n$ such that weight$(x) \leq 1$.

(1) Choose index k with probability w'_k.

(2) If $k < r$ insert item k, i.e., $x_k = 1$. If $k = r$, flip another independent coin and insert item r, i.e., $x_r = 1$ with probability p', otherwise discard it.

(3) Then insert the items $1, 2, \ldots, k-1, k+1, \ldots, r$ in the greedy order, i.e., for $j = 1, 2, \ldots, k-1, k+1, \ldots, r$ set $x_j = 1$ if the item j still fits.

Theorem 3.15. *We have that* $\alpha(\text{RGREEDY}) \geq 7/32$.

Proof. For simplicity assume that $\sum_{j=1}^{r} \mu_j = 1$ and that $\Phi(1) = \sum_{j=1}^{r} v_j = 1$. Then $\text{ADAPT}(p, W) \leq \Phi(2) \leq 2$, but more strongly

$$\text{ADAPT}(p, W) \leq \Phi(2) \leq 1 + \omega,$$

where $\omega = v_r/\mu_r \leq (\sum_{j=1}^{r} v_j)/(\sum_{j=1}^{r} \mu_j) = 1$. With $\sum_{j=1}^{r} \mu_j = 1$, the algorithm has the simpler form:

(1) Choose $k \in \{1, \ldots, r\}$ with probability v_k and insert item k.

(2) Then insert items $1, \ldots, k-1, k+1, \ldots, r$ in the greedy order.

We estimate the expected value achieved by this algorithm, with respect to the randomization of the weights and with respect to our own randomization. Fix any k. The item k is inserted first with probability v_k, inserted after $\{1, \ldots, k-1\}$ with probability $\sum_{j=1}^{k-1} v_j$, and after $\{j, 1, \ldots, k-1\}$ with probability v_j for any $k < j \leq r$. If k is the first

item, then its expected profit is $v_k = p_k \Pr[W_k \leq 1]$. Otherwise we use Lemma 3.11, i.e., a variant of Markov's inequality:

$$\Pr[\text{item } k \text{ fits}] = \Pr\left[\sum_{j=1}^{k} W_j \leq 1\right] \geq 1 - \sum_{j=1}^{k} \mu_j.$$

Similarly for the case when k is preceded by $\{j, 1, \ldots, k-1\}$.

With the obvious observation $1 - \sum_{j=1}^{k} \mu_j \geq v_j(1 - \sum_{j=1}^{k} \mu_j)$ for any j, be denote by c_k the following lower bound on the expected profit of item k:

$$\begin{aligned} c_k &= v_k \left(v_k + \sum_{j=1}^{k-1} v_j (1 - \sum_{\ell=1}^{k} \mu_\ell) + \sum_{k+1}^{r} (1 - \sum_{\ell=1}^{k} \mu_\ell - \mu_j) \right) \\ &= v_k \left(\sum_{j=1}^{r} v_j (1 - \sum_{\ell=1}^{k} \mu_\ell) + v_k \sum_{\ell=1}^{k} \mu_\ell - \sum_{\ell=k+1}^{r} v_j \mu_j \right). \end{aligned}$$

It holds that $\textsc{rgreedy}(p, W) \geq \sum_{k=1}^{r} c_k$ and using $\sum_{j=1}^{r} v_j = 1$ and $\sum_{j=1}^{r} \mu_j = 1$ we simplify the bound to

$$\begin{aligned} \textsc{rgreedy}(p, W) &\geq \sum_{k=1}^{r} v_k \left(1 - \sum_{j=1}^{k} \mu_j + v_k \sum_{j=1}^{k} \mu_j - \sum_{j=k+1}^{r} w_j \mu_j \right) \\ &= 1 + \sum_{j \leq k \leq r} (-v_k \mu_j + v_k^2 \mu_j) - \sum_{k \leq j \leq r} v_k v_j \mu_j \\ &= 1 + \sum_{j \leq k \leq r} (-v_k \mu_j + v_k^2 \mu_j + v_k v_j \mu_j) - \sum_{j,k=1}^{r} v_k v_j \mu_j \\ &= 1 + \sum_{j \leq k \leq r} v_k \mu_j (v_j + v_k - 1) - \sum_{j=1}^{r} v_j \mu_j. \end{aligned}$$

To symmetrize this polynomial, we apply the condition of the greedy ordering. For any $j < k$ we have $v_j + v_k - 1 \leq 0$ and the greedy ordering implies $v_k \mu_j \leq v_j \mu_k$, allowing us to replace $v_k \mu_j$ by $(v_k \mu_j + v_j \mu_k)/2$ for all pairs $j < k$. This yields

$$\begin{aligned} \textsc{rgreedy}(p, W) &\geq 1 + \frac{1}{2} \sum_{j \leq k \leq r} (v_k \mu_j + v_j \mu_k)(v_j + v_k - 1) + \sum_{j=1}^{r} v_j \mu_j (2v_k - 1) - \sum_{j=1}^{r} v_j \mu_j \\ &= 1 + \frac{1}{2} \sum_{j,k=1}^{r} v_k \mu_j (v_j + v_k - 1) + \frac{1}{2} \sum_{j=1}^{r} v_j \mu_j (2v_j - 1) - \sum_{j=1}^{r} v_j \mu_j \end{aligned}$$

and using $\sum_{j=1}^{r} v_j = \sum_{j=1}^{r} \mu_j = 1$ gives

$$\geq \frac{1}{2} + \frac{1}{2} \sum_{k=1}^{r} v_k^2 + \sum_{k=1}^{r} v_k^2 \mu_k - \sum_{k=1}^{r} v_k \mu_k.$$

We want to compare this expression to $1 + \omega$, where $\omega = \min\{v_j/\mu_j : j < r\}$. We use the

value of ω to estimate $\sum_{k=1}^{r} v_k^2 \geq \omega \sum_{k=1}^{r} v_k \mu_k$ and we obtain

$$\begin{aligned}
\textsc{rgreedy}(p, W) &\geq \frac{1}{2} + \frac{\omega}{2} \sum_{k=1}^{r} v_k \mu_k + \sum_{k=1}^{r} v_k^2 \mu_k - \sum_{k=1}^{r} v_k \mu_k \\
&= \frac{1}{2} + \sum_{k=1}^{r} \mu_k v_k \left(\frac{\omega}{2} + v_k - 1 \right) \\
&\geq \frac{1}{2} - \sum_{k=1}^{r} \mu_k \left(\frac{1}{2} - \frac{\omega}{4} \right)^2
\end{aligned}$$

where the last inequality was obtained from minimizing a quadratic function in v_k for each k. Finally, with $\sum_k \mu_k = 1$ we find

$$\geq \frac{1}{4} + \frac{\omega}{4} - \frac{\omega^2}{16}.$$

We compare this to the adaptive optimum, which is bounded by $1 + \omega$ and minimize for $\omega \in [0, 1]$:

$$\frac{\textsc{rgreedy}(p, W)}{\textsc{adapt}(p, W)} \geq \frac{1}{4} - \frac{\omega^2}{16(1+\omega)} \geq \frac{7}{32}.$$

It remains to remove the assumption that $\sum_{j=1}^{r} \mu_j = 1$. We claim that if $\sum_{j=1}^{r} \mu_j > 1$, the randomized greedy algorithm performs just like the simplified algorithm we just analyzed, on a modified instance with values v'_j and mean sizes μ'_j (so that $\sum_{j=1}^{r} \mu'_j = 1$; see the description of the algorithm). Indeed, $\Phi(1) = 1$ and $\omega = v_r/\mu_r = v'_r/\mu'_r$ in both cases. Hence the bound on $\textsc{adapt}(p, W)$ is the same. For an item $k < r$, our estimate of the expected profit in both instances is

$$c_k = v'_k \left(v'_k + \sum_{j=1}^{k-1} v'_j (1 - \sum_{\ell=1}^{k} \mu'_\ell) + \sum_{k+1}^{r} (1 - \sum_{\ell=1}^{k} \mu'_\ell - \mu'_j) \right).$$

For the original instance, this is because the expected contributions of item r to the total weight, conditioned on being selected first, is $p'\mu_r = \mu'_r$. If it is not selected first it is not counted at all. The expected profit for item r is $c_r = v'_r p' v_r = (v'_r)^2$ in both instances. This reduces the analysis to the case we dealt with already, completing the proof of the theorem. □

Chapter 4

Completion Time Scheduling

4.1 Deterministic Scheduling

Suppose that we are given one machine for scheduling jobs. Any job j of the n given jobs J has a priority, i.e., a *weight* w_j and a *processing requirement* p_j. We assume that executing j takes time p_j and that once we started the job we must execute it until termination, i.e., we say that *preemption* is not allowed. This implies that any schedule, i.e., assignment of jobs to time intervals, can be seen as just an ordering, i.e., a list of the jobs. We denote by $\Pi(J)$ the set of permutations of J. The *total weighted completion time* of a list π is

$$\text{val}(\pi) = \sum_j w_j c_j(\pi),$$

where $c_j(\pi)$ denotes the time when job j finishes under the ordering π. This objective function measures the total (weighted) time the jobs have to wait in the system before completion. Of course, our goal is to minimize the objective function.

Problem 4.1 Completion Time Scheduling

$$\begin{aligned} \text{minimize} \quad & \text{val}(\pi) = \sum_j w_j c_j(\pi) \\ \text{subject to} \quad & \pi \in \Pi(J) \end{aligned}$$

In order to get an intuition for the objective function and that the ordering indeed plays a role, let us have a look at the following instance.

Example 4.1. There are n jobs $J = \{1, \ldots, n\}$, all with weight $w_j = 1$, and the following processing times: $p_1 = p$, $p_2 = \cdots = p_n = 1$, where p is a large number. Consider the orderings

$$\pi = (1, 2, \ldots, n) \quad \text{and} \quad \pi^* = (2, \ldots, n, 1).$$

We find $\text{val}(\pi) = p + p + 1 + p + 2 + \cdots + p + n - 1 = np + n(n-1)/2$ and $\text{val}(\pi^*) = 1 + 2 + \cdots + n - 1 + n - 1 + p = n(n-1)/2 + n - 1 + p$. So if we choose p substantially larger than n^2, then the ratio $\text{val}(\pi)/\text{val}(\pi^*)$ is arbitrarily close to n. We will see very soon how to solve this problem in polynomial time (which will imply that π^* is optimal).

The above example indicates that one might want to schedule the jobs in increasing order of processing time. For the weighted version it may be a good idea to schedule the

jobs in increasing order of the ratio p_j/w_j, called *efficiency*. So we may assume that the jobs are given in the *greedy order*

$$\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq \cdots \leq \frac{p_n}{w_n}.$$

For any list π denote by $j \leq^\pi k$ that job j is not after job k in π; analogously $j <^\pi k$ for j before k. In any ordering π, whenever some job j is somewhere before a job k, i.e., $j <^\pi k$, such that $p_j/w_j > p_k/w_k$, then j and k from an *inversion*. Notice that the greedy ordering has no inversions.

Theorem 4.2. *The greedy order is optimal for total weighted completion time scheduling.*

Proof. We show that eliminating an inversion strictly improves the objective function. Let there be an inversion, otherwise there is nothing to show. Then there is an inversion where the involved job j directly precedes the other one k, say, i.e., the ordering π has the form $\pi = (\ldots, j, k, \ldots)$. Swap j and k and leave the rest of the ordering unchanged yielding the ordering $\pi' = (\ldots, k, j, \ldots)$. Now π' has exactly one inversion less than π.

We obviously have

$$\mathrm{val}(\pi') = \mathrm{val}(\pi) - w_k p_j + w_j p_k.$$

Since j and k from an inversion it holds that $-w_k p_j + w_j p_k < 0$ since $p_j/w_j > p_k/w_k$ completing the proof. □

4.2 Stochastic Non-Clairvoyant Scheduling

In the preceding section we assumed that the processing requirements p_j are given at the outset. However, this assumption is not always justified. For example, in computer science we usually have no idea how long algorithms, respectively programs we run usually take. Instead, we may want to learn about the "typical" duration by recording the respective data from earlier executions and base our scheduling decisions on that data.

In this section we are given the probability distribution P_j of the processing requirement of any job j. Our only assumptions will be that the jobs are independent and take integer values. Notice that we learn the outcome p_j of the random variable P_j only upon completing the job j. And, as before, the priorities w_j are given in advance and we are not allowed to interrupt a job once begun.

So we are interested in algorithms that determine (in advance) an ordering of the jobs and then schedule the jobs according to this. In the notation of Chapter 3, this is a *non-adaptive* algorithm. In contrast to Chapter 3 we do not compare a non-adaptive with an adaptive algorithm. Instead, we compare (in expectation) the solution our algorithm produces directly to the optimal solution given the whole data, i.e., the w_j and the p_j in advance.

More precisely, let ALG be an algorithm that chooses the ordering π, say, on an instance $P = (P_1, \ldots, P_n)$ and $w = (w_1, \ldots, w_n)$. For any outcome p of P denote by

$$\mathrm{ALG}(p, w) = \mathrm{val}(\pi) = \sum_j w_j c_j(\pi) = \sum_j w_j \Big(\sum_{k \leq^\pi j} p_k\Big) = \sum_j \sum_{k >^\pi j} w_k p_j$$

the total weighted completion time of that algorithm. $\mathrm{ALG} = \mathrm{ALG}(P, w)$ denotes the induced random variable. Given any outcome p of P, let

$$\mathrm{OPT} = \mathrm{OPT}(p, w) = \min_{\pi \in \Pi(J)} \mathrm{val}(\pi)$$

be the optimal objective value. Let $\textsc{opt} = \textsc{opt}(P, w)$ be the induced random variable and Π^* the respective random ordering. As mentioned already, we measure the quality of an algorithm with the expected value of the approximation ratio

$$e(\textsc{alg}) = \mathbb{E}\left[\frac{\textsc{alg}}{\textsc{opt}}\right],$$

where the expectation is taken with respect to the processing time distribution.

Greedy Algorithm

Maybe the most natural algorithm is to determine the expected values $\mu_j = \mathbb{E}[P_j]$ and to schedule the jobs in the ordering

$$\frac{\mu_1}{w_1} \leq \frac{\mu_2}{w_2} \leq \cdots \leq \frac{\mu_n}{w_n}.$$

Let this algorithm be called Egreedy.

The question is how good is this algorithm? Suppose that the algorithm scheduled the job j before k since $\mu_j/w_j \leq \mu_k/w_k$, but the outcome is such that $p_j/w_j > p_k/w_k$. This means that the algorithm scheduled the jobs j and k in the wrong order. How large is the contribution to the objective function of this error?

If we look at the proof of Theorem 4.2 we see that a pair of adjacent jobs that are in the wrong order contribute $w_j p_k - w_k p_j$ to the objective function. They are in wrong position if and only if $p_j/w_j > p_k/w_k$. This suggests the following definitions: For any pair j and k of jobs define

$$\delta_{jk} = w_j p_k - w_k p_j,$$

and

$$x_{jk} = \begin{cases} 1 & \text{if } p_j/w_j > p_k/w_k, \\ 0 & \text{otherwise.} \end{cases}$$

The intuition here is that δ_{jk} measures the influence of the error, i.e., of the inversion of j and k, to the objective function and x_{jk} triggers if this error had occurred. Thus the following lemma, whose proof is left as an exercise, comes to mind: Let π be an arbitrary ordering and π^* be an optimal ordering for the instance p and w.

Lemma 4.3. *We have* $\mathrm{val}(\pi) = \mathrm{val}(\pi^*) + \sum_j \sum_{k >^\pi j} \delta_{jk} x_{jk}$.

For technical reasons in the analysis below, we need a little generalized version of this lemma. We alter the definition of δ_{jk} a little bit, but x_{jk} stays the same: For a pair j and k of jobs and any number $\beta \geq 1$ define $\delta_{jk} = w_k p_j - \beta w_j p_k$. This yields a similar result.

Lemma 4.4. *For any* $\beta \geq 1$ *we have* $\mathrm{val}(\pi) \leq \beta \mathrm{val}(\pi^*) + \sum_j \sum_{k >^\pi j} \delta_{jk} x_{jk}$.

Proof. Define the variable $y_{jk}(\pi)$ that is one if $k >^\pi j$ and otherwise zero. We use $y_{jk} = y_{jk}(\pi)$ and $y^*_{jk} = y_{jk}(\pi^*)$ as a shorthand. Observe that $x_{jk} = 1$ if and only if $y_{jk} = 1$ and $y^*_{jk} = 0$. To see this recall that $y^*_{jk} = 0$, i.e., $k \leq^{\pi^*} j$ implies $p_k/w_k \leq p_j/w_j$ and hence $\delta_{jk} \geq 0$. Further note that we have $y_{jk} = 1 - y_{kj}$ and $y^*_{jk} = 1 - y^*_{kj}$ for $j \neq k$.

For every list π it holds that

$$\mathrm{val}(\pi) = \sum_j \sum_{k \geq^\pi j} w_k p_j = \sum_j \sum_k w_k p_j y_{jk}(\pi) + \sum_j w_j p_j$$

Using $\beta \geq 1$ we calculate

$$\begin{aligned}
\mathrm{val}(\pi) - \beta\mathrm{val}(\pi^*) &\leq \sum_j \sum_k w_k p_j (y_{jk} - \beta y^*_{jk}) \\
&= \sum_j \sum_{k >^\pi j} \left(w_k p_j (y_{jk} - \beta y^*_{jk}) + w_j p_k (y_{kj} - \beta y^*_{kj}) \right) \\
&\leq \sum_j \sum_{k >^\pi j} \left(w_k p_j (y_{jk} - y^*_{jk}) - w_j p_k ((\beta - \beta y_{kj}) - (\beta - \beta y^*_{kj})) \right) \\
&= \sum_j \sum_{k >^\pi j} \left(w_k p_j (y_{jk} - y^*_{jk}) - \beta w_j p_k (y_{jk} - y^*_{jk}) \right) \\
&= \sum_j \sum_{k >^\pi j} (w_k p_j - \beta w_j p_k)(y_{jk} - y^*_{jk}) = \sum_j \sum_{k >^\pi j} \delta_{jk} (1 - y^*_{jk}) \\
&= \sum_j \sum_{k \geq^\pi j} \delta_{jk} x_{jk}
\end{aligned}$$

and the proof is complete. □

We need additional notation before stating the result. Let j and k be two jobs with associated weights w_j and w_k such that $\mu_j / w_j \leq \mu_k / w_k$. We denote by Δ_{jk} and X_{jk} the induced random variables by δ_{jk} and x_{jk}. Define the quantity $\alpha_{jk} = \Pr[X_{jk} = 1] = \Pr[P_j / w_j > P_k / w_k]$ and $\alpha = \max_{jk} \alpha_{jk}$. Notice that the probability that the EGREEDY algorithm finishes the jobs j and k in wrong order in comparison to the optimum is α_{jk}. It will turn out that the expected approximation ratio depends on the worst of these probabilities, i.e., on α.

Here is a little bit technical definition. The intuition is that we want to be able to say something about the (expected) contribution of an error, provided that it has occurred: For every distribution there is a number $\beta \geq 1$ such that for all j and k as above

$$\mathbb{E}\left[\frac{w_k P_j - \beta w_j P_k}{\mathrm{OPT}} \;\middle|\; w_k P_j > w_j P_k \right] \leq \mathbb{E}\left[\frac{w_k P_j}{\mathrm{OPT}} \right] \tag{4.1}$$

and $\Pr[P_j / w_j > P_k / w_k] \leq \alpha$.

Now we state a bound on the expected approximation ratio of the EGREEDY algorithm that depends on the parameters α and β as defined above.

Theorem 4.5. *We have that* $e(\textsc{Egreedy}) = \mathbb{E}[\textsc{egreedy}/\textsc{opt}] \leq \beta/(1-\alpha)$.

For an example application of the result, suppose that the jobs are independent geometric distributed (with arbitrary expectations).

Corollary 4.6. *For geometric processing time distributions we have* $e(\textsc{Egreedy}) \leq 2$.

Proof. It suffices to prove that the geometric distribution allows the choice $\beta = 1$ and $\alpha \leq 1/2$. It is straightforward to show that the geometric distribution yields

$$\alpha_{jk} = w_k \mu_j / (w_k \mu_j + w_j \mu_k) \leq 1/2$$

and it is an exercise to prove the following lemma.

Lemma 4.7. *Let* $h : \mathbb{N}_1 \times \cdots \times \mathbb{N}_n \to \mathbb{N}$ *be a monotone non-decreasing function. Let* $P = (P_1, \ldots, P_n)$ *denote the vector of independent geometric distributed random variables* P_j*. Then it holds for any* j *and any integer* $t \geq 0$ *that*

$$\mathbb{E}\left[\frac{P_j - t}{h(P)} \;\middle|\; P_j > t \right] \leq \mathbb{E}\left[\frac{P_j}{h(P)} \right].$$

It is not hard to see that, since the jobs are assumed to be independent, t *is* allowed to be the random variable $w_j P_k$. Observe also that OPT is a monotone non-decreasing function. (Why?) So we apply the lemma to the random variable $w_k P_j$ and see that we are allowed to choose $\beta = 1$ in (4.1). □

Proof of Theorem 4.5. Let the n jobs be indexed such that $\mu_1/w_1 \leq \cdots \leq \mu_n/w_n$, i.e., in the greedy order. Let $\pi = (1, \ldots, n)$ be the list of the jobs in that order. We use Lemma 4.4 and property (4.1) to find

$$\begin{aligned}
\mathbb{E}\left[\frac{\text{EGREEDY}}{\text{OPT}}\right] &= \mathbb{E}\left[\frac{\text{val}(\pi)}{\text{OPT}}\right] = \mathbb{E}\left[\frac{\beta\text{val}(\Pi^*) + \text{val}(\pi) - \beta\text{val}(\Pi^*)}{\text{OPT}}\right] \\
&\leq \beta + \sum_j \sum_{k >^\pi j} \mathbb{E}\left[\frac{\Delta_{jk} X_{jk}}{\text{OPT}}\right] \\
&= \beta + \sum_j \sum_{k >^\pi j} \mathbb{E}\left[\frac{w_k P_j - \beta w_j P_k}{\text{OPT}} \;\middle|\; X_{jk} = 1\right] \Pr\left[X_{jk} = 1\right] \\
&\leq \beta + \sum_j \sum_{k >^\pi j} \alpha \mathbb{E}\left[\frac{w_k P_j}{\text{OPT}}\right] \\
&\leq \beta + \alpha \mathbb{E}\left[\frac{\sum_j \sum_{k >^\pi j} w_k P_j}{\text{OPT}}\right] \\
&= \beta + \alpha \mathbb{E}\left[\frac{\text{EGREEDY}}{\text{OPT}}\right].
\end{aligned}$$

Rearranging completes the proof. □

Chapter 5

Paging

Memory management is a fundamental problem in computer architecture and operating systems. In its simplest form, the memory system consists of two levels: a fast but small cache and a slow but large main memory. The cache is a temporary memory for data items needed. Copying from main memory to the cache takes time, but has the advantage that the data in the cache can be accessed fast. A replacement strategy decides which data item is to be evicted in case an item which is currently missing in the cache is requested. It is well-known that the chosen replacement strategy is crucial for the overall system performance.

The underlying theoretical problem is known as the PAGING problem. We are given a collection of data items, called *pages*, and cache of *size* k, which means that it can store up to k pages. Further, a sequence of page-requests has to be served by making each requested page available in the fast memory. If the page is already cached, the request is served without cost. Otherwise, a *page fault* occurs and the requested page must be brought to fast memory, which might involve the removal of a currently cached page. The objective function of a replacement strategy is to minimize the number of page faults.

In the *offline* version of the problem, the whole request sequence is known in advance. The LONGEST-FORWARD-DISTANCE algorithm (denoted OPT), is an optimal offline algorithm. The algorithm always evicts the page with the most distant next request. In *online* paging, requests arrive one-by-one and eviction decisions must be made at every arriving request, without any knowledge of the future. Examples for online strategies are LEAST-RECENTLY-USED (LRU), which evicts the page in the cache whose last access was earliest, and FIRST-IN-FIRST-OUT (FIFO), which evicts the page that has been in fast memory longest.

It is common to evaluate the performance of online algorithms by using competitive analysis. In that model, an online algorithm ALG is compared to an optimum offline algorithm OPT on the same input sequence. The request sequence is chosen by an adversary A out of a set S_A of admissible sequences. Denote the respective number of faults of ALG and OPT on the request sequence R by $\text{ALG}(R)$ and $\text{OPT}(R)$. An algorithm is said to be *c-competitive* against the adversary A if

$$\text{ALG}(R) \leq c \cdot \text{OPT}(R) + \gamma \quad \text{for all } R \in S_A,$$

where $c = c(k)$ may depend on k but not on the length of R and γ is a constant independent of both. The smallest such c is called the *competitive ratio* of the algorithm and denoted $r_A(\text{ALG})$. If no such c exists, then the algorithm is called *not competitive*.

It is known that LRU (and several other paging algorithms including FIFO) are k-competitive against the unrestricted adversary, and that this bound is tight. However, the

theoretical k-competitiveness seems to be overly pessimistic to be meaningful for practical situations. For example, in an experimental study the observed competitive ratio of LRU ranged between one and three but was frequently around 3/2. This gap between theoretical worst-case bounds and practically observed behaviour called for research. There is consensus that the practical success of LRU (and other algorithms) is due to a phenomenon called *locality of reference*. This means that a requested page is likely to be requested again in the near future. Thus a lot of the related theoretical work was devoted to modeling locality appropriately. The two most common ways are randomized adversaries, where request sequences obey an underlying probability distribution and deterministic adversaries, for which the set of admissible sequences is restricted explicitly.

Overview

One difficulty in analyzing the paging problem was the lack of a strong lower bound for the minimum number of faults needed to serve a given request sequence. So far, this number was bounded from below by partitioning the sequence into phases, giving insight only into *local* structure of the paging problem.

In Section 5.1 we give an analysis, which is based on a lower bound that captures the structure of the *entire* sequence. Furthermore, the framework characterizes the regime in which an offline optimal strategy can substantially outperform LRU. Two requests to the same page with exactly ℓ distinct pages inbetween are called a *pair* with *distance* ℓ. Define the *characteristic vector*

$$\mathbf{c}(R) = (c_0(R), c_1(R), \ldots, c_{p-1}(R))$$

of a sequence R, where every entry $c_\ell(R)$ counts the number of pairs in the sequence with distance exactly ℓ. Here $p = p(R)$ denotes the number of distinct pages accessed in R. This definition allows us to characterize the number of faults of LRU because it will have evicted a certain page if and only if at least k distinct pages were requested since the last request to that page. Therefore we have

$$\textsc{lru}(R) = \sum_{\ell \geq k} c_\ell(R) + p(R).$$

The crucial result for our analysis is the lower bound

$$\textsc{opt}(R) \geq \frac{1}{2} \sum_{\ell \geq k} \frac{\ell - k + 1}{\ell} c_\ell(R).$$

The similar form of these bounds allows direct comparison of the number of faults of LRU and OPT on any given sequence and any cache size k.

In Section 5.2 we propose a deterministic adversary, which is only mildly restricted by two parameters. The adversary not only captures locality of reference, but also another phenomenon which has not been considered in previous theoretical work: *typical memory access patterns*. The idea here is that if data is accessed, many of the items will be needed again, but *not* in the near future. For those requests *any* algorithm, especially also an optimum one, will most likely incur a fault. We explain the favorable observed competitive ratio to a large extend with this phenomenon. In specific, we prove a bound for the competitive ratio of LRU which depends on the parameters of our adversary. We argue that the values for these in real programs are indeed such that our bound gives values between two and five, coming relatively close to the competitive ratios observed in pratice.

We formalize this intuition with the (α, β)-*adversary*. Let $\alpha > 1$ and $\beta \geq 0$ be fixed. The (α, β)-adversary is free to choose any sequence R respecting the constraint

$$\sum_{\ell=k}^{\alpha k-1} c_\ell(R) \leq \beta \sum_{\ell=\alpha k}^{p-1} c_\ell(R).$$

The intuition behind this definition is that there are not "too many" request pairs with "critical" distance between k and $\alpha k - 1$: The pairs with distance less than k do no harm because neither LRU nor OPT fault there. On the other extreme, both, LRU and OPT will fault on many pairs with distance at least αk. But the ones where OPT might outperform LRU are those with distance inbetween.

Our analysis yields that the competitive ratio of LRU for this class of sequences is bounded by

$$r_{(\alpha,\beta)}(\text{LRU}) \leq 2(1+\beta)\frac{\alpha}{\alpha - 1}.$$

Observe that, depending on the values for α and β, our bound can actually become as small as two. Furthermore, the notions of the characteristic vector and the (α, β)-adversary are a formal framework describing on which sequences LRU performs good and explains why it performs poorly on others. We performed extensive experiments and calculated feasible values for α and β. The results supported the intuition that the sequences in practice feature "large" α and "small" β. Our bound for the competitive ratio of LRU on these sequences ranged between two and five, coming close to practical experience. See Figure 5.2 and Figure 5.3 for two representative results.

5.1 Competitive Analysis

Characteristic Vector

Let $M = \{1, 2, \ldots, m\}$ be the set of *pages* stored in the large memory. Let $R = (r_1, r_2, \ldots, r_n)$ denote a sequence of *requests* of *length* n, where we refer to the set $T = \{1, 2, \ldots, n\}$ as *time*. We require $r_t \in M$ for every $t \in T$ and call $P = \{r_1, \ldots, r_p\} \subseteq M$ the set of *requested pages*. Throughout, $p = p(R) = ||P||$ counts the number of requested pages in the sequence.

A *request pair* (i, j) is defined by two time-indices i and j within R such that $1 \leq i < j \leq n$. We call a request-pair (i, j) *consecutive* if $r_i = r_j$ and all pages r_ℓ for $i < \ell < j$ are distinct from r_i. Let

$$C = \{(s, t) : \text{consecutive request pair } (s, t) \text{ in } R\}$$

denote the set of consecutive request pairs.

Further define the set of distinct pages *between* (i, j) by $D(i, j) = \{r_{i+1}, \ldots, r_{j-1}\}$ and define the *distance* of a consecutive request pair (i, j) by $\text{dist}(i, j) = ||D(i, j)||$. Let

$$C_\ell(R) = \{(s, t) \in C : \text{dist}(s, t) = \ell\}$$

denote the set of consecutive pairs with distance ℓ.

The *characteristic vector* of a sequence R is defined by

$$\mathbf{c}(R) = (c_0(R), c_1(R), \ldots, c_{p-1}(R)),$$

where $c_\ell(R) = ||C_\ell(R)||$. Observe that each $c_\ell(R)$ counts the number of consecutive pairs (i, j) that have exactly ℓ distinct pages between i and j.

Bounds for LRU and OPT

The characteristic vector enables us to derive algebraic descriptions for the number of faults of LRU, see Theorem 5.1. Furthermore, it allows us to bound the number of faults of any optimal offline paging algorithm OPT from below, see Theorem 5.2. Especially the latter result is crucial for our analysis.

Theorem 5.1. *Let k denote the cache size and let R be a request sequence with characteristic vector $\mathbf{c}(R)$, then*

$$\textsc{lru}(R) = \sum_{\ell \geq k} c_\ell(R) + p(R).$$

Proof. If a certain page is in the cache, it will be evicted by LRU before it is requested the next time if and only if there are requests to at least k distinct pages before that. The additional $p(R)$ in the bound is due to the initial faults, i.e., the faults incurred when the page is brought to the cache for the first time. □

Theorem 5.2. *Let k denote the cache size and let R be a request sequence with characteristic vector $\mathbf{c}(R)$ and $p(R) \geq k+1$, then*

$$\textsc{opt}(R) \geq \frac{1}{1 + \frac{k-1}{k} - \frac{k-1}{p-1}} \sum_{\ell \geq k} \frac{\ell - k + 1}{\ell} c_\ell(R).$$

The bound is tight.

For practical applications, we can assume that the number of distinct requested pages p is large. Hence we state the following corollary for future reference.

Corollary 5.3. *Let k denote the cache size and let R be a request sequence with characteristic vector $\mathbf{c}(R)$ and $p(R) \geq k+1$, then*

$$\textsc{opt}(R) \geq \frac{1}{2} \sum_{\ell \geq k} \frac{\ell - k + 1}{\ell} c_\ell(R).$$

For an intuitive understanding of this bound, consider an $(\ell+1)$-*multicycle*, i.e., a sequence of the form $(1, 2, \ldots, \ell+1, 1, 2, \ldots, \ell+1, \ldots)$. It is easy to see that the optimum algorithm faults $\ell - k + 1$ times per repetition of the cycle $(1, 2, \ldots, \ell+1)$, and that every consecutive request pair has distance ℓ. Hence, averaging over the whole sequence yields that each consecutive request pair contributes $(\ell - k + 1)/\ell$ to the total number of faults.

The lower bound provides the abstraction that an *arbitrary* sequence can be seen as if it were a *collection of multicycles* with the same characteristic vector.

Before we proceed to the proof of Theorem 5.2, we want to give some intuition about the obstacles we have to overcome. Let R be a request sequence which requests p distinct pages; then

$$\textsc{opt}(R) \geq \max_{\ell \geq k} \left\{ \frac{\ell - k + 1}{\ell} c_\ell(R) \right\},$$

where k denotes the cache size. To see this, observe that OPT faults at least $\ell - k + 1$ times between two consecutive requests to a page, with ℓ distinct pages inbetween. This is because at the point in time of the first request it has cached at most $k - 1$ of the pages inbetween. Therefore, by considering only consecutive request pairs of distance ℓ, we overcount by at most a factor ℓ because every fault is counted for at most ℓ consecutive request pairs. Observe that the bound is tight for $(\ell+1)$-multicycles.

Theorem 5.2 states that the max in the above lower bound can be replaced by a *sum*, losing a factor which is at most two. The main difficulty for the proof is that consecutive request pairs of different distances may overlap, which has to be handled appropriately. We make use of amortized analysis to deal with it. The remainder of this section is devoted to the proof.

Proof of Theorem 5.2. Recall that C denotes the set of all consecutive request pairs and also recall the set T which is referred to as time. Consider an execution of an arbitrary optimal offline algorithm OPT on a given sequence R, e.g., LONGEST-FORWARD-DISTANCE. Define

$$F = F(R) = \{t \in T : \text{OPT faults at time } t\} \tag{5.1}$$

and

$$f_t(R) = \begin{cases} 1 & \text{if } t \in F(R), \\ 0 & \text{otherwise,} \end{cases}$$

which yields

$$\text{OPT}(R) = \sum_{t \in T} f_t(R).$$

Let $t \in T$ be a point in time and let $(i, j) \in C$ be a consecutive request pair. We prove that there exitsts a function $\delta_t(i, j)$ which has the two properties expressed in Claim 5.4 and Claim 5.5 below.

Claim 5.4. *Let $(i, j) \in C$. If $\text{dist}(i, j) \geq k$, then it holds that*

$$\sum_{t \in T} \delta_t(i, j) = \frac{\text{dist}(i, j) - k + 1}{\text{dist}(i, j)}.$$

Otherwise, i.e., if $\text{dist}(i, j) < k$, then $\sum_{t \in T} \delta_t(i, j) = 0$.

Claim 5.5. *For every $t \in T$ we have that*

$$\sum_{(i,j) \in C} \delta_t(i, j) \leq \left(1 + \frac{k-1}{k} - \frac{k-1}{p-1}\right) f_t(R).$$

Suppose that Claim 5.4 and Claim 5.5 hold. Then the following argument completes the proof of the lower bound.

$$\begin{aligned}
\text{OPT}(R) &= \sum_{t \in T} f_t(R) \\
&\overset{\text{(Claim 5.5)}}{\geq} \frac{1}{1 + \frac{k-1}{k} - \frac{k-1}{p-1}} \sum_{t \in T} \sum_{(i,j) \in C} \delta_t(i, j) \\
&= \frac{1}{1 + \frac{k-1}{k} - \frac{k-1}{p-1}} \sum_{(i,j) \in C} \sum_{t \in T} \delta_t(i, j) \\
&\overset{\text{(Claim 5.4)}}{=} \frac{1}{1 + \frac{k-1}{k} - \frac{k-1}{p-1}} \sum_{\substack{(i,j) \in C, \\ \text{dist}(i,j) \geq k}} \frac{\text{dist}(i, j) - k + 1}{\text{dist}(i, j)} \\
&= \frac{1}{1 + \frac{k-1}{k} - \frac{k-1}{p-1}} \sum_{\ell \geq k} \frac{\ell - k + 1}{\ell} c_\ell(R).
\end{aligned}$$

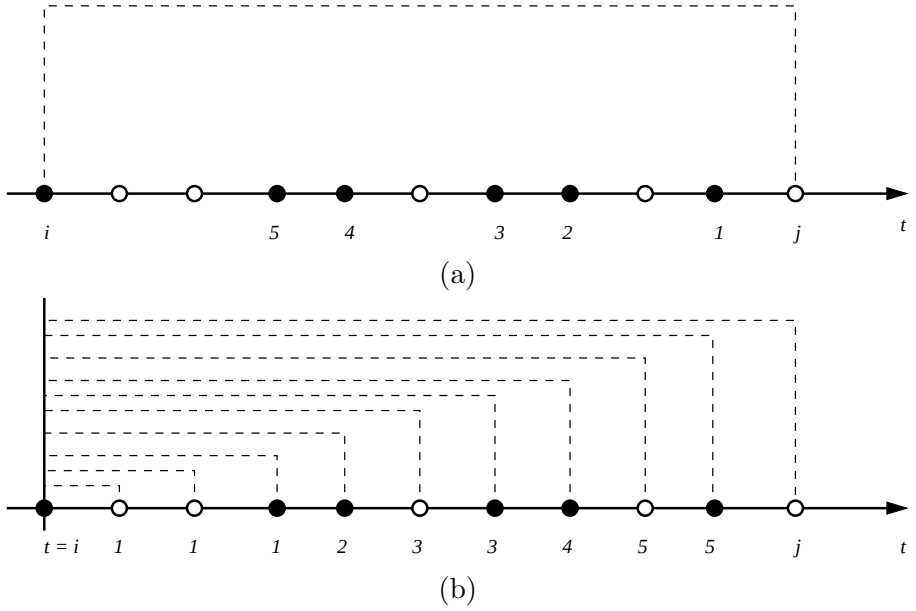


Figure 5.1: Two views on the rank: fixed pair and fixed time.

It is easy to see from the discussion preceeding this proof, that the bound is tight for $(k+1)$-multicycles $(1, 2, \ldots, k+1, 1, 2, \ldots, k+1, \ldots)$. These sequences request $p = k+1$ distinct pages and hence $1 + (k-1)/k - (k-1)/(p-1) = 1$.

Corollary 5.3 is implied from the observation

$$\left(1 + \frac{k-1}{k} - \frac{k-1}{p-1}\right) f_t(R) \leq 2 f_t(R).$$

Below, we present the proofs of Claim 5.4 and Claim 5.5 that complete the proof of the theorem. □

Before we actually prove Claim 5.4 and Claim 5.5 we need additional notation. For every pair $(i, j) \in C$ define the set

$$F(i, j) = \{t \in F : i < t < j\},$$

i.e., the times when Opt faults between i and j. For every $t \in F(i, j)$, define the *rank* of the fault at time t within (i, j) by

$$\text{rank}_t(i, j) = ||\{s \in F(i, j) : s \geq t\}||.$$

The intuition is that the fault with rank r is the r-th *last* fault within (i, j).

Figure 5.1 depicts two perspectives on the rank. Black circles indicate points in time when the optimum faults, white circles the others. It turns out that both perspectives are needed to prove Claim 5.4 and Claim 5.5.

In Figure 5.1 (a), a fixed pair (i, j) is considered and the ranks of the points in time when the optimum faults are shown. These ranks decrease over time.

In Figure 5.1 (b), the fixed time $i = t \in F$ is considered and all the pairs (v, w) with $w > t$ (and $v < t$) are shown. The numbers below the time axis indicate the respective rank of time t for each pair.

Let

$$L(i,j) = \{t \in F(i,j) : \text{rank}_t(i,j) \leq \text{dist}(i,j) - k + 1\}$$

denote the set of the $\text{dist}(i,j) - k + 1$ *last* faults of (i,j).

Define the *value of* (i,j) *at time* $t \in T$ by

$$\text{val}_1(i,j) = \text{dist}(i,j), \tag{5.2}$$

$$\text{val}_{t+1}(i,j) = \begin{cases} \text{val}_t(i,j) - 1 & \text{if } t \in L(i,j), \\ \text{val}_t(i,j) & \text{otherwise.} \end{cases} \tag{5.3}$$

Observe that the value of a pair (i,j) decreases by exactly one at a time whenever one of the $\text{dist}(i,j) - k + 1$ last faults in $L(i,j)$ occurs. At all other times, the value remains constant. It is not hard to see that the following equality holds. It relates the rank of a fault at time $t \in L(i,j)$ with the value at that time:

$$\text{val}_t(i,j) = \text{rank}_t(i,j) + k - 1. \tag{5.4}$$

Hence $\text{val}_t(i,j) \in \{k, \ldots, \text{dist}(i,j)\}$ for all $t \in L(i,j)$.

We are now in position to define the function $\delta_t(i,j)$ based on $\text{val}_t(i,j)$:

$$\delta_t(i,j) = \begin{cases} \frac{k-1}{\text{val}_{t+1}(i,j)} - \frac{k-1}{\text{val}_t(i,j)} & \text{if } \text{dist}(i,j) \geq k, \\ 0 & \text{otherwise.} \end{cases} \tag{5.5}$$

Observe that $\delta_t(i,j) = 0$ for all times except $t \in L(i,j)$. The two crucial properties of this definition are the following. First, for each pair (i,j) with $\text{dist}(i,j) \geq k$ we have $\sum_{t \in T} \delta_t(i,j) = (\text{dist}(i,j) - k + 1)/\text{dist}(i,j)$, which will be shown below. Second, observe that (5.5) is defined in such a way that the smaller the rank of a fault is, the more it contributes to $\sum_{t \in L(i,j)} \delta_t(i,j)$. This second property is crucially needed for the proof of Claim 5.5 below.

Proof of Claim 5.4. Let $(i,j) \in C$ be a consecutive request pair.

First, let $\text{dist}(i,j) < k$ and observe that the definition (5.5) implies $\sum_{t \in T} \delta_t(i,j) = 0$.

Second, let $\text{dist}(i,j) \geq k$. Observe that the number of faults of Opt within (i,j) is $||F(i,j)|| \geq \text{dist}(i,j) - k + 1$ for all pairs with distance at least k. This is because there are $\text{dist}(i,j)$ distinct pages between i and j and there are at most $k-1$ of these in the cache of Opt at time i. Hence, for pairs (i,j) with $\text{dist}(i,j) \geq k$ the property $||F(i,j)|| \geq \text{dist}(i,j) - k + 1$ implies that $||L(i,j)|| = \text{dist}(i,j) - k + 1$.

Then (5.1), (5.2), (5.3), and (5.5) imply

$$\begin{aligned} \sum_{t \in T} \delta_t(i,j) &= \sum_{t \in F} \delta_t(i,j) = \sum_{t \in L(i,j)} \delta_t(i,j) \\ &= \sum_{t \in L(i,j)} \frac{k-1}{\text{val}_{t+1}(i,j)} - \frac{k-1}{\text{val}_t(i,j)} \\ &= (k-1) \sum_{v=k}^{\text{dist}(i,j)} \frac{1}{v-1} - \frac{1}{v} \\ &= (k-1) \left(\frac{1}{k-1} - \frac{1}{\text{dist}(i,j)} \right) \\ &= (k-1) \frac{\text{dist}(i,j) - k + 1}{(k-1)\text{dist}(i,j)} \\ &= \frac{\text{dist}(i,j) - k + 1}{\text{dist}(i,j)} \end{aligned}$$

and Claim 5.4 is established. □

Before we proceed to the proof of Claim 5.5, we give some of the intuition behind it and introduce additional notation.

Let $t \in F$ be a point in time when an Opt fault occurs. We have to prove that the sum $\sum_{(i,j)\in C} \delta_t(i,j)$ of all the pairs (i,j) for which $\delta_t(i,j) > 0$ is not "too large." As mentioned above (see the discussion after (5.5)), for every consecutive request pair (i,j) with $\mathrm{dist}(i,j) \geq k$, a fault at time t contributes the more to $\sum_{t\in L(i,j)} \delta_t(i,j)$ the smaller the value of that pair is at time t. Hence it is crucial to prove that there are not "too many" pairs with small value at time t. This central property is expressed in Lemma 5.6.

For every $t \in T$ define the set

$$X(t) = \{(i,j) \in C : t \in L(i,j)\}.$$

Observe that if no fault occurs at time t, then $X(t)$ is empty. Otherwise, i.e., for $t \in F$, the set $X(t)$ comprises all the pairs for which the fault at time t is one of the $\mathrm{dist}(i,j) - k + 1$ last faults. Hence, for these pairs the value $\mathrm{val}_t(i,j)$ decreases by definition (5.3) at time $t+1$. Furthermore, observe that

$$\sum_{(i,j)\in C} \delta_t(i,j) = \sum_{(i,j)\in X(t)} \delta_t(i,j),$$

i.e., it actually suffices to consider the pairs $(i,j) \in X(t)$, because $\delta_t(v,w) = 0$ for all the other pairs $(v,w) \notin X(t)$.

For every $\ell = 1, \ldots, p-1$ define the sets and quantities

$$X_\ell(t) = \{(i,j) \in X(t) : \mathrm{rank}_t(i,j) = \ell\}$$
$$x_\ell(t) = ||X_\ell(t)||$$

i.e., the (number of) pairs for which the fault at time t has rank ℓ.

Lemma 5.6. *For all $t \in F$ it holds that*

$$\sum_{\ell=1}^{r} x_\ell(t) \leq r + k - 1$$

for all $r = 1, \ldots, p-k$ and $x_{p-k+1}(t) = \cdots = x_{p-1}(t) = 0$.

Proof. Consider a fixed time $t \in F$. For simplicity of notation, we omit the argument t, e.g., we write X_ℓ and X instead of $X_\ell(t)$ and $X(t)$.

First observe that (5.4) implies $\mathrm{rank}_t(i,j) \leq \mathrm{dist}(i,j) - k + 1 \leq p - k$. Hence $X_{p-k+1} = \cdots = X_{p-1} = \emptyset$ and $x_{p-k+1} = \cdots = x_{p-1} = 0$.

The sets X_ℓ partition the pairs $(i,j) \in X$ according to their rank. Hence the sets X_ℓ induce a disjoint partition of X and we have $||\cup_{\ell=1}^{r} X_\ell|| = \sum_{\ell=1}^{r} ||X_\ell|| = \sum_{\ell=1}^{r} x_\ell$.

For sake of contradiction, fix r such that $\cup_{\ell=1}^{r} X_\ell$ comprises $m > r + k - 1$ pairs. Let these be $(i_1, j_1), \ldots, (i_m, j_m)$. Without loss of generality $t < j_1 < \cdots < j_{m-1} < j_m$. Observe that all the pages $r_t, r_{j_1}, \ldots, r_{j_m}$ are distinct from each other. Hence, at time t, the set $\{r_t, r_{j_1}, \ldots, r_{j_{m-1}}, r_{j_m}\}$ comprises exactly $m+1$ distinct pages. Therefore *any* algorithm with cache size k faults at least $m + 1 - k > r + k - 1 + 1 - k = r$ times. Hence there are *more than* r faults of Opt within the time range $t, \ldots, j_m$. Consider the pair (i_m, j_m) and observe that the fault at time t within that pair has rank more than r, i.e., $\mathrm{rank}_t(i_m, j_m) > r$. Hence $(i_m, j_m) \notin \cup_{\ell=1}^{r} X_\ell$ yields the desired contradiction. □

Proof of Claim 5.5. First observe that for every $t \in T \backslash F$ we have

$$\sum_{(i,j)\in C} \delta_t(i,j) = 0 \leq \left(1 + \frac{k-1}{k} - \frac{k-1}{p-1}\right) f_t$$

by definition of $\delta_t(i,j)$.

Second, let $t \in F$ and consider the set $X(t)$. Recall the disjoint partition $X(t) = X_1(t) \cup \cdots \cup X_{p-k}(t)$. By (5.3), (5.4), and (5.5) each pair $(i,j) \in X_\ell(t)$ contributes

$$\delta_t(i,j) = \frac{k-1}{\ell+k-2} - \frac{k-1}{\ell+k-1}$$

to $\sum_{(i,j)\in X(t)} \delta_t(i,j)$.

Now interpret the x_ℓ as variables of the following linear program, where the constraints (5.7) are implied from Lemma 5.6.

$$\text{maximize} \quad \sum_{\ell=1}^{p-k} x_\ell \left(\frac{k-1}{\ell+k-2} - \frac{k-1}{\ell+k-1}\right) \tag{5.6}$$

$$\text{subject to} \quad \sum_{\ell=1}^{r} x_\ell \leq r+k-1 \quad \text{for } r = 1,\ldots,p-k \tag{5.7}$$

Clearly, the optimum value of the linear program (5.6) immediately yields an upper bound on $\sum_{(i,j)\in X(t)} \delta_t(i,j)$.

Now we prove that the unique optimal solution to (5.6) is $\mathbf{x}^* = (k,1,\ldots,1)$. Observe that this solution $\mathbf{x}^*$ has the property that all constraints in (5.7) are satisfied with equality.

Consider an arbitrary optimal solution $\mathbf{x} = (x_1,\ldots,x_{p-k})$ where not all constraints in (5.7) are satisfied with equality. Let v be the smallest such index, i.e., $\sum_{\ell=1}^{v} x_\ell = v+k-1-\varepsilon$ for some $\varepsilon > 0$.

Suppose $v = p-k$ holds. Then we may increase x_v by ε and achive a feasible solution with larger objective value and hence a contradiction. Otherwise, i.e., for $v < p-k$, consider the solution $\mathbf{x}_\varepsilon = (x_1,\ldots,x_v+\varepsilon, x_{v+1}-\varepsilon,\ldots,x_{p-k})$, which is indeed feasible. We have that the objective value of $\mathbf{x}_\varepsilon - \mathbf{x}$ is

$$\varepsilon\left(\frac{k-1}{(v+k-1)(v+k-2)} - \frac{k-1}{(v+k)(v+k-1)}\right) > 0$$

which is a contradiction. Hence $\mathbf{x}^* = (k,1,\ldots,1)$ is the unique optimum solution to (5.6).

This solution and a simple calculation yields

$$\sum_{(i,j)\in C} \delta_t(i,j) \leq \left(1 + \frac{k-1}{k} - \frac{k-1}{p-1}\right) f_t(R) \tag{5.8}$$

and the proof of Claim 5.5 is complete. □

5.2 Structure of Typical Request Sequences

What does the request sequence of a typical program look like? To answer this question, recall that programs are organized in two parts: *code* and *data*. The algorithms executed by the program are implemented in the code segment, which is usually small compared to the cache size. By *locality of reference*, most consecutive requests to code pages will

have short distance. For example, a lot of time of the control flow is spent in executing loops resulting in an enormous number of requests to few pages. However, the size of the datastructures needed by an algorithm can be assumed to be large compared to the cache size, e.g., databases, matrices, or graphs. This data is mostly accessed in a structured way, e.g., linearly reading, or in an irregular way, e.g., the queries to a database. In both cases, we expect a *typical memory access pattern*: a consecutive request to a page either has small or large distance compared to the cache size.

We argue that request distances can be divided into three classes: short, long, and intermediate. Intuitively, for short distance, "no" (reasonable) algorithm faults, for long distance, "any" algorithm faults (especially any optimal algorithm), but it is not clear for the distances inbetween. As discussed below, the treatment of the latter turns out to be crucial for the performance of a strategy. Let k denote the cache size, let $\alpha > 1$ be such that αk is an integer, and let $\beta \geq 0$. If a request pair has distance in the interval $[0, k-1]$, then its distance is *short*, if it is in the interval $[k, \alpha k - 1]$ it is *intermediate* (also called *critical*), and if it is in the interval $[\alpha k, p-1]$ it is called *long*.

The (α, β)-*adversary* is free to choose any sequence R in the set $S_{(\alpha,\beta)}$ defined by

$$S_{(\alpha,\beta)} = \left\{ R \text{ with } \mathbf{c}(R) : \sum_{\ell=k}^{\alpha k-1} c_\ell(R) \leq \beta \sum_{\ell=\alpha k}^{p-1} c_\ell(R) \right\}. \tag{5.9}$$

The intuition behind this definition is that there must not be "too many" pairs with critical distance compared to the long-distance pairs. This classification of sequences is directly derived from the discussion preceeding this definition. The parameter β controls the number of consecutive requests with distance in the critical interval $[k, \alpha k - 1]$. To see why this interval is critical, consider an $(\ell+1)$-multicycle and observe that each request pair has distance ℓ. For these sequences, the competitive ratio of LRU is $\ell/(\ell - k + 1)$. This function decreases from k to $\alpha/(\alpha-1)$ as ℓ grows from k to $\alpha k - 1$. Notice that we do not restrict the number of requests with short distance, i.e., in the interval $[0, k-1]$. Hence, our model *implicitly* captures a notion of locality of reference. Observe that the interval $[\alpha k, p-1]$ is also not constrained, allowing sequences that have a typical memory access pattern.

Summarizing, the (α, β)-adversary is restricted with the number of pairs with distance the interval $[k, \alpha k - 1]$, but only mildly. We just require that the total number of intermediate requests $\sum_{\ell=k}^{\alpha k-1} c_\ell(R)$ can be balanced with the long-distance requests $\sum_{\ell=\alpha k}^{p-1} c_\ell(R)$.

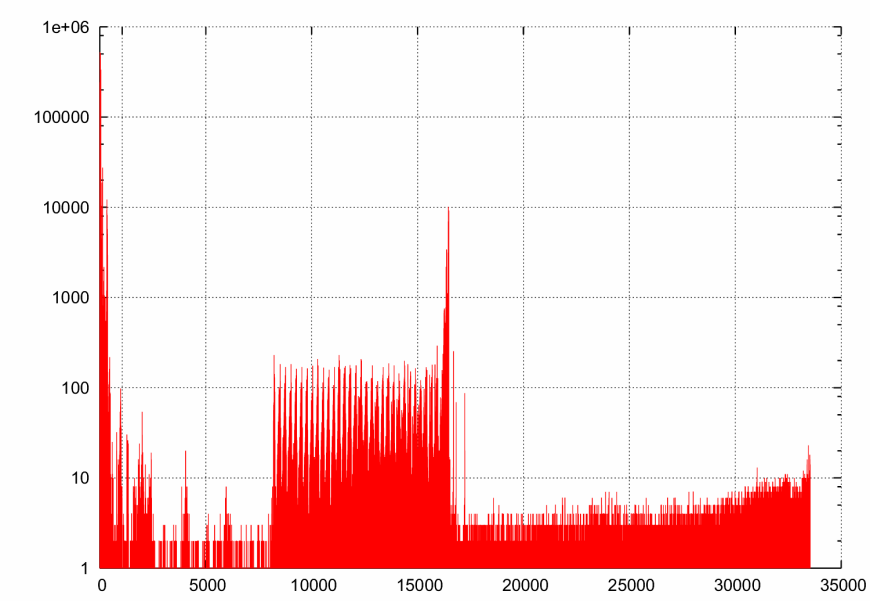
Theorem 5.7. *For the (α, β)-adversary and cache size k, it holds that*

$$r_{(\alpha,\beta)}(\text{LRU}) \leq f(\alpha, \beta) := 2(1+\beta)\frac{\alpha}{\alpha - 1}.$$

Proof. Let $R \in S_{(\alpha,\beta)}$ be an admissible sequence. Further, with Theorem 5.1, Theorem 5.2, (5.9), and $\sum_{\ell \geq k} \frac{\ell-k+1}{\ell} c_\ell(R) \geq (\alpha-1)/\alpha \sum_{\ell=\alpha k}^{p} c_\ell(R)$ we find

$$\begin{aligned}
\text{LRU}(R) &= \sum_{\ell=k}^{p} c_\ell(R) + p(R) = \sum_{\ell=k}^{\alpha k-1} c_\ell(R) + \sum_{\ell=\alpha k}^{p} c_\ell(R) + p(R) \\
&\leq (1+\beta) \sum_{\ell=\alpha k}^{p} c_\ell(R) + p(R) \leq (1+\beta)\frac{\alpha}{\alpha-1} \sum_{\ell \geq k} \frac{\ell - k + 1}{\ell} c_\ell(R) + p(R) \\
&\leq 2(1+\beta)\frac{\alpha}{\alpha-1}\text{OPT}(R) + p(R)
\end{aligned}$$

and the proof is complete, where we have used that $p(R) \leq p$ is a constant. □



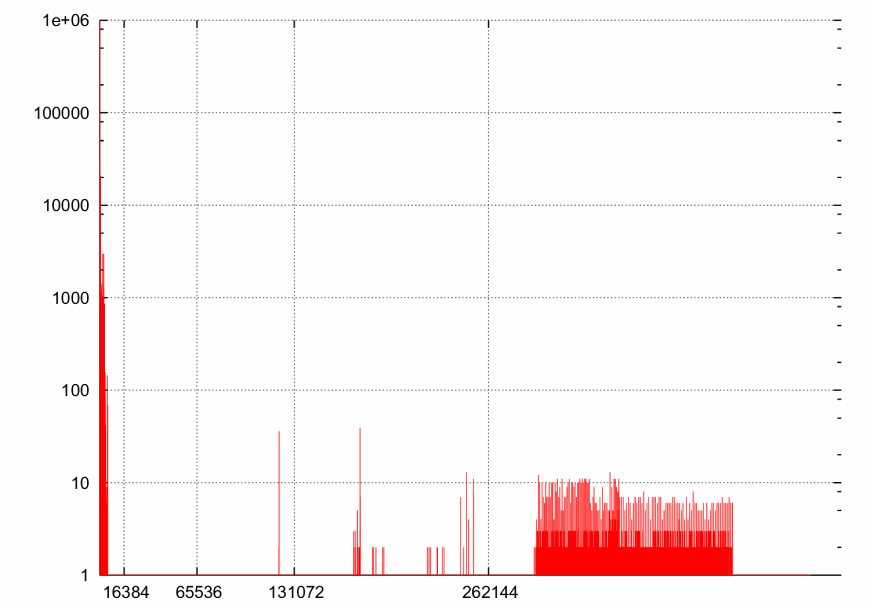
SPEC	Hydro
purpose	fluid dynamics
pages	7 451 717
x-axis	ℓ
y-axis	c_ℓ (log-scale)
k	1024 (4 MB)
(α, β)	$(16.002, 0.012)$
$f(\alpha, \beta)$	2.16

Figure 5.2: SPEC benchmark Hydro

This result can be considered a theoretical underpinning of the good practical behaviour of LRU. It gives a criterion for the sequences on which the algorithm achieves constant competitive ratio. We argue that request sequences that occur in practice satisfy this criterion.

The bound $f(\alpha, \beta)$ is a constant if α is "large" and β is "small". Indeed, the discussion above suggests that practical sequences are in a set $S_{(\alpha,\beta)}$ with this property. To substantiate this claim, we performed experiments with memory traces generated by our implementations of standard algorithms and by the execution of SPEC benchmark programs. SPEC benchmarks are standard for performance evaluation in computer industry. They provide programs for a broad variety of applications, e.g., numerical simulations or user programs. We calculated feasible values for α and β and obtained that our bound $f(\alpha, \beta)$ gives values between two and five for these request sequences, see Figure 5.2 and Figure 5.3.

Further experimental study is needed to determine consistent values of α and β. However, according to the above reasoning, the size of data needed by a program is large compared to the cache size, i.e., at least a constant factor. We hence expect that α is "large" in practice. In our experiments, we also monitored the structure of the characteristic vector of these sequences. Figure 5.2 and Figure 5.3 depict two representative examples of our results. The x-axis shows the length ℓ of the consecutive request and the



SPEC	Nasa7
purpose	weather simulation
pages	31 764 036
x-axis	ℓ
y-axis	c_ℓ (log-scale)
k	1024 (4 MB)
(α, β)	$(3.97, 0.6)$
$f(\alpha, \beta)$	4.28

Figure 5.3: SPEC benchmark Nasa7

y-axis shows the value of the respective c_ℓ in log-scale. There is a peak for smaller values of ℓ and there are many large entries c_ℓ for larger values of ℓ, corresponding to short and long distance. This structure suggests that β should be "small" in practice.

Chapter 6

Bin Packing

The problem Bin Packing is also a rather basic combinatorial optimization problem. It occurs, for example, when certain items have to be packed into as few as possible trucks, where each truck has a weight limitation.

6.1 Deterministic Bin Packing

Formally, the problem is the following. We are given a set of items indexed through $I = \{1, \ldots, n\}$ along with their *weights* $w = (w_1, \ldots, w_n)$, where $0 \leq w_i \leq 1$. We are furthermore given an arbitrary number of *bins* with *capacity* one, each. For any subset $S \subseteq I$ let $\text{weight}(S) = \sum_{i \in S} w_i$ be the total weight of the items in that set. The task is to place the items into as few as possible bins, such that the capacity of no bin is exceeded. More formally, let $B_j \subseteq I$ denote the set of items placed in bin j, then the problem reads as follows:

Problem 6.1 Bin Packing

$$
\begin{array}{lrl}
\text{minimize} & m & \\
\text{subject to} & B_j \subseteq I, & 1 \leq j \leq m, \\
& \cup_{j=1}^{m} B_j = I, & \\
& \text{weight}(B_j) \leq 1, & 1 \leq j \leq m.
\end{array}
$$

The problem is NP-hard. Specifically, the offline version, i.e., the set I and the vector w are given at the outset, admits a polynomial-time approximation algorithm with guarantee $3/2$. But the problems is not approximable within $3/2 - \varepsilon$ for any $\varepsilon > 0$ in polynomial time, unless P = NP. Under the (natural) assumption that the optimal number of bins diverges as the number of items grows, the problem admits a (so-called) asymptotic PTAS, i.e., the problem can be approximated within $1 + \varepsilon$ in polynomial time for any constant $\varepsilon > 0$. However, this is not a practical algorithm, since the constants involved are rather large and since there is an exponential dependency in $1/\varepsilon$.

Online Bin Packing

In this chapter, we shall examine the online version of the problem, i.e., the set I is not known beforehand. Instead, we learn the weights $w_1, w_2, \ldots, w_n$ of the items one-by-one

and have to place item i into some bin before the learn the weight of item $i+1$. For techncal reasons we will assume that $\lim_{n\to\infty} \text{weight}(I) = \infty$.

In worst-case perspective, we measure the quality of an algorithm with its (asymptotic) competitive ratio

$$r(\text{ALG}) = \lim_{n\to\infty} \frac{\text{ALG}(w)}{\text{OPT}(w)}$$

where $\text{ALG}(I)$ and $\text{OPT}(I)$ denote the respective number of bins used by the algorithm ALG and an offline optimal algorithm OPT.

One of the simplest online algorithms for Bin Packing is Next-Fit (NF): Initially $i = 1$ and $j = 1$. Pack items $i, i+1, \ldots, k$ into bin B_j as long as $\sum_{\ell=i}^{k} w_\ell \leq 1$. When violated, set $i = k+1$ and $j = j+1$ and repeat the previous step until there are no items left.

The idea is to keep always one bin *active*, i.e., the one that receives items. Once an item yields that the capacity of the currently active bin is exceeded, the bin is *closed* and another bin is opened, i.e., becomes active. Closed bins will never be opened again. Next-Fit is 2-competitive, which is also best-possible for the algorithm.

Theorem 6.1. *We have that*

$$r(\text{NF}) = 2.$$

Proof. We may assume that NF is using more than one bin, since otherwise there is nothing to show.

Consider the event that NF closes a bin which has remaining capacity p and opens a new bin. The item that is placed into the new bin has weight at least p. Thus the total weight of any two consecutive bins is at least one, i.e., at most half of the total capacity is wasted. Thus, $\text{weight}(B_1) + \text{weight}(B_2) \geq 1, \ldots, \text{weight}(B_{2k-1}) + \text{weight}(B_{2k}) \geq 1$, for $k = \lfloor m/2 \rfloor$. Since all items are placed in the bins $B_1, \ldots, B_m$ we have $\sum_{i\in I} w_i \geq \lfloor m/2 \rfloor \geq (m-1)/2$. Thus,

$$\text{ALG}(w) = m \leq 2 \sum_{i\in I} w_i + 1.$$

Obviously, since we have unit-capacity bins, we have $\text{OPT}(w) \geq \sum_i w_i$, which yields

$$\text{ALG}(w) \leq 2\text{OPT}(w) + 1$$

and the upper bound $r(\text{NF}) \leq 2$ is shown.

For a tight example, let the number of items be $n = 4k$ and consider the weights $w = (1/2, 1/2k, 1/2, 1/2k, \ldots, 1/2, 1/2k)$. The optimal packing uses $k+1$ bins, where k bins are filled with the $1/2$-items, two for each bin, and one bin with the $2k$ items of weight $1/2k$. Obviously, NF places one $1/2$-item and one $1/2k$-item into each bin, thus needing $2k$ bins. □

6.2 Stochastic Bin Packing

In average-case view, we assume that the item weights W_i are independent and identically distributed over $[0, 1]$, which yields an n-element vector $W = (W_1, \ldots, W_n)$ of random variables. We will be interested in the ratio of the expected number of bins as the number of items grows, i.e.,

$$e'(\text{ALG}) = \lim_{n\to\infty} \frac{\mathbb{E}\left[\text{ALG}(W)\right]}{\mathbb{E}\left[\text{OPT}(W)\right]},$$

where ALG(W) and OPT(W) denote the respective random variables for the number of bins for W.

We consider the (stochastic) process of NEXT FIT operating on the random weights $W_1, \ldots, W_n$ given one-by-one. The main result of this section is for uniform distributed weights (but the analysis is more general and can in principle be transfered to other distributions as well).

Theorem 6.2. *If the weights W_i are independent and identically uniform distributed over $[0,1]$, then*

$$e'(\text{NF}) \leq \frac{4}{3}.$$

The process NEXT-FIT induces several other processes:

(1) The sequence $(L_j)_{j\geq 1}$ denotes the total weight, also called *level*, of items packed in B_j when bin j is closed. The complementary sequence $(R_j)_{j\geq 1}$ with $R_j = 1 - L_j$ gives the *remainder* of bin j when closed, respectively.

(2) The sequence $(N_j)_{j\geq 1}$ of the number of items packed into bin j when closed respectively.

(3) For any non-empty bin j, $(U_j)_{j\geq 1}$ denotes the sequence of the sizes of the first item packed into j.

(4) The process $(M_k)_{k\geq 1}$ denotes the sequence of the number of bins when exactly k items are given.

We shall assume throughout that the item-weights W_j are independent and identically distributed over $[0,1]$ according to some random variable X. First we derive some general structural properties of the above processes, and then we will materialize a bound on NF for uniform distributed weights.

General Properties

We investigate the process $(L_j)_{j\geq 1}$ first. Since NF never opens a bin once closed, it follows that the distribution of L_{j+1} depends only on L_j but not on L_k for any $1 \leq k < j$. This means that $(L_j)_{j\geq 1}$ induces a (first-order) Markov chain, i.e., the Markov condition

$$\Pr\left[L_{j+1} \leq t \mid L_j = t_j, \ldots, L_1 = t_1\right] = \Pr\left[L_{j+1} \leq t \mid L_j = t_j\right]$$

is true for all $t, t_j, \ldots, t_1 \geq 0$.

Define $F_{L_j}(t) = \Pr\left[L_j \leq t\right]$ as the distribution of L_j. Denote

$$K(s,t) = \Pr\left[L_{j+1} \leq t \mid L_j = s\right]$$

in the sequel. It is known from Markov chain theory that the distributions for L_{j+1} and L_j satisfy

$$F_{L_{j+1}}(t) = \int_0^1 K(s,t)\, dF_{L_j}(s), \tag{6.1}$$

with the boundary condition

$$F_{L_1}(t) = K(1,t),$$

since L_1 can be seen as a bin following a bin with $L_0 = 1$.

For any k denote by $S_k = \sum_{i=1}^{k} W_i$. If the level at closure of bin j is s, then the first item placed into bin $j+1$ has weight more than $1-s$. In our notation: $L_j = s$ implies $U_{j+1} > 1-s$. With this we find the following expression for $K(s,t)$:

$$K(s,t) = \begin{cases} 0 & \text{if } t \le 1-s, \\ \sum_{k\ge 0} \Pr\left[U_{j+1} + S_k \le t, U_{j+1} + S_k + X > 1 \mid L_j = s\right] & \text{if } t > 1-s. \end{cases}$$

Above, we have used that the distribution of W_{k+1} is identical to X. The distribution of U_{j+1} is also that of X, conditioned on being larger than $1 - L_j$, i.e.,

$$F_{U_{j+1}|L_j}(u|s) = \begin{cases} 0 & \text{if } u \le 1-s \\ \frac{F_X(u) - F_X(1-s)}{1 - F_X(1-s)} & \text{if } 1-s < u \le 1. \end{cases}$$

With these observations, (6.1) is well defined. It is, again, known from Markov chain theory that the Markov chain for $(L_j)_{j\ge 1}$ is ergodic, which implies that it has a unique limit distribution $F_L = \lim_{j\to\infty} F_{L_j}$. Specifically, the following theorem gives a way how to compute it:

Theorem 6.3. *The process $(L_j)_{j\ge 1}$ is an ergodic Markov chain with a limit distribution F_L satisfying*

$$F_L(t) = \int_0^1 K(s,t) dF_L(s).$$

As j increases, F_{L_j} converges (geometrically fast) to F_L and $\mathbb{E}[L_j]$ to $\mathbb{E}[L]$.

Uniform Distributed Weights

Here we assume that the weights W_i are identically uniform distributed over $[0,1]$. Thus, we W_i have the distribution $F_X(t) = t$ for any $0 \le x \le 1$. Further, the distribution of the S_k in the range $0 \le t \le 1$ is known to be $F_{S_k}(t) = t^k/k!$. For these distributions, we can materialize the analysis given above.

Lemma 6.4. *If $F_X(t) = t$ for $0 \le t \le 1$, then $K(s,t)$ defined in (6.1) is given through*

$$K(s,t) = \begin{cases} 0 & \text{if } t \le 1-s \\ 1 - \frac{1}{s}(1-t)e^{-(1-t)+s} & \text{if } 1-s < t \le 1. \end{cases}$$

For the limit distribution F_L it holds

$$F_L(t) = t^3 \quad \text{for } 0 \le t \le 1 \quad \text{and} \quad \mathbb{E}[L] = \frac{3}{4}.$$

Before we prove Lemma 6.4 we show how it implies our main result.

Proof of Theorem 6.2. Define $R = 1 - L$ as the remaining capacity in the limit. Fix any n, let M_n and M_n^* be the respective number of bins used by NF and OPT, respectively, for W. Observe the lower bound

$$M_n^* \ge M_n - \sum_{j=1}^{M_n} R_i = \sum_{j=1}^{M_n} 1 - R_j = \sum_{j=1}^{M_n} L_j.$$

Thus we may write

$$\frac{\mathbb{E}\left[\textsc{nf}(W)\right]}{\mathbb{E}\left[\textsc{opt}(W)\right]} = \frac{\mathbb{E}\left[M_n\right]}{\mathbb{E}\left[M_n^*\right]} \leq \frac{\mathbb{E}\left[M_n - \sum_{j=1}^{M_n} R_j\right] + \mathbb{E}\left[\sum_{j=1}^{M_n} R_j\right]}{\mathbb{E}\left[M_n - \sum_{j=1}^{M_n} R_j\right]} = 1 + \frac{\mathbb{E}\left[\sum_{j=1}^{M_n} R_j\right]}{\mathbb{E}\left[\sum_{j=1}^{M_n} L_j\right]}.$$

As $n \to \infty$, implies $M_n \to \infty$ with probability one, we have

$$\lim_{n\to\infty} \frac{\mathbb{E}\left[\sum_{j=1}^{M_n} R_j\right]}{\mathbb{E}\left[\sum_{j=1}^{M_n} L_j\right]} = \lim_{m\to\infty} \frac{\mathbb{E}\left[\sum_{j=1}^{m} R_j\right]}{\mathbb{E}\left[\sum_{j=1}^{m} L_j\right]} = \frac{\lim_{m\to\infty} \frac{1}{m}\mathbb{E}\left[\sum_{j=1}^{m} R_j\right]}{\lim_{m\to\infty} \frac{1}{m}\mathbb{E}\left[\sum_{j=1}^{m} L_j\right]} = \frac{\mathbb{E}\left[R\right]}{\mathbb{E}\left[L\right]} = \frac{1}{3},$$

which proves the theorem using $\mathbb{E}\left[L\right] = 3/4$ and $\mathbb{E}\left[R\right] = 1/4$ as given in Lemma 6.4. □

Proof of Lemma 6.4. Using $F_X(t) = t$ for $0 \leq t \leq 1$ we first of all find

$$F_{U_{j+1}|L_j}(u|s) = \begin{cases} 0 & \text{if } u \leq 1 - s \\ \frac{1}{s}(s + u - 1) & \text{if } 1 - s < u \leq 1. \end{cases}$$

Let $f_{U|L}$ denote the respective density. This gives

$$\begin{aligned} K(s,t) &= \sum_{k\geq 0} \int_{u=1-s}^{t} f_{U|L}(u|s) \Pr\left[S_k \leq t - u, S_k + X > 1 - u\right] du \\ &= \sum_{k\geq 0} \int_{u=1-s}^{t} \int_{x=0}^{t-u} f_{U|L}(u|s) f_{S_k}(x) \Pr\left[X > 1 - u - x\right] dx du \end{aligned}$$

where we have used independence. Using $\Pr\left[X > x\right] = 1 - x$ and $f_{U|L}(u|s) = 1/s$ yields

$$K(s,t) = \sum_{k\geq 0} \int_{u=1-s}^{t} \int_{x=0}^{t-u} \frac{1}{s} \frac{x^{k-1}}{(k-1)!} (u + x) dx du.$$

The term for $k = 0$ gives the contribution $\int_{u=1-s}^{t} u/s du = (t^2 - (1-s)^2)/2s$. Changing the order of summation (which is allowed as the sums converge uniformly) and integrating yields the claim. □

Chapter 7

Traveling Repairman

Here we consider the following online problem: A server operates on a graph G with non-negative edge-weights and requests to vertices arrive over time (chosen by an adversary), where each request requires servicetime of one unit. Each request must be served, but the server is free to choose their ordering and intends to minimize the total flowtime. The flowtime of a request is the time between its arrival and departure, i.e., the time spent waiting.

First of all, from the worst-case perspective we observe that no deterministic online algorithm can have competitive ratio less than $\mathrm{diam}(G)$, where $\mathrm{diam}(G)$ denotes the diameter of G, i.e., the maximum value of all shortest path lengths between all pairs of vertices. Even worse, but not at all surprising, *no* algorithm of the IGNORE-type is competitive, i.e., has bounded competitive ratio, see Theorem 7.2.

These negative results for the natural IGNORE-type algorithms from the worst-case perspective justify an average-case model. We assume that the requests arrive according to a any stochastic process with rate $\lambda > 0$ (and two further natural assumptions), but an adversary remains free to choose the vertices of the requests. This model captures as special cases deterministic arrivals and Poisson arrivals (that can be observed in many real-world scenarios, e.g., arrival times of phone-calls at a call-center). We give an IGNORE-type algorithm that has expected competitive ratio $c(\lambda)\mathrm{ham}(G)$ if $\lambda \neq 1$, where $c(\lambda)$ is a constant depending on λ, see Theorem 7.3. The value of $c(\lambda)$ can be as small as two but diverges as λ tends to one. Thus, for $\lambda \neq 1$, the algorithm *is* competitive. The intuition is as follows: If $\lambda < 1$, then the requests arrive slower than they are served by our algorithm. If $\lambda > 1$, then they arrive faster than they can be served even by an optimum offline algorithm. For the remaining case $\lambda = 1$, *no* IGNORE-type algorithm is competitive. The reason is that the requests arrive as fast as they can be served by an optimal algorithm, but online algorithms can be forced to make routing-mistakes that accumulate in the flowtimes of many requests.

Model and Notation

Throughout, we consider the following model. A server operates on a connected undirected graph $G = (V, E)$ with n vertices $V = \{1, 2, \ldots, n\}$, edges $e \in E$, and edge-weights $w(e) \geq 0$. There is a distinguished initial vertex, 1 say, where the server is located at time zero. The server moves at unit speed, i.e., the time to pass an edge e is equal to $w(e)$. We define the *distance* w_{ij} between two vertices i, j as the weight of a shortest path connecting i with j. Further, let the *hamiltonicity* $\mathrm{ham}(G)$ be the weight of a shortest, not necessarily simple, circle in G that visits all vertices.

At certain points in time $0 \leq a_1 \leq a_2 \leq \cdots \leq a_m$ requests $R = (r_1, r_2, \ldots, r_m)$ arrive. Each request has the form $r_j = (a_i, v_i)$, i.e., its *arrival* a_j and a vertex v_j. It is demanded that the server eventually visits the vertex v_j to serve the request, which takes unit time. The time d_j when the request is served is called its *departure*. Hence the request has waited for time $f_j = d_j - a_j$, which is called its *flowtime*.

The server is free to choose the ordering of service and is allowed to wait; these decisions induce a *schedule*. For any schedule S, $F(S) = \sum_{j=1}^{m} f_j(S)$ denotes its *total flowtime*, where $f_j(S)$ is the flowtime of request r_j induced by S. Our objective function is to minimize the total flowtime. The schedule that minimizes F given the request sequence R in advance is denoted S^*. For ease of notation we usually write F instead of $F(S)$ and F^* instead of $F(S^*)$. An algorithm ALG is called *online* if its decision at any point in time t depends only on the past, i.e., the requests $r_1, \ldots, r_j$ with $a_j \leq t$. Otherwise the algorithm is called *offline*. We write $\text{ALG}(R) = F$ and $\text{OPT}(R) = F^*$.

We study two different sources for the request sequence R. Firstly, a malicious deterministic adversary is free to choose R. We measure the quality of schedules produced by an online algorithm ALG facing such an adversary with the *competitive ratio*

$$r(\text{ALG}) = \sup_R \frac{\text{ALG}(R)}{\text{OPT}(R)}.$$

If $r(\text{ALG})$ is not bounded, then the algorithm ALG is *not competitive*. Secondly, we study the following stochastic adversary. For a fixed $\lambda > 0$, the adversary is free to choose a probability distribution $D(\lambda)$ on the arrival times of the requests, provided that the expected number of arrivals is λ. Furthermore, the vertices of the requests can also be chosen arbitrarily. We define the *expected competitive ratio* by

$$e_\lambda(\text{ALG}) = \sup_{D(\lambda)} \mathbb{E}\left[\frac{\text{ALG}}{\text{OPT}}\right],$$

where ALG and OPT denote the respective random variables induced by $D(\lambda)$.

We consider a family of algorithms, called IGNORE. Such an algorithm maintains two memories M and M' that are served in *phases* as follows. Suppose that the algorithm has collected several requests in the memory M. As soon as the algorithm starts serving those, all requests that arrive in the meantime are stored in the memory M' (and are hence ignored in the current phase). After all requests in M have been served, the phase ends and the procedure repeats with changed roles of M and M'. Note that we have not yet specified how the requests in a phase are actually served, which leaves space for various strategies.

7.1 Deterministic Arrivals

The following negative result states that no lazy online algorithm for this problem is competitive within less than $\text{diam}(G)$, where $\text{diam}(G) = \max_{ij \in V} w_{ij}$ is the *diameter* of G. An algorithm is called *lazy* if it only changes the vertex it is located if there are pending requests. The situation looks worse for IGNORE algorithms: They are not even competitive, see Theorem 7.2. This result is perhaps not very surprising but justifies that we consider an average-case analysis of this natural type of algorithms.

Observation 7.1. *There is a graph G such that for every lazy online algorithm* ALG *it holds* $r(\text{ALG}) \geq \text{diam}(G)$.

Proof. Consider the graph $G = (\{1,2\}, \{\{1,2\}\})$ with $w(\{1,2\}) = h > 0$, where h is arbitrary. Clearly $\mathrm{diam}(G) = h$.

Initially, the server of the algorithm and the optimum are at vertex 1. Consider just one request at vertex 2 which arrives at time h. The optimal server anticipates the arrival and starts traversing the edge at time zero, reaches vertex 2 at time h and serves the request with flowtime 1. The lazy online algorithm starts traversing the edge at time h and serves the request with flowtime $h+1$. □

Theorem 7.2. *No deterministic* IGNORE*-algorithm is competitive for the online traveling repairman problem with flowtimes.*

Proof. We assume that the IGNORE algorithm starts its first phase immediately when the first request arrives. Furthermore we assume that the algorithm serves the requests in each separate phase optimally. We show below how to extend the lower bound to arbitrary IGNORE algorithms.

Consider the graph $G = (\{1,2\}, \{\{1,2\}\})$ with $w(\{1,2\}) = h > 0$, where h is an arbitrary integer. Let the initial vertex of both servers be 1.

We compose a request sequence R out of *blocks* $B_1, B_2, \ldots, B_k$ of requests as defined below. The idea is as follows: The *delay* of the algorithm at a block B_i is the difference between the respective minimum times the algorithm and the optimum start serving any of the requests in B_i. The essential property of the construction is that the algorithm is forced to traverse the edge more often than the optimum which increases its delay and yields unbounded competitive ratio.

We encode the requests with the format (δ, v), where v denotes the respective vertex and δ the difference between the arrival times of a request and its predecessor (respectively time zero if there is no predecessor).

$$
\begin{aligned}
B_1 &= (h,2)\\
B_2 &= \underbrace{(1,2),\ldots,(1,2)}_{h \text{ requests at } 2},\ \underbrace{(1,1)}_{\text{one trap request at } 1},\ \underbrace{(1,2),\ldots,(1,2)}_{3h+1 \text{ requests at } 2}\\
B_3 &= \underbrace{(h+2,1),(1,1),\ldots,(1,1)}_{5h-1 \text{ requests at } 1},\ \underbrace{(1,2)}_{\text{one trap request at } 2},\ \underbrace{(1,1),\ldots,(1,1)}_{5h+1 \text{ requests at } 1}\\
B_4 &= \underbrace{(h+2,2),(1,2),\ldots,(1,2)}_{9h-1 \text{ requests at } 2},\ \underbrace{(1,1)}_{\text{one trap request at } 1},\ \underbrace{(1,2),\ldots,(1,2)}_{7h+1 \text{ requests at } 2}\\
&\vdots
\end{aligned}
$$

Now we show that the delay increases by $2h$ with every block.

The optimum serves the sequence $R = (B_1, B_2, \ldots, B_k)$ as follows. At time zero its server travels to vertex 2, arrives there at time h, i.e., just in time to serve the first request with flowtime 1 at time $h+1$. Now the server is at the right vertex to serve all the requests at this vertex of the first block, except for the trap-request. This is the last request of the block to be served after traveling to vertex 1. Then the server is at the right spot to handle the requests of the next block. We continue in this manner for every block. All requests, except traps, are served at flowtime 1, each. The trap at block B_i, say, has flowtime $(2i-1)h + h + 1$. This yields total flowtime

$$\mathrm{OPT}(R) \le 1 + \sum_{i=2}^{k}((4(i-2)+1)h + (2i-1)h + 2ih + 2) \le 8hk^2.$$

Now we consider the algorithms. Its server is at vertex 1 until the first request arrives which is served with flowtime $h+1$ at time $2h+1$. Notice that the delay of the algorithm is already h. Until time $2h+1$ the first $h+1$ requests of B_2 have arrived. These include the trap request at vertex 1. As we are dealing with an IGNORE algorithm it will serve the trap during the next phase. As the phases are served optimally the trap is handled last in the phase after one edge-traversal. But now the server is at the wrong vertex to serve the remaining requests of the block and the edge needs to be traversed once more. Thus the algorithm crossed the edge two times more than the optimum server and hence its delay increased by $2h$. We continue in this manner and see that the total delay before block B_i is (at least) $2h(i-1)$. Thus we find the lower bounds on the total flowtime

$$\text{ALG}(R) \geq \sum_{i=1}^{k} 2h(i-1)(2ih) \geq \sum_{i=k/2}^{k} 2h^2 i^2 \geq h^2 k^3/2$$

and the competitive ratio $r(\text{ALG}) \geq kh/16$, which is unbounded.

To adapt the lower bound for general IGNORE algorithms first recall that any such algorithm waits for a determined time (which may depend on the requests seen so far) before starting the first phase. In this case we start by giving the block B_1, i.e., one request at vertex 2 at time h which is served by the optimal server at time $h+1$. Then we wait for the server of the algorithm to move. Right at this time we issue the blocks $B_2, B_3, \ldots, B_k$ similarly as before.

We can also remove the assumption that the IGNORE algorithm serves each phase optimally. If this is not the case we adapt the blocks as follows, where we use that the algorithm is deterministic: Let t_1 be the time the algorithms needs to finish block B_1. Similarly to B_2 we issue requests to vertex 2 and one trap requests to vertex 1 in the time until t_1. If the delay of the algorithm after this phase is less than $2h$ we continue similarly to the remainder of block B_2, otherwise we continue with block B_3 and so forth. □

7.2 Stochastic Arrivals

In this section we consider the following stochastic variant of the problem. As before, each request requires time one for service. An adversary is free to choose the vertices for the arriving requests and has to specify a stochastic process $(X_t)_{t\geq 0}$, where X_t denotes the number of arrivals up to some time t. However, the process has to satisfy the following natural conditions:

(1) For some fixed rate $\lambda > 0$ the expected number of arrivals per unit-time is λ.

(2) For any stopping time T, $\mathbb{E}[X_{T+t} - X_T] \leq \mathbb{E}[X_t]$ holds for all $t \geq 0$. (A random variable T is called *stopping time* for $(X_t)_{t\geq 0}$ if the event $T = t$ depends only on $(X_s)_{s\leq t}$.)

(3) We also require that $\text{Var}[X_t] \leq c\mathbb{E}[X_t]$ for some constant c, i.e., the process has to have bounded variance.

Notice that Poisson-distributed and deterministic arrivals fall into this class as special cases.

We are interested in the expected competitive ratio defined by $e_\lambda(\text{ALG}) = \lim_{t\to\infty} e_\lambda(\text{ALG}, t)$, where $e_\lambda(\text{ALG}, t) = \mathbb{E}[\text{ALG}/\text{OPT}]$ when the process is stopped at some time t and the expectation is taken with respect to X_t. We show that, in this model, bounded expected

competitive ratios are possible for IGNORE algorithms if and only if $\lambda \neq 1$. For the case $\lambda = 1$ the ratio is unbounded for all deterministic IGNORE algorithms.

In specific, we consider the following IGNORE algorithm ALG. At time zero, the algorithm computes a *fixed* circle in the graph G that visits all vertices and has length equal to $\mathrm{ham}(G)$. Let w be a parameter to be specified later on. The algorithm waits for time w and then serves the requests that arrived using the precomputed tour. All requests that arrive in the meantime are ignored and served in the subsequent phase, etc.

Theorem 7.3. *For any stochastic process as defined above, if $\lambda \neq 1$, then there is a constant $c(\lambda)$ such that the algorithm* ALG *yields*

$$e_\lambda(\text{ALG}) \leq c(\lambda) \cdot (\mathrm{ham}(G) + c).$$

There is a process with $\lambda = 1$ such that no IGNORE *algorithm is expected competitive.*

The statement for $\lambda = 1$ of the theorem follows from the proof of Theorem 7.2 by observing that the arrival rate of the requests is one. The statement for $\lambda \neq 1$ follows directly from Lemma 7.5 and Lemma 7.6. The ideas behind these are as follows: For $\lambda < 1$ the requests arrive slower than they are served, i.e., the algorithm is competitive. For $\lambda > 1$ they arrive faster than they can be served, i.e., even the optimum is large.

Now we establish the following technical lemma – the main tool for our analysis – which relates $\mathbb{E}[\text{ALG}/\text{OPT}]$ with $\mathbb{E}[\text{ALG}]/\mathbb{E}[\text{OPT}]$. The intended application is as follows. We partition the probability space into two parts. The first part is a "good event", which, whenever it occurs, implies a bounded competitive ratio. The second part is a "bad event" for which the value of ALG/OPT might be large. However, the hope is that this bad event is unlikely.

Lemma 7.4. *Let X and Y be two positive random variables. Then for every $\delta > 0$ such that $\Pr[Y \geq \delta \mathbb{E}[Y]] > 0$ we have*

$$\mathbb{E}\left[\frac{X}{Y}\right] \leq \frac{\mathbb{E}[X]}{\delta \mathbb{E}[Y]} + \mathbb{E}\left[\frac{X}{Y} \;\middle|\; Y < \delta \mathbb{E}[Y]\right] \Pr[Y < \delta \mathbb{E}[Y]].$$

Proof. First condition on the event $Y \geq \mathbb{E}[Y]\,\delta$ and then use $\mathbb{E}[X \mid A] \leq \mathbb{E}[X]/\Pr[A]$ which holds for any non-negative random variable and any event A with $\Pr[A] > 0$. We deduce

$$\begin{aligned}
\mathbb{E}\left[\frac{X}{Y}\right] &= \mathbb{E}\left[\frac{X}{Y} \;\middle|\; Y \geq \delta \mathbb{E}[Y]\right] \Pr[Y \geq \delta \mathbb{E}[Y]] \\
&\quad + \mathbb{E}\left[\frac{X}{Y} \;\middle|\; Y < \delta \mathbb{E}[Y]\right] \Pr[Y < \delta \mathbb{E}[Y]] \\
&\leq \frac{\mathbb{E}[X \mid Y \geq \delta \mathbb{E}[Y]] \Pr[Y \geq \delta \mathbb{E}[Y]]}{\delta \mathbb{E}[Y]} \\
&\quad + \mathbb{E}\left[\frac{X}{Y} \;\middle|\; Y < \delta \mathbb{E}[Y]\right] \Pr[Y < \delta \mathbb{E}[Y]] \\
&\leq \frac{\mathbb{E}[X]}{\delta \mathbb{E}[Y]} + \mathbb{E}\left[\frac{X}{Y} \;\middle|\; Y < \delta \mathbb{E}[Y]\right] \Pr[Y < \delta \mathbb{E}[Y]].
\end{aligned}$$

The next to last inequality holds since for every event A with $\Pr[A] > 0$ the conditional expectation can be bounded in the way $\mathbb{E}[X \mid A] = (\mathbb{E}[X] - \mathbb{E}[X \mid \overline{A}] \Pr[\overline{A}])/\Pr[A] \leq \mathbb{E}[X]/\Pr[A]$, where we have used that X is non-negative. □

Below we use the following additional notation. A request r_j is called *pending* while it has arrived but not departed. By definition, the algorithm initially waits until time $T_0 = w$. Let the number of requests that arrive during the time interval $I_0 = [0, T_0)$ be A_0. The time when the algorithm has served these requests and is back at the initial vertex 1 be T_1. Further let the number of arriving requests in the time interval $I_1 = [T_0, T_1)$ be A_1. This induces a sequence of intervals $I_0, I_1, I_2, \ldots$, called *phases*, *stopping times* $T_0, T_1, T_2, \ldots$ and numbers $A_0, A_1, A_2, \ldots$, called *arrivals*. F_j denotes the (random) flowtime of a request r_j and $I(j)$ the phase in which r_j arrives. The optimum flowtime of request r_j is denoted F_j^*.

Lemma 7.5. *For $\lambda < 1$ we have $e_\lambda(\text{ALG}) \leq (4/(1-\lambda) + c/2) \cdot (\text{ham}(G) + 1)$.*

Proof. We begin by choosing a value for the parameter w, i.e., the time the algorithm waits before starting its first phase. For any positive value of w, we expect that λw requests arrive while waiting. Hence we expect that the first phase takes time $\text{ham}(G) + \lambda w$. We require that this time be no more than w, i.e., we need $\text{ham}(G) + \lambda w \leq w$. Hence we choose $w = \text{ham}(G)/(1-\lambda)$ which is possible for $\lambda < 1$.

Let t be any fixed time and let X_t be the (random) number of requests that arrive until time t. Now we establish the bounds $F^* \geq X_t$ and $\mathbb{E}[F] \leq \mathbb{E}[X_t]\, 2(\text{ham}(G)+1)/(1-\lambda)$.

The optimum flowtime for any request r_j is $F_j^* \geq 1$ and for F_j we have $F_j \leq \text{ham}(G) + A_{I(j)-1} + \text{ham}(G) + A_{I(j)} + 1$.

We prove by induction that $\mathbb{E}[A_i] \leq \lambda w$ for all $i \geq 0$. This implies $\mathbb{E}[F_j] \leq 2\text{ham}(G) + (2\lambda w + 1) \leq 2\text{ham}(G)/(1-\lambda) + 1$, by choice of w. The base case $\mathbb{E}[A_0] = \lambda w$ is clear. For the inductive case we exploit the following assumed property of our process: For any stopping time T, $\mathbb{E}[X_{T+t} - X_T] \leq \mathbb{E}[X_t]$ holds for all $t \geq 0$. By induction hypothesis $\mathbb{E}[A_i] \leq \lambda w$ holds for a fixed i and it follows that $\mathbb{E}[A_{i+1}] = \lambda \mathbb{E}[T_{i+1} - T_i] \leq \lambda \mathbb{E}[\text{ham}(G) + A_i] \leq \lambda(\text{ham}(G) + \lambda w) \leq \lambda w$. The crude estimates $F \leq X_t^2(\text{ham}(G)+1)$ and $F^* \geq X_t$ give

$$\mathbb{E}\left[\frac{F}{F^*} \;\middle|\; X_t < \frac{1}{2}\mathbb{E}[X_t]\right] \leq \frac{\mathbb{E}[X_t]\,(\text{ham}(G)+1)}{2}.$$

With Lemma 7.4, Chebyshev's inequality, and the choice $\delta = 1/2$ we have

$$\begin{aligned} e_\lambda(\text{ALG}, t) &\leq \mathbb{E}\left[\frac{F}{X_t} \;\middle|\; X_t \geq \frac{1}{2}\mathbb{E}[X_t]\right] \\ &\quad + \mathbb{E}\left[\frac{F}{F^*} \;\middle|\; X_t < \frac{1}{2}\mathbb{E}[X_t]\right] \Pr\left[X_t < \frac{1}{2}\mathbb{E}[X_t]\right] \\ &\leq \frac{2\mathbb{E}[F]}{\mathbb{E}[X_t]} + \frac{\mathbb{E}[X_t]\,(\text{ham}(G)+1)}{2} \Pr\left[|X_t - \mathbb{E}[X_t]| \geq \frac{1}{2}\mathbb{E}[X_t]\right] \\ &\leq \frac{4(\text{ham}(G)+1)}{1-\lambda} + \frac{\mathbb{E}[X_t]\,(\text{ham}(G)+1)}{2} \cdot \frac{\text{Var}[X_t]}{\mathbb{E}[X_t]^2}, \end{aligned}$$

which yields the claim using $\text{Var}[X_t] \leq c\mathbb{E}[X_t]$, as assumed. □

Lemma 7.6. *Let $0 < \delta < 1$ be an any constant such that $(1-\delta)\lambda > 1$. Then we have*

$$E^{\text{ALG}}(\lambda) \leq \left(\frac{2}{\delta^2} + \frac{c\lambda^2}{(\lambda - \frac{1}{1-\delta})^2}\right) \cdot (\text{ham}(G) + 1).$$

Proof. Here we choose $w = 0$, i.e., the algorithms begins its first phase immediately after the arrival of the first request. We stop the process at any fixed time t and let X_t be the number of requests that arrived.

As upper bound we use the crude estimate $F \leq X_t^2(\mathrm{ham}(G)+1)$. Now we establish the lower bound $F^* \geq X_t^2\delta^2/2$ which holds conditional on the event that $X_t(1-\delta) \geq t$. At time t the optimum can have served at most t many requests. Thus, by the condition on X_t, there are $P \geq X_t - t \geq \delta X_t$ requests pending. As each request takes time at least one we have $F^* \geq \sum_{i=1}^{P} i \geq \delta^2 X_t^2/2$. Thus, using Chebyshev's inequality, we find $\Pr[X_t(1-\delta) < t] \leq \Pr[|X_t - \mathbb{E}[X_t]| > t(\lambda - 1/(1-\delta))] \leq c\lambda/t(\lambda - 1/(1-\delta))^2$ and hence

$$\begin{aligned}
E_t^{\mathrm{ALG}}(\lambda) &\leq \mathbb{E}\left[\frac{2X_t^2(\mathrm{ham}(G)+1)}{\delta^2 X_t^2} \;\middle|\; X_t(1-\delta) \geq t\right] \\
&\qquad + \mathbb{E}\left[\frac{X_t^2(\mathrm{ham}(G)+1)}{X_t} \;\middle|\; X_t(1-\delta) < t\right] \Pr[X_t(1-\delta) < t] \\
&\leq \frac{2(\mathrm{ham}(G)+1)}{\delta^2} + \frac{(\mathrm{ham}(G)+1)t}{(1-\delta)} \cdot \frac{c\lambda t}{t^2(\lambda - \frac{1}{1-\delta})^2} \\
&\leq \frac{2(\mathrm{ham}(G)+1)}{\delta^2} + \frac{c(\mathrm{ham}(G)+1)\lambda^2}{(\lambda - \frac{1}{1-\delta})^2}
\end{aligned}$$

which yields the claim using $\lambda(1-\delta) > 1$. □