Randomized- and Online Algorithms

Lecture Notes, Spring Term 2018

University of Zurich

P.D. Dr. Alexander Souza

Contents

Part I

Randomized Algorithms

Chapter 1

Introduction

Generally speaking, *randomized algorithms* have access to (arbitrarily many) random bits, and are allowed to decide based on their outcomes. This, of course, yields that either the output and/or the running time of the algorithm are random variables. As a consequence we can not expect that the algorithm behaves exactly the same, when given the same input. (Notice that deterministic algorithms do have this property.) Why would one want to allow such indefiniteness? There are several reasons: For example, approximation algorithms are sometimes "fooled" by rather artifical counterexamples that rely on the specific strategy of the algorithm. In that perspective, randomizing strategies often yield improvements in approximation guarantee (in expectation). Furthermore, random choices sometimes yield significant speed-up of running time (in expectation), compared to worst-case running times. We will see examples hereof shortly.

If it is not desired to have a randomized algorithm, respectively a randomized construction method, then one can also try to *derandomize* a randomized algorithm. As the name suggests, this refers to the process of turning a randomized algorithm into a deterministic one. This is often achieved at the price of higher running times and/or deterioration of solution quality.

1.1 Maximum Cut

We begin with an example. The problem Maximum Cut is one of the classical NP-hard problems and defined as follows: Let $G = (V, E)$ be an undirected graph with n vertices and m edges. A *cut* is a partition of the vertices into a pair (L, R) with $L \subseteq V$ and $R = V - L$. A *cut-edge* is an edge ℓr with $\ell \in L$ and $r \in R$. The problem is to find a cut which maximizes the number of cut-edges among all cuts.

Consider the following easy algorithm Simple-Max-Cut: Initially $R = L = \emptyset$. For each vertex $v \in V$, assign v to L with probability $1/2$, and to R otherwise. Return (L, R).

The charm of this procedure is

(i) its simplicity,

(ii) its speed, and

(iii) nontheless achieves a certain solution-quality, see below.

Theorem 1.1. *Let X be the number of cut-edges X returned by* Simple-Max-Cut. *We have that*

(i) $\mathbb{E}[X] = m/2$,

(ii) $\mathrm{Var}[X] = m/4$, *and*

(iii) for any constant $c > 0$

$$\Pr[|X - \mathbb{E}[X]| \geq c \cdot m] \leq \frac{1}{4c^2 m},$$

Proof. For the proof of (i) we compute the expected value of X. For any edge $e \in E$ define an indicator variable

$$X_e = \begin{cases} 1 & \text{if } e \text{ is a cut-edge,} \\ 0 & \text{otherwise.} \end{cases}$$

Hence $X = \sum_{e \in E} X_e$ counts the number of cut-edges. We have

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{e \in E} X_e\right] = \sum_{e \in E} \mathbb{E}[X_e] = \sum_{e \in E} \Pr[X_e = 1],$$

where we have crucially used linearity of expectation.

An edge $e = vw \in E$ is a cut-edge in the cut (L, R) constructed by Simple-Max-Cut if its end-vertices v and w are in distinct sets. That is, v is assigned to L and w to R (or vice versa). The respective probability is $1/4$ and the total probability that e is a cut-edge is hence $1/2$, i.e., $\Pr[X_e = 1] = 1/2$.

For (ii), we compute the variance $\mathrm{Var}[X] = \mathbb{E}[(X - \mathbb{E}[X])] = \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2$. Since we already know $\mathbb{E}[X]$, we are now interested only in $\mathbb{E}\left[X^2\right]$ and find

$$\begin{aligned} \mathbb{E}\left[X^2\right] &= \mathbb{E}\left[\left(\sum_{e \in E} X_e\right)^2\right] \\ &= \mathbb{E}\left[\sum_{e \neq e' \in E} X_e \cdot X_{e'} + \sum_{e \in E} X_e^2\right] \\ &= \sum_{e \neq e' \in E} \mathbb{E}[X_e \cdot X_{e'}] + \mathbb{E}\left[\sum_{e \in E} X_e\right], \end{aligned}$$

where we have used $X_e^2 = X_e$ since X_e is an indicator variable.

For computing $\mathbb{E}[X_e \cdot X_{e'}]$ we distinguish two cases: If $e \cap e' = \emptyset$, then X_e and $X_{e'}$ are independent and we have $\mathbb{E}[X_e \cdot X_{e'}] = \mathbb{E}[X_e] \cdot \mathbb{E}[X_{e'}] = 1/4$. Otherwise $e \cap e' \neq \emptyset$ and the edges share exactly one endpoint. We have $X_e \cdot X_{e'} = 1$ if and only if the non-shared endpoints are in the same set (both in L or both in R) and the shared one is in the respective other set. We hence have

$$\mathbb{E}[X_e \cdot X_{e'}] = \Pr[X_e = 1, X_{e'} = 1] = 2 \cdot \frac{1}{8} = \frac{1}{4}.$$

We continue

$$\begin{aligned} \mathbb{E}\left[X^2\right] &= \sum_{e \neq e' \in E} \mathbb{E}[X_e \cdot X_{e'}] + \mathbb{E}[X] \\ &= \frac{1}{4} \cdot m(m-1) + \frac{m}{2}. \end{aligned}$$

We put the things together and find

$$\mathrm{Var}[X] = \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2 = \frac{1}{4} \cdot m(m-1) + \frac{m}{2} - \frac{m^2}{4} = \frac{m}{4}.$$

For (iii), application of Chebyshev's inequality $\Pr[|X - \mathbb{E}[X]| \geq t] \leq \mathrm{Var}[X]/t^2$ yields

$$\Pr[|X - \mathbb{E}[X]| \geq c \cdot m] \leq \frac{\mathrm{Var}[X]}{c^2 m^2} = \frac{1}{4c^2 m},$$

as claimed. □

Let $\textsc{opt}(G)$ denote the maximim number of cut-edges in G. We obviously have $\textsc{opt}(G) \leq m$ since the cut with maximum number of cut-edges can not have more than m cut-edges.

Corollary 1.2. *We have that*

$$\mathbb{E}[X] \geq \frac{\textsc{opt}(G)}{2} \quad \textit{and} \quad \Pr[X < \textsc{opt}(G)/2 - c \cdot m] \leq \frac{1}{4c^2 m}.$$

Proof. Firstly, $\mathbb{E}[X] = m/2 \geq \textsc{opt}(G)/2$ by $\textsc{opt}(G) \leq m$. Secondly, we have

$$\begin{aligned}
\Pr[X < \textsc{opt}(G)/2 - c \cdot m] &\leq \Pr[X < m/2 - c \cdot m] \\
&= \Pr[X - \mathbb{E}[X] < -c \cdot m] \\
&\leq \Pr[|X - \mathbb{E}[X]| \geq c \cdot m] \\
&\leq \frac{1}{4c^2 m}
\end{aligned}$$

which was claimed. □

1.2 Combinatorial Optimization

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

U	the *solution space* of possible outputs,
P	the *feasibility predicate*,
val	the *value function* $\mathrm{val} : U \to \mathbb{R}$,
extr	the desired *extremum*, i.e., max or min.

extr and val together define the *objective function*. The feasibility predicate P induces a set:

$$S \quad \text{the set of } \textit{feasible solutions}\text{: } S = \{X \in U : X \text{ satisfies } P\}.$$

Our goal is to find a feasible solution where the desired extremum of val is attained. Any such solution is called an *optimum solution*, or simply an *optimum*. U and S are usually not given explicitly, but implicitly.

Example 1.3. Let us investigate the problem Maximum Cut of Section 1.1 in this framework.

The solution space is the set of pairs of vertex-sets, i.e., $\{(L, R) : L \subseteq V, R \subseteq V\}$. The feasibility predicate is "(L, R) defines a cut in G", i.e., (L, R) defines a partition of the vertex-set V of G. The value function is the number of cut-edges, i.e., the number of edges that have one end-vertex in L and the other in R. Finally, the desired extremum is "max", i.e., we seek to maximize the value function.

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*.

Many combinatorial optimization problems can be solved by using an appropriate algorithm. Informally, an *algorithm* is given a (valid) input, i.e., a description of an instance of a problem and computes a solution after a finite number of "elementary steps". The number of bits used to describe an input I is called the (binary) *length* or *size* of the input and denoted $\text{size}(I)$.

Let $t : \mathbb{N} \to \mathbb{R}$ be a function. We say that an algorithm *runs* in time $O(t)$ if there is a constant α such that the algorithm uses at most $\alpha t(\text{size}(I))$ many elementary steps to compute a solution given any input I. An algorithm is called *polynomoial time* if $t : n \mapsto n^c$ for some constant c. This contrasts *exponential time* algorithms where $t : n \mapsto c^n$ for some constant $c > 1$.

Because the running times of exponential time algorithms grow rather rapidly as the input size grows, we are mostly interested in polynomial time algorithms. Of course, we desire to find an optimum solution for any given COP in polynomial time. Unfortunately this is not always possible as many COPs are NP-hard. (It is widely believed that no polynomial time algorithm exists that solves some NP-hard COP optimally on every instance.) Thus our goal is a find "good" solutions in polynomial time.

Let $\Pi = \{I_1, I_2, \dots\}$ be a set of instances of a COP, where each $I \in \Pi$ is of the form $I = (U, P, S, \text{val}, \text{extr})$. For any $I \in \Pi$, let $\text{OPT}(I) = \text{extr}_{X \in S(I)} \text{val}(X)$ denote the respective optimum value. An *approximation algorithm* ALG for Π is a polynomial time algorithm that computes some solution $X \in S(I)$ for every instance $I \in \Pi$. The respective value obtained is denoted $\text{ALG}(I) = \text{val}(X)$. The *approximation ratio* of ALG on an instance I is defined by

$$\rho_{\text{ALG}}(I) = \frac{\text{ALG}(I)}{\text{OPT}(I)}.$$

The algorithm ALG is a *ρ-approximation* algorithm if

$$\begin{aligned} \rho_{\text{ALG}}(I) &\leq \rho \qquad \text{for all } I \in \Pi \text{ and extr} = \min, \\ \rho_{\text{ALG}}(I) &\geq \rho \qquad \text{for all } I \in \Pi \text{ and extr} = \max. \end{aligned}$$

1.3 Randomized Algorithms

Generally speaking, *randomized algorithms* have access to (arbitrarily many) random bits, and are allowed to decide based on their outcomes. This, of course, yields that either the output and/or the running time of the algorithm are random variables. As a consequence we can not expect that the algorithm behaves exactly the same, when given the same input. (Notice that deterministic algorithms do have this property.) Why would one want to allow such indefiniteness? There are several reasons: For example, approximation algorithms are sometimes "fooled" by rather artifical counterexamples that rely on the specific strategy of the algorithm. In that perspective, randomizing strategies often yield improvements in approximation guarantee (in expectation). Furthermore, random choices sometimes yield significant speed-up of running time (in expectation), compared to worst-case running times.

There are two types of randomized algorithms: *Las Vegas* algorithms are allowed to use the random bits, but must always return correct answers. In contrast, *Monte Carlo*

algorithms are allowed to return false answers, but this must not happen with "too large" probability.

If it is not desired to have a randomized algorithm, respectively a randomized construction method, then one can also try to *derandomize* a randomized algorithm. As the name suggests, this refers to the process of turning a randomized algorithm into a deterministic one. This is often achieved at the price of higher running times and/or deterioration of solution quality.

1.3.1 Model of Computation

A *computation tree* T is a directed tree whose vertices represent computational states and whose edges represent transitions between those states. We can think of a vertex v and its (finitely many) outgoing edges $E(v)$ as a set of alternatives that an algorithm is allowed to select among. A *leaf* ℓ in the tree T, i.e., a vertex without alternatives, represents termination of the algorithm. Every finite path from the root to a leaf of T corresponds to a computation, called a *run*.

The above notions give rise to a *randomized algorithm* A, which, at any vertex in a computation tree, chooses one of its alternatives with a certain probability, independently of each other. Specifically, given such a tree T, let v be a vertex with outgoing edges $E(v)$ therein. Further, a probability distribution on the edges $e \in E(v)$ with the meaning

$$\Pr[A \text{ chooses } e \in E(v) \text{ at vertex } v] = p_e$$

for each $e \in E(v)$ defines R. Our assumption that every run is finite yields that every vertex v of T is reached with a certain probability p_v. More precisely, if vertex v is reachable from the root r of T along exactly the edges $e_1, \ldots, e_k$, then

$$\Pr[A \text{ reaches } v] = p_v = p_{e_1} \cdot p_{e_2} \cdot \cdots \cdot p_{e_k}.$$

Let $\text{length}_T(v)$ be the number of vertices on the path from r to v.

Since each leaf ℓ (i.e., output) is reached with some probability p_ℓ by A, we can identify the randomized algorithm with a probability distribution over the leafs (outputs, runs, computations).

Example 1.4. Recall the Maximum Cut problem and let the graph $G = (\{v_1, v_2\}, \{v_1v_2\})$ be simply an edge connecting two vertices v_1 and v_2. There are hence four cuts that qualify as outputs of Simple-Max-Cut, namely $(\{v_1, v_2\}, \{\}), (\{v_1\}, \{v_2\}), (\{v_2\}, \{v_1\})$, and $(\{\}, \{v_1, v_2\})$. Implicitly from its definition, the algorithm Simple-Max-Cut assigns probability 1/4 to each of those cuts/outputs.

There are two notions of running time that can be defined for randomized algorithms. The first one is in purely worst-case perspective, the second one makes use of the underlying probability distributions. Firstly, let $\text{size}(I)$ denote the encoding length of input I (which depends on the application). Secondly, for any vertex v in the computation tree T of A on input I, let

$$\text{rt}_A(v) = \text{length}_T(v)$$

be the *running time* of A for v.

Below, $\ell \in T$ denotes a leaf in T is the computation tree induced by A on input I.

The *worst-case running time* of a randomized algorithm A on input I is

$$\text{rt}_A(I) = \max\{\text{rt}_A(\ell) : \ell \in T\}.$$

The *worst-case running time* of A on inputs of size at most n is

$$\mathrm{rt}_A(n) = \max\{\mathrm{rt}_A(I) : \mathrm{size}(I) \leq n\}.$$

The *expected running time* of A on input I is

$$\mathrm{ert}_A(I) = \sum_{\ell \in T} \mathrm{rt}_A(\ell) \cdot p_\ell.$$

The *expected running time* of A on inputs of size at most n is

$$\mathrm{ert}_A(n) = \max\{\mathrm{ert}_A(I) : \mathrm{size}(I) \leq n\}.$$

Having defined the running times of randomized algorithms, we will classify them with respect to the "degree of correctness" of their outputs.

1.3.2 Las Vegas Algorithms

Suppose that we want an algorithm that computes a certain function $f : I \to S$, mapping an *instance* (input) I to a *solution* $S = f(I)$. Let $A(I)$ denote the output of an algorithm A on input I.

There are two variants of Las Vegas algorithms: The first type of Las Vegas algorithms is allowed to output a symbol "?" with the meaning "I don't know". However, if an output different from "?" is returned, then it must be a solution. The second type has to compute a solution for any given input.

Let $0 < \varepsilon \leq 1$ be constant. A randomized algorithm A is called *ε-ordinary Las Vegas*, if for every input I of f we have

(i) $\Pr[A(I) = f(I)] \geq \varepsilon$, and

(ii) $\Pr[A(I) = \text{"?"}] = 1 - \Pr[A(I) = f(I)] \leq 1 - \varepsilon$.

Here the output "?" means that A did not find a solution of f on input I. On the other hand, the *successful* event "$A(I) = f(I)$" must have sufficiently high probability, say ε.

As a special case, a 1-ordinary Las Vegas algorithm is called *strict*. If $0 < \varepsilon < 1$ we can arbitarily increase the success probability by repeating the algorithm sufficiently many times. This is the statement of the result below, called *probability amplification*. Suppose we run the algorithm A for k many times. The induced algorithm A_k shall return "?" if all k runs of A yield "?". Otherwise A_k returns the first different output.

Theorem 1.5. *If there is an ε-ordinary Las Vegas algorithm A, then there is also an ε'-ordinary Las Vegas algorithm A'.*

Proof. Consider the algorithm $A' = A_k$ for a suitable k defined below. The probability that "?" is output in each of the k runs of A is at most $(1-\varepsilon)^k < 1$. Thus the probability for the event "$A_k(I) = f(I)$" is at least $1-(1-\varepsilon)^k$. We resolve the inequality $1-(1-\varepsilon)^k \geq \varepsilon'$ for k and achieve that $k \geq \lceil \ln(1-\varepsilon')/\ln(1-\varepsilon) \rceil$ yields the claim. □

1.3.3 Monte Carlo Algorithms

For Monte Carlo algorithms, we want to allow that the algorithm also returns false answers. However, we will require that the probability that the algorithm produces a correct result is more than $1/2$.

Let $0 < \varepsilon \leq 1/2$. A randomized algoritm A is called *ε-ordinary Monte Carlo algorithm*, if for every input I of f we have

$$\Pr[A(I) = f(I)] \geq \frac{1}{2} + \varepsilon.$$

It important to note that ε need *not* be a constant in this definition. Instead ε is allowed to decrease with size(I). Similarly to Las Vegas algorithms, we can still amplificate the success-probability. Define A_k by running A independently k many times and let the outputs be $O_1, \ldots, O_k$. If there exists an output O which occurs at least $\lceil k/2 \rceil$ many times in $O_1, \ldots, O_k$ then return O. Otherwise return "?".

Theorem 1.6. *If there is an ε-ordinary Monte Carlo algorithm A, then there is also an ε'-ordinary Monte Carlo algorithm A'.*

Proof. For an input I let $p = p(I) = \Pr[A(I) \in f(I)] = 1/2 + \varepsilon_I$ for an $\varepsilon_I \geq \varepsilon$ be a shorthand for the success-probability of A on input I. The algorithm $A' = A_k$ computes a wrong result or "?" if the correct result was computed less than $\lceil k/2 \rceil$ many times in $O_1, \ldots, O_k$. We now prove that this is "relatively unlikely". Let $p_{i,k}(I)$ be the probability that A_k computes the correct answer exactly i times (in its k independent repetitions). For $i < \lceil k/2 \rceil$ we have

$$\begin{aligned}
p_{i,k}(I) &= \binom{k}{i} p^i (1-p)^{k-i} = \binom{k}{i} (p^i (1-p)^i)(1-p)^{k-2i} \\
&= \binom{k}{i} \left(\left(\frac{1}{2} + \varepsilon_I \right)^i \left(\frac{1}{2} - \varepsilon_I \right)^i \right) \left(\frac{1}{2} - \varepsilon_I \right)^{2(k/2-i)} \\
&= \binom{k}{i} \left(\frac{1}{4} - \varepsilon_I^2 \right)^i \left(\left(\frac{1}{2} - \varepsilon_I \right)^2 \right)^{k/2-i} \\
&< \binom{k}{i} \left(\frac{1}{4} - \varepsilon_I^2 \right)^i \left(\left(\frac{1}{2} - \varepsilon_I \right) \left(\frac{1}{2} + \varepsilon_I \right) \right)^{k/2-i} \\
&= \binom{k}{i} \left(\frac{1}{4} - \varepsilon_I^2 \right)^i \left(\frac{1}{4} - \varepsilon_I^2 \right)^{k/2-i} \\
&= \binom{k}{i} \left(\frac{1}{4} - \varepsilon_I^2 \right)^{k/2}
\end{aligned}$$

The algorithm A_k returns the correct result if it was computed at least $\lceil k/2 \rceil$ many times.

Thus

$$\begin{aligned}
\Pr\left[A_k(I) = f(I)\right] &= 1 - \sum_{i=0}^{\lceil k/2\rceil - 1} p_{i,k}(I) \\
&> 1 - \sum_{i=0}^{\lceil k/2\rceil - 1} \binom{k}{i} \left(\frac{1}{4} - \varepsilon_I^2\right)^{k/2} \\
&= 1 - \left(\frac{1}{4} - \varepsilon_I^2\right)^{k/2} \sum_{i=0}^{\lceil k/2\rceil - 1} \binom{k}{i} \\
&> 1 - \left(\frac{1}{4} - \varepsilon_I^2\right)^{k/2} 2^k \\
&= 1 - \left(1 - 4\varepsilon_I^2\right)^{k/2} \\
&\geq 1 - \left(1 - 4\varepsilon^2\right)^{k/2}.
\end{aligned}$$

The error probability $(1 - 4\varepsilon^2)^{k/2}$ decreases as k increases (since $\varepsilon > 0$). We seek a value for k such that

$$\Pr\left[A_k(I) = f(I)\right] \geq \varepsilon'.$$

The choice

$$k = \left\lceil \frac{2\ln(1-\varepsilon')}{\ln(1-4\varepsilon^2)} \right\rceil$$

is sufficient. □

The motivation behind the class of Monte Carlo algorithms with *bounded error* is to ensure that a constant number of independent repetitions always suffices for reducing the error-probability below a certain constant δ. The principal difference between ordinary Monte Carlo algorithms and those with bounded error is, that for the latter, there exists a universal fixed distance ε from $1/2$ in error-probability. For the former, as we have seen, it may happen that the distance between the error probability and $1/2$ decreases as the input size increases.

A randomized algoritm A is a *Monte Carlo algorithm with bounded error* if there exists a *constant* $0 < \varepsilon \leq 1/2$ such that, for every input I of f we have

$$\Pr\left[A(I) = f(I)\right] \geq \frac{1}{2} + \varepsilon.$$

In this definition, the choice $1/2 + \varepsilon$ for a constant ε is actually important for ensuring that a constant number of repetitions suffices for reducing the error-probability below any given constant δ.

Corollary 1.7. *If ε and δ are constant, then $\Pr\left[A_k(I) \in f(I)\right] \geq 1 - \delta$ for a constant*

$$k \geq \left\lceil \frac{2\ln\delta}{\ln(1-4\varepsilon^2)} \right\rceil.$$

Chapter 2

Linearity of Expectation

Linearity of expectation basically says that the expected value of a sum of random variables is equal to the sum of the individual expectations. Its importance can hardly be overestimated for the area of randomized algorithms and probabilistic methods. Its main power lies in the facts that it

(i) is applicable for sums of *any* random variables (independent or not), and

(ii) that it often allows simple "local" arguments instead of "global" ones.

2.1 Basics

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

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

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

Example 2.1. Let X_1 and X_2 denote two independent rolls of a fair dice. What is the expected value of the sum $X = X_1 + X_2$? We use the definition, calculate and obtain

$$\mathbb{E}[X] = 2 \cdot \frac{1}{36} + 3 \cdot \frac{2}{36} + \cdots + 12 \cdot \frac{1}{36} = 7.$$

As stated already, linearity of expectation allows us to compute the expected value of a sum of random variables by computing the sum of the individual expectations.

Theorem 2.2. *Let $X_1, \ldots, X_n$ be any finite collection of discrete random variables and let $X = \sum_{i=1}^{n} X_i$. Then we have*

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i].$$

Proof. We use the definition, reorder the sum by its finiteness, and obtain

$$\begin{aligned}\mathbb{E}[X] &= \sum_{\omega\in\Omega} X(\omega)\Pr[\omega] \\ &= \sum_{\omega\in\Omega} (X_1(\omega) + \cdots + X_n(\omega))\Pr[\omega] \\ &= \sum_{i=1}^{n}\sum_{\omega\in\Omega} X_i(\omega)\Pr[\omega] \\ &= \sum_{i=1}^{n} \mathbb{E}[X_i],\end{aligned}$$

which was claimed. □

It can be shown that linearity of expectation also holds for countably infinite summations in certain cases. For example, it holds that

$$\mathbb{E}\left[\sum_{i=1}^{\infty} X_i\right] = \sum_{i=1}^{\infty} \mathbb{E}[X_i]$$

if $\sum_{i=1}^{\infty} \mathbb{E}[|X_i|]$ converges.

Example 2.3. Recalling Example 2.1, we first compute $\mathbb{E}[X_1] = \mathbb{E}[X_2] = 1 \cdot 1/6 + \cdots + 6 \cdot 1/6 = 7/2$ and hence

$$\mathbb{E}[X] = \mathbb{E}[X_1] + \mathbb{E}[X_2] = \frac{7}{2} + \frac{7}{2} = 7.$$

Admittedly, in such a trivial example, the power of linearity of expectation can hardly be seen. This should, however, change in the applications to come.

2.2 Applications

2.2.1 Balls Into Bins

Many problems in computer science and combinatorics can be formulated in terms of a Balls into Bins process. We give some examples here. Suppose we have m balls, labeled $i = 1, \ldots, m$ and n bins, labeled $j = 1, \ldots, n$. Each ball is thrown into one of the bin independently and uniformly at random.

Theorem 2.4. *Let X_j denote the number of balls in bin j. Then, for $j = 1, \ldots, m$ we have*

$$\mathbb{E}[X_j] = \frac{m}{n}$$

Proof. Define an indicator variable

$$X_{i,j} = \begin{cases} 1 & \text{ball } i \text{ falls into bin } j, \\ 0 & \text{otherwise,} \end{cases}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. We have $\Pr[X_{i,j} = 1] = \mathbb{E}[X_{i,j}] = 1/n$ and by linearity of expectation

$$\mathbb{E}[X_j] = \mathbb{E}\left[\sum_{i=1}^{m} X_{i,j}\right] = \sum_{i=1}^{m} \mathbb{E}[X_{i,j}] = \sum_{i=1}^{m} \frac{1}{n} = \frac{m}{n}$$

as claimed. □

Theorem 2.5. *Let X denote the number of empty bins when m balls are thrown independently into n bins. Then we have*

$$\mathbb{E}[X] = n \cdot \left(1 - \frac{1}{n}\right)^m \simeq n \cdot e^{-m/n}.$$

Proof. Define an indicator variable

$$X_{i,j} = \begin{cases} 1 & \text{ball } i \text{ misses bin } j, \\ 0 & \text{otherwise,} \end{cases}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. We obviously have

$$\Pr[X_{i,j} = 1] = \mathbb{E}[X_{i,j}] = 1 - \frac{1}{n}.$$

For any $j = 1, \ldots, m$, $X_j = \Pi_{i=1}^{m} X_{i,j}$ indicates if all balls have missed bin j. By independence,

$$\mathbb{E}[X_j] = \mathbb{E}[\Pi_{i=1}^{m} X_{i,j}] = \Pi_{i=1}^{m} \mathbb{E}[X_{i,j}] = \left(1 - \frac{1}{n}\right)^m,$$

where we have used $\mathbb{E}[X \cdot Y] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$ for independent X and Y. Finally, linearity of expectation yields

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{j=1}^{n} X_j\right] = \sum_{j=1}^{n} \mathbb{E}[X_j] = \sum_{j=1}^{n} \left(1 - \frac{1}{n}\right)^m = n \cdot \left(1 - \frac{1}{n}\right)^m \simeq n \cdot e^{-m/n},$$

as claimed. □

Suppose we have $m = n$. By Theorem 2.4 we expect that each bin contains one ball. But by Theorem 2.5 we have $\mathbb{E}[X] = n \cdot (1 - 1/n)^n \simeq n/e$ (for n sufficiently large). This means we expect that a constant fraction of the bins remains empty. At first sight this may seem contradictive, but it is not, because there is also a constant fraction of the bins that contain more than one ball. (Excercise.)

2.2.2 Coupon Collector

The Coupon Collector problem is the following: Suppose there are n types of coupons in a lottery and each lot contains one coupon (with probability $1/n$ each). How many lots have to be bought (in expectation) until we have at least one coupon of each type. (In terms of a Balls into Bins process, the question is, how many balls have to be thrown (in expectation) until no bin is empty.)

Example 2.6. A famous German brand of Haselnusstafeln hides a picture of some footballplayer in each sweet. How many sweets have to be bought until one has a picture of each player (under the unrealistic assumption that all players are equiprobable)?

Theorem 2.7. *Let X be the number of lots bought until at least one coupon of each type is drawn. Then we have*

$$\mathbb{E}[X] = n \cdot H_n,$$

where $H_n = \sum_{i=1}^{n} 1/i$ denotes the n-th Harmonic number.

Proof. We partition the process of drawing lots into phases. In phase P_i for $i = 1, \ldots, n$ we have already collected $i-1$ distinct coupons and the phase ends once we have drawn i distinct coupons. Let X_i be the number of lots bought in phase P_i.

Suppose we are in phase P_i, then the probability that the next lot terminates this phase is $(n-i+1)/n$. This is because there are $n-(i-1)$ many coupon-types we have not yet collected. Any of those coupons will be the i-th distinct type to be collected (since we have exactly $i-1$ at the moment). These events happen with probability $1/n$, each. These considerations imply that the random variable X_i has geometric distribution with success-probability $(n-i+1)/n$, i.e., $X_i \sim \text{Geo}((n-i+1)/n)$. Its expected value is the reciprocal, i.e., $\mathbb{E}[X_i] = n/(n-i+1)$.

Now we invoke linearity of expectation and obtain

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i=1}^{n} \frac{n}{n-i+1} = n \cdot H_n$$

as claimed. □

Recall that $\log n \leq H_n \leq \log n + 1$. Thus we basically have to buy $n \log n$ lots.

2.2.3 Quicksort

The problem of Sorting is the following: We are given a sequence $x = (x_1, x_2, \ldots, x_n)$ of (pairwise distinct) numbers and are asked to find a permutation π of $(1, 2, \ldots, n)$ such that the sequence $(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$ satisfies $x_{\pi(1)} \leq x_{\pi(2)} \leq \cdots \leq x_{\pi(n)}$. The assumption that the numbers are pairwise distinct can easily be removed, but we consider it for clarity of exposition. Any algorithm for this problem is allowed to ask queries of the type "$a < b$?", called a comparison. We are interested in the number of comparisons of an algorithm A given a sequence x.

The idea of the algorithm Quicksort is to choose some element p from x, called the *pivot*, and divide x into two subsequences x' and x''. The sequence x' contains the elements $x_i < p$ and x'' those $x_i > p$. Quicksort is then called recursively until the input-sequence is empty. The sequence $(\text{Quicksort}(x'), p, \text{Quicksort}(x''))$ is finally returned.

The exposition of Quicksort here is actually not yet a well-defined algorithm, because we have not yet said how we want to choose the pivot element p. This choice drastically affects the running time as we will see shortly. In the sequel let X denote the number of comparisons "$<$" executed by Quicksort.

Deterministic Algorithm

Suppose we choose always the first element in the input-sequence, i.e., $p = x_1$. It is well-known that this variant of Quicksort has the weakness that it may require $\Omega(n^2)$ comparisons.

Observation 2.8. *There is an instance with $X = n(n-1)/2$.*

Algorithm 2.1 QUICKSORT

Input. Sequence $(x_1, x_2, \ldots, x_n)$

Output. Sequence $(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$

(1) If $n = 0$ return.

(2) Otherwise choose $p \in x$ arbitrarily and remove p from x. Let x' and x'' be two empty sequences.

(3) For $i = 1, \ldots, n$, if $x_i < p$ append x_i to x', otherwise append x_i to x''.

(4) Return (QUICKSORT$(x'), p,$ QUICKSORT$(x''))$

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

Randomized Algorithm

Now suppose that we always choose the pivot element equiprobably among the available elements. This gives obviously rise to a Las Vegas algorithm, because we will never compute a wrong result. These choices merely affect the (expected) running time.

Theorem 2.9. *We have that* $\mathbb{E}[X] = 2(n+1)H_n - 4n$.

Proof. Without loss of generality (by renaming the numbers in x), we assume that the original sequence x is a permutation of $(1, 2, \ldots, n)$. So, for any $i < j \in \{1, 2, \ldots, n\}$ let the random variable $X_{i,j}$ 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 $X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{i,j}$. Thus, by linearity of expectation,

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

which shows that 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 x and the random choices of the algorithm induce a random sequence $P = (P_1, P_2, \ldots, P_n)$ of pivots.

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 from the set $\{i, \ldots, j\}$ in the sequence P. If i and j are compared, then either one must be the pivot and they must be in the same subsequence of x. 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 of x, otherwise. Hence, either i or j is the first pivot in the set $\{i, \ldots, j\}$ appearing in P. The converse direction is trivial: If one of i or j is the first pivot from the set $\{i, \ldots, j\}$ in P, then i and j are still in the same subsequence of x and will hence be compared.

What is the probability that, say, i is the first pivot of $\{i, \ldots, j\}$? Consider the subsequence S of P induced by the elements $\{i, \ldots, j\}$. 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 P, i.e., the first element in S. Since the pivot is uniformly distributed the probability that i is the first element of S is exactly $1/(j - i + 1)$. Analogous reasoning for j yields the overall probability $\Pr[X_{i,j} = 1] = 2/(j - i + 1)$.

This allows us to continue our calculation

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

and the proof is complete. □

So we have shown that the expected number of comparisons of QUICKSORT is $O(n \log n)$. In the next chapter, we will 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.

Chapter 3

Bounds on Probabilities

3.1 Basics

Consider the question: "How meaningful is the expected value of a random variable?" We begin with an introductory example. Define for any $c \geq 1$ a random variable X_c by

$$X_c = \begin{cases} c & \text{with probability } 1/c, \\ 0 & \text{otherwise.} \end{cases}$$

We obviously have $\mathbb{E}[X_c] = 1$ for any $c \geq 1$, i.e., in terms of expected value the X_c are indistinguishable. However, the variables are obviously very different: The value of X_1 is just a constant, while the value of X_c for $c > 1$ is either "huge" or "tiny". Hence it is interesting to consider the random variable $|X_c - \mathbb{E}[X_c]|$, because it measures if X_c, takes a value "far away" from the expected value.

Generally speaking, in this chapter, we are interested in inequalities of the form

$$\Pr[|X - \mathbb{E}[X]| \geq t] \leq f(t).$$

3.2 Markov and Chebyshev

The most basic such inequalities are those of Markov and Chebyshev.

Theorem 3.1 (Markov)**.** *Let $X \geq 0$ be any random variable. Then, for any $t > 0$ it holds that*

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

Proof. For any $t > 0$ define the indicator variable

$$Y = \begin{cases} 1 & \text{if } X \geq t, \\ 0 & \text{otherwise.} \end{cases}$$

Observe that $Y \leq X/t$ since $X \geq 0$. Now, using $\Pr[X \geq t] = \mathbb{E}[Y]$ since Y is a 0/1-variable, we calculate

$$\Pr[X \geq t] = \mathbb{E}[Y] \leq \mathbb{E}\left[\frac{X}{t}\right] = \frac{\mathbb{E}[X]}{t}$$

and the claim is established. □

The *variance* of any random variable X is

$$\mathrm{Var}[X] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2.$$

Since the random variable $(X - \mathbb{E}[X])^2$ is never negative, the variance of X is also never negative.

Theorem 3.2 (Chebyshev)**.** *Let X be any random variable. Then, for any $t > 0$ it holds that*

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

Proof. First notice the equality $\Pr[|X - \mathbb{E}[X]| \geq t] = \Pr\left[(X - \mathbb{E}[X])^2 \geq t^2\right]$. Now we apply Theorem 3.1 to the non-negative random variable $(X - \mathbb{E}[X])^2$ and obtain

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

completing the proof. □

Example 3.3. Let X be the number of "heads" when a fair coin is flipped n times. As X is obviously binomial distributed, i.e., $X \sim \mathrm{Bin}(n, 1/2)$, we have $\mathbb{E}[X] = n/2$ and $\mathrm{Var}[X] = n/4$. Suppose we are interested in this question: "How likely is it that at least $3n/4$ many times 'heads' is obtained?"

Theorem 3.1 tells us

$$\Pr[X \geq 3n/4] \leq \frac{n/2}{3n/4} = \frac{2}{3}.$$

Theorem 3.2 tells us

$$\Pr[X \geq 3n/4] = \Pr[X - n/2 \geq n/4] \leq \Pr[|X - \mathbb{E}[X]| \geq n/4] \leq \frac{n/4}{(n/4)^2} = \frac{4}{n}.$$

Thus, for sufficiently large n, Chebyshev's inequality gives a substantially better bound than Markov's inequality.

3.3 Bounds with Moment-Generating Functions

The inequalities of Markov and Chebyshev are rather general as they hold for basically any random variable. Substantially stronger bounds can be obtained for special cases. Here we first give a general approach for deriving such stronger bounds and then apply it for sums of independent variables.

3.3.1 Method

Let X be any random variable. The basic idea is to take the random variable e^{sX} (with free parameter $s \geq 0$) and apply Markov's inequality, i.e.,

$$\Pr[X \geq t] = \Pr\left[e^{sX} \geq e^{st}\right] \leq \frac{\mathbb{E}\left[e^{sX}\right]}{e^{st}}.$$

Now, *if* one can show that $\mathbb{E}\left[e^{sX}\right]$ is not "too large", then the above inequality looks promising, since the denominator grows *exponentially* in t. Compare this with Chebyshev's

inequality, where the denominator only grows quadratically. The free parameter s is used to optimize the bound we obtain.

Let X be any random variable. For any $k \in \mathbb{N}_0$, the value $M_k = \mathbb{E}\left[X^k\right]$ is called the *k-th moment* X. The function $M_X(s) = \mathbb{E}\left[e^{sX}\right]$ in the indeterminate s is called the *moment-generating function* of X. The reason for the nomenclature is explained by the following property of $M_X(s)$. For any $k \in \mathbb{N}_0$ let $M_X^{(k)}(s)$ be the k-th derivative of M_X evaluated at s. The theorem below basically says that one can compute the k-th moment by evaluating the k-th derivative of M_X at zero.

Theorem 3.4. *Let X be any random variable. If exchanging the expectation- and differentiation-operations is legitimate for X and any $k \in \mathbb{N}_0$, then*

$$\mathbb{E}\left[X^k\right] = M^{(k)}(0).$$

Proof. We prove by induction that $M_X^{(k)}(s) = \mathbb{E}\left[X^k e^{sX}\right]$ under the assumption that exchanging the expectation- and differentiation-operations is legitimate for X and any $k \in \mathbb{N}_0$. The base case is $M_X^{(0)}(s) = M_X(s) = \mathbb{E}\left[e^{sX}\right] = \mathbb{E}\left[X^0 e^{sX}\right]$. For the inductive case we have

$$M_X^{(k+1)}(s) = \frac{d}{ds} M_X^{(k)}(s) = \frac{d}{ds}\mathbb{E}\left[X^k e^{sX}\right] = \mathbb{E}\left[\frac{d}{ds} X^k e^{sX}\right] = \mathbb{E}\left[X^{k+1} e^{sX}\right].$$

Evaluating $M_X^{(k)}(0)$ yields the claim. □

It is known that exchanging the expectation- and differentiation-operations is legitimate for X for any $k \in \mathbb{N}_0$ if M_X exists in a neighborhood of zero. This will always be the case in the applications considered here. Another important property is factorization stated below.

Theorem 3.5. *Let X and Y be independent random variables. Then we have*

$$M_{X+Y}(s) = M_X(s) \cdot M_Y(s).$$

Proof. If any two random variables X and Y are independent, then we have $\mathbb{E}[X \cdot Y] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$. Now

$$M_{X+Y}(s) = \mathbb{E}\left[e^{s(X+Y)}\right] = \mathbb{E}\left[e^{sX} \cdot e^{sY}\right] = \mathbb{E}\left[e^{sX}\right] \cdot \mathbb{E}\left[e^{sY}\right] = M_X(s) \cdot M_Y(s),$$

where we have used that e^{sX} and e^{sY} are independent if X and Y are. □

Finally, here is the theorem which we will apply later on for deriving bounds on the concentration of measure.

Theorem 3.6. *Let X be any random variable. Then, for any $t > 0$ we have*

$$\Pr[X \geq t] \leq \min_{s \geq 0} \frac{M_X(s)}{e^{st}} \text{ and } \Pr[X \leq t] \leq \min_{s \leq 0} \frac{M_X(s)}{e^{st}}.$$

Proof. By Theorem 3.1 we clearly have

$$\Pr[X \geq t] = \Pr\left[e^{sX} \geq e^{st}\right] \leq \frac{\mathbb{E}\left[e^{sX}\right]}{e^{st}} = \frac{M_X(s)}{e^{st}}$$

for any $s \geq 0$. We simply minimize the function $M_X(s)/e^{st}$ with respect to $s \geq 0$ to obtain the best bound the approach can give. The proof for $\Pr[X \leq t]$ is analogous. □

The approach is often useful when $X = \sum_{i=1}^{n} X_i$ is a sum of independent random variables. This is because we only have to compute the moment generating functions of the individual X_i. Repeated application of Theorem 3.5 gives:

Corollary 3.7. *Let $X = \sum_{i=1}^{n} X_i$, where the X_i are independent. Then, for any $t > 0$ we have*

$$\Pr[X \geq t] \leq \min_{s \geq 0} \frac{\Pi_{i=1}^{n} M_{X_i}(s)}{e^{st}} \text{ and } \Pr[X \leq t] \leq \min_{s \leq 0} \frac{\Pi_{i=1}^{n} M_{X_i}(s)}{e^{st}}.$$

3.3.2 Chernoff

Here we are interested in

$$X = \sum_{i=1}^{n} X_i,$$

where the $X_i \in \{0, 1\}$ are independent indicator variables.

Let $B \in \{0, 1\}$ with $\Pr[B = 1] = p$ and $\Pr[B = 0] = 1 - p$. Then

$$M_B(s) = \mathbb{E}\left[e^{sB}\right] = e^{s \cdot 0} \cdot (1 - p) + e^{s \cdot 1} \cdot p = 1 + p(e^s - 1) \leq e^{p(e^s - 1)},$$

where we have used the inequality $1 + x \leq e^x$ which holds for any x.

Lemma 3.8. *Let $X = \sum_{i=1}^{n} X_i$, where the $X_i \in \{0, 1\}$ are independent indicator variables. Then we have*

$$M_X(s) \leq e^{(e^s - 1)\mathbb{E}[X]}.$$

Proof. Let $p_i = \Pr[X_i = 1]$. Now, by Theorem 3.5 we have

$$M_X(s) = \Pi_{i=1}^{n} M_{X_i}(s) \leq \Pi_{i=1}^{n} e^{(e^s - 1)p_i} = e^{(e^s - 1)\sum_{i=1}^{n} p_i} = e^{(e^s - 1)\mathbb{E}[X]}$$

as claimed. □

In the theorem below, the first bound is stronger, but the second one is easier to state and to compute with in many situations.

Theorem 3.9 (Chernoff – upper tail)**.** *Let $X = \sum_{i=1}^{n} X_i$, where the $X_i \in \{0, 1\}$ are independent indicator variables. Then we have*

(i) for any $\delta > 0$

$$\Pr[X \geq (1 + \delta)\mathbb{E}[X]] \leq \left(\frac{e^{\delta}}{(1 + \delta)^{(1+\delta)}}\right)^{\mathbb{E}[X]},$$

(ii) for $0 < \delta \leq 1$

$$\Pr[X \geq (1 + \delta)\mathbb{E}[X]] \leq e^{-\mathbb{E}[X]\delta^2/3}.$$

Proof. To obtain (i), apply Theorem 3.6 to $t = (1+\delta)\mathbb{E}[X]$ and Lemma 3.8 to yield

$$\Pr[X \geq (1+\delta)\mathbb{E}[X]] \leq \min_{s \geq 0} \frac{e^{(e^s-1)\mathbb{E}[X]}}{e^{s(1+\delta)\mathbb{E}[X]}} \leq \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^{\mathbb{E}[X]},$$

where we have chosen $s = \ln(1+\delta) > 0$ which is true for any $\delta > 0$.

For (ii) we have to show that

$$\frac{e^\delta}{(1+\delta)^{(1+\delta)}} \leq e^{-\delta^2/3}.$$

Take the natural logarithm on both sides which yields that we have to show

$$f(\delta) := \delta - (1+\delta)\ln(1+\delta) + \frac{\delta^2}{3} \leq 0.$$

The first and second derivatives are $f'(\delta) = -\ln(1+\delta) + 2\delta/3$ and $f''(\delta) = -1/(1+\delta) + 2/3$. We have $f''(\delta) < 0$ for $0 \leq \delta \leq 1/2$ and $f''(\delta) > 0$ for $\delta > 1/2$. Thus f' first decreases and then increases in the interval $[0,1]$. As $f'(0) = 0$ and $f'(1) < 0$ we have $f'(\delta) \leq 0$ for $\delta \in [0,1]$. Since $f(0) = 0$ we can conclude $f(\delta) \leq 0$ for $\delta \in [0,1]$. □

The proof of the following symmetric version of the Chernoff bound is left as an excercise.

Theorem 3.10 (Chernoff – lower tail)**.** *Let $X = \sum_{i=1}^n X_i$, where the $X_i \in \{0,1\}$ are independent indicator variables. Then we have*

(i) for $0 < \delta < 1$

$$\Pr[X \leq (1-\delta)\mathbb{E}[X]] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mathbb{E}[X]},$$

(ii) for $0 < \delta < 1$

$$\Pr[X \leq (1-\delta)\mathbb{E}[X]] \leq e^{-\mathbb{E}[X]\delta^2/2}.$$

Corollary 3.11. *Let $X = \sum_{i=1}^n X_i$, where the $X_i \in \{0,1\}$ are independent indicator variables. Then we have for any $0 < \delta < 1$*

$$\Pr[|X - \mathbb{E}[X]| \geq \delta\mathbb{E}[X]] \leq 2e^{-\mathbb{E}[X]\delta^2/3}.$$

Example 3.12. Recall Example 3.3, where we have n fair coin flips and X counts the number of "heads". Theorem 3.9 tells us

$$\Pr[X \geq 3n/4] = \Pr[X \geq (1+1/2)\mathbb{E}[X]] \leq e^{-\frac{n/2\cdot(1/2)^2}{3}} = e^{-n/24},$$

which is really much stronger than Markov or Chebyshev for sufficiently large n.

3.3.3 Bernstein and Azuma-Hoeffding

One of the drawbacks of Chernoff's inequality is that it only covers indicator variables. Here we present Bernstein's inequality and Azuma's inequality (without proof), which can handle more general random variables. The principal approach is again by moment-generating functions.

Theorem 3.13 (Bernstein)**.** *Let $X_1, \ldots, X_n$ be independent random variables with $\mathbb{E}[X_i] = 0$ and $|X_i| \leq M$ for $i = 1, \ldots, n$. Let $X = \sum_{i=1}^{n} X_i$. Then for all $t > 0$ we have*

$$\Pr[X > t] \leq e^{-\frac{t^2/2}{\sum_{i=1}^{n} \mathbb{E}\left[X_i^2\right] + M \cdot t/3}}.$$

The proof basically uses Chebyshev's inequality applied to the random variable e^{sX}, for a suitable choice of $s > 0$. Notice that the assumption $\mathbb{E}[X_i] = 0$ is not essential since we can always choose $X_i = Y_i - \mathbb{E}[Y_i]$ for general Y_i. With $Y = \sum_{i=1}^{n} Y_i$ we also have $X = Y - \mathbb{E}[Y]$. Another important bound is Azuma-Hoeffding's inequality.

Theorem 3.14 (Azuma-Hoeffding)**.** *Let $X_1, \ldots, X_n$ be independent random variables with $a \leq X_i \leq b$ for $i = 1, \ldots, n$. Let $X = \frac{1}{n} \cdot \sum_{i=1}^{n} X_i$. Then, for any $t > 0$ we have*

$$\Pr[|X - \mathbb{E}[X]| \geq t] \leq 2e^{-\frac{n \cdot t^2}{(b-a)^2}}.$$

3.4 Applications

3.4.1 Balls Into Bins

Recall that in the Balls into Bins model there are n bins and m balls are thrown independently and uniformy at random. The theorem below essentially states that it is unlikely that we have $O\left(\sqrt{m/n \cdot \ln n}\right)$ more balls than expected in any fixed bin j.

Theorem 3.15. *Let X_j denote the number of balls in bin j. Then, for $j = 1, \ldots, m$ and any $c > 0$ with $\sqrt{3c \cdot n/m \cdot \ln n} \leq 1$ we have*

$$\Pr\left[X_j \geq \frac{m}{n} + \sqrt{3c \cdot \frac{m}{n} \cdot \ln n}\right] \leq n^{-c}.$$

Proof. Recall from Theorem 2.4 that $\mathbb{E}[X_j] = m/n$. Let $X_{i,j}$ indicate if ball i falls into bin j. The $X_{i,j}$ are independent indicator variables, so Chernoff can be applied. We use Theorem 3.9 and the choice $\delta = \sqrt{3c \cdot n/m \cdot \ln n} \leq 1$ to obtain

$$\Pr\left[X_j \geq \frac{m}{n} + \sqrt{3c \cdot \frac{m}{n} \cdot \ln n}\right] = \Pr[X \geq (1+\delta)\mathbb{E}[X]] \leq e^{-\mathbb{E}[X]\delta^2/3} = n^{-c}$$

as claimed. □

Chernoff's inequality also allows us to estimate the (expected) maximum number of balls in any bin.

Theorem 3.16. *Let X_j denote the number of balls in bin j and let $X = \max_{j \in \{1,\ldots,m\}} X_j$. Then, for any $c > 0$ with $\sqrt{3c \cdot n/m \cdot \ln(n+m)} \leq 1$ we have*

$$\Pr\left[X > \frac{m}{n} + \sqrt{3c \cdot \frac{m}{n} \cdot \ln(n+m)}\right] \leq (n+m)^{-c+1}$$

and

$$\mathbb{E}[X] \leq \frac{m}{n} + \sqrt{6 \cdot \frac{m}{n} \cdot \ln(n+m)} + 1.$$

Proof. We apply the union bound and Theorem 3.9 with $\delta = \sqrt{3c \cdot n/m \cdot \ln(n+m)}$ and $\mathbb{E}[X_j] = m/n$ for $j = 1, \ldots, m$ to find

$$\Pr[X \geq t] = \Pr[X_1 \geq t \vee \cdots \vee X_m \geq t] \leq \sum_{j=1}^{m} \Pr[X_j \geq t] \leq m \cdot (n+m)^{-c} \leq (n+m)^{-c+1},$$

where $t = (1+\delta) \cdot \mathbb{E}[X]$. We trivially have $X \leq n$. Now choose $c = 2$ in the definition of t, observe $\Pr[X \leq t] \leq 1$ and obtain

$$\begin{aligned}
\mathbb{E}[X] &= \mathbb{E}[X \mid X < t] \Pr[X < t] + \mathbb{E}[X \mid X \geq t] \Pr[X \geq t] \\
&\leq t \cdot 1 + n \cdot (n+m)^{-c+1} \\
&\leq \frac{m}{n} + \sqrt{6 \cdot \frac{m}{n} \cdot \ln(n+m)} + 1,
\end{aligned}$$

as claimed. □

3.4.2 Coupon Collector

Recall the Coupon Collector problem, where we have n types of coupons in a lottery and each lot contains one coupon (with probability $1/n$ each). The theorem below says that it is relatively unlikely that we need more than c times as many lots as expected.

Theorem 3.17. *Let X be the number of lots bought until at least one coupon of each type is drawn. Then, for any $c > 1$ we have*

$$\Pr[X \geq c \cdot \mathbb{E}[X]] \leq \frac{\pi^2/6}{(c-1)^2 H_n^2},$$

where $H_n = \sum_{i=1}^{n} 1/i$ denotes the n-th Harmonic number*.*

Proof. Our goal is to apply Chebyshev's inequality. Thus we have to estimate the variance of X. We partition the process of drawing lots into phases. In phase P_i for $i = 1, \ldots, n$ we have already collected $i-1$ distinct coupons and the phase ends once we have drawn i distinct coupons. Let X_i be the number of lots bought in phase P_i. Recall that the X_i are geometrically distributed with success-probability $(n-i+1)/n$. Also observe that the X_i are independent. For a geometric distributed random variable G with success-probability p we have $\mathbb{E}[G] = 1/p$ and $\mathrm{Var}[G] = (1-p)/p^2 \leq 1/p^2$. Thus we have

$$\begin{aligned}
\mathrm{Var}[X] &= \mathrm{Var}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathrm{Var}[X_i] \\
&\leq \sum_{i=1}^{n} \left(\frac{n}{n-i+1}\right)^2 = n^2 \cdot \sum_{i=1}^{n} \frac{1}{i^2} \\
&\leq n^2 \cdot \frac{\pi^2}{6},
\end{aligned}$$

where we have used the well-known identity $\sum_{i=1}^{\infty} 1/i^2 = \pi^2/6$.

Hence, recalling $\mathbb{E}[X] = n \cdot H_n$ from Theorem 2.7 and applying Chebyshev yields

$$\Pr[X \geq c \cdot \mathbb{E}[X]] \leq \Pr[|X - \mathbb{E}[X]| \geq (c-1)n \cdot H_n] \leq \frac{\pi^2/6}{(c-1)^2 \cdot H_n^2}$$

and the proof is complete. □

3.4.3 Quicksort

Recall the randomized QUICKSORT algorithm. It is relatively unlikely that the algorithm needs c times more comparisons as expected.

Theorem 3.18. *Let X denote the number of comparisions "$<$" needed by the randomized* QUICKSORT *algorithm. For any $c > 1$ we have that*

$$\Pr\left[X \geq c \cdot \mathbb{E}\left[X\right]\right] \leq \frac{1}{(c-1)^2 \cdot H_n} + o\left(H_n^{-1}\right).$$

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

Suppose that we can prove $\mathrm{Var}\left[X\right] \leq 4n^2 \cdot H_n$ then we have

$$\begin{aligned}
\Pr\left[X > c\mathbb{E}\left[X\right]\right] \leq \Pr\left[|X - \mathbb{E}\left[X\right]| > (c-1)\mathbb{E}\left[X\right]\right] &\leq \frac{\mathrm{Var}\left[X\right]}{(c-1)^2\mathbb{E}\left[X\right]^2} \\
&\leq \frac{4n^2 \cdot H_n}{(c-1)^2(2(n+1)\cdot H_n - 4n)^2} \\
&= \frac{1}{(c-1)^2 \cdot H_n \cdot (1 - o\left(1\right))} \\
&= \frac{1}{(c-1)^2 \cdot H_n} + o\left(H_n^{-1}\right),
\end{aligned}$$

which is what we wanted to show.

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

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

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

$$\mathbb{E}\left[X_{i,j}X_{i,k}\right] = \mathbb{E}\left[X_{i,k} \mid X_{i,j} = 1\right]\Pr\left[X_{i,j} = 1\right] \leq \frac{2}{j-i+1}.$$

This yields

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

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

3.4.4 Sampling with Replacement

Consider the following approach to estimate the value of an (unknown) sum. Let $U = \{u_1, \ldots, u_m\}$ be a set of m constants (independent of m) with $a \leq u_i \leq b$ for $i = 1, \ldots, m$ and $u = \sum_{j=1}^{m} u_i$. Suppose we draw n elements independently and uniformly from U (with replacement) and let X denote the sum. For $i = 1, \ldots, n$ and $j = 1, \ldots, m$ let $X_{i,j}$ indicate the event if u_j is chosen in the i-th draw. Then $X = \sum_{i=1}^{n} \sum_{j=1}^{m} u_j \cdot X_{i,j}$ is the sampled sum.

Then the random variable $m/n \cdot X$ is an estimator for u because

$$\mathbb{E}\left[\frac{m}{n} \cdot X\right] = \frac{m}{n} \cdot \mathbb{E}[X] = \frac{m}{n} \cdot \sum_{i=1}^{n} \sum_{j=1}^{m} u_j \cdot \mathbb{E}[X_{i,j}] = \frac{m}{n} \cdot \frac{1}{m} \cdot n \cdot u = u.$$

The question is how good this estimate is. The following theorem, whose proof uses the Azuma-Hoeffding bound, gives an answer. It basically states that one has to sample $\Theta\left(\ln m/\varepsilon^2\right)$ many times in order to have estimated u within an additive error of $m \cdot \varepsilon$.

Theorem 3.19. *With the above notation, for any $c > 0$, if $n = \lceil c \cdot (b-a)^2 \cdot \ln m/\varepsilon^2 \rceil$ then we have*

$$\Pr\left[\frac{m}{n} \cdot X \in [u - \varepsilon \cdot m, u + \varepsilon \cdot m]\right] \geq 1 - 2 \cdot m^{-c}$$

Proof. Let $X_i = \sum_{j=1}^{m} u_j \cdot X_{i,j}$ be the value of the i-th element drawn. The X_i are independent, $a \leq X_i \leq b$ and hence satisfy the assumptions for Theorem 3.14. Let $\overline{X} = \frac{1}{n} \cdot X$ and for any $t > 0$ we have

$$\Pr\left[\left|\frac{m}{n} \cdot X - \frac{m}{n} \cdot \mathbb{E}[X]\right| \geq \varepsilon \cdot m\right] = \Pr\left[|\overline{X} - \mathbb{E}\left[\overline{X}\right]| \geq \varepsilon\right] \leq 2 \cdot e^{-\frac{n \cdot \varepsilon^2}{(b-a)^2}}$$

For the choice $n = \lceil c \cdot (b-a)^2 \cdot \ln m/\varepsilon^2 \rceil$ the claim follows. □

Chapter 4

Probabilistic Method

4.1 Basics

The *probabilistic method* is mainly used for proving the existence of certain (mathematical) objects without explicitly constructing them. By principle, if one is interested if an object with certain properties exists, one defines a (suitable) probability space over all candidate-objects. If the probability of choosing an object with the required properties is *strictly* positive, then the object in question must exist.

4.2 Expectation Argument

The theorem below can be used as a basis for probabilistic existence proofs. It states that, *it is possible* that a random variable takes a value not smaller (respectively not larger) than its expectation.

Theorem 4.1. *Let X be a random variable (defined over some suitable probability space) with $\mathbb{E}[X] = \mu$. Then*

$$\Pr[X \geq \mu] > 0 \quad \textit{and} \quad \Pr[X \leq \mu] > 0.$$

Proof. Assume that $\Pr[X \geq \mu] = 0$. Then

$$\mu = \mathbb{E}[X] = \mathbb{E}[X \mid X < \mu] \Pr[X < \mu] + \mathbb{E}[X \mid X \geq \mu] \Pr[X \geq \mu] < \mu$$

yields a contradiction. Similarly for $\Pr[X \leq \mu] = 0$. □

4.3 First and Second Moment Method

The *first and second moment method* is another rather powerful probabilistic method. The following innocent looking result is its basis.

Theorem 4.2 (First and Second Moment Method)**.** *Let $X_n \in \mathbb{N}_0$ be a sequence of (integer-valued) random variables for $n = 0, 1, \ldots$.*

(i) If $\lim_{n\to\infty} \mathbb{E}[X_n] = 0$ then

$$\lim_{n\to\infty} \Pr[X_n = 0] = 1.$$

(ii) If $\lim_{n\to\infty} \operatorname{Var}[X_n] / \mathbb{E}[X_n]^2 = 0$ *then*

$$\lim_{n\to\infty} \Pr[X_n = 0] = 0.$$

Proof. The first moment method (i) is an application of Markov's inequality

$$\Pr[X_n > 0] = \Pr[X_n \geq 1] \leq \frac{\mathbb{E}[X_n]}{1} \to 0 \quad \text{for } n \to \infty.$$

The second moment method (i) uses Chebyshev's inequality

$$\Pr[X_n = 0] \leq \Pr[|X_n - \mathbb{E}[X_n]| \geq \mathbb{E}[X_n]] \leq \frac{\operatorname{Var}[X_n]}{\mathbb{E}[X_n]^2} \to 0 \quad \text{for } n \to \infty,$$

as claimed. □

4.4 Applications

4.4.1 Maximum Satisfiability

Theorem 4.3. *Let F be a Boolean formula consisting of m clauses, where clause number i has k_i literals for $i = 1, \ldots, m$. Let $k = \min_{i=1,\ldots,m} k_i$. Then there is a truth assignment satisfying at least*

$$\sum_{i=1}^{m} (1 - 2^{-k_i}) \geq m \cdot (1 - 2^{-k})$$

many clauses.

Proof. We set each variable in the clause to 0 with probability 1/2 and 1 otherwise. Let X_i be an indicator if clause i is satisfied under such a random assignment. Then $X = \sum_{i=1}^{m} X_i$ is the number of satisfied clauses. We obviously have $\mathbb{E}[X_i] = \Pr[X_i = 1] = 1 - 2^{-k_i}$. Hence using linearity of expectation yields

$$\mathbb{E}[X] = \sum_{i=1}^{m} \mathbb{E}[X_i] = \sum_{i=1}^{m} (1 - 2^{-k_i}) \geq m \cdot (1 - 2^{-k})$$

and Theorem 4.1 the claim. □

4.4.2 Independent Sets

Theorem 4.3 was a direct application of the expectation argument given in Theorem 4.1. It is sometimes helpful to proceed as follows: First sample a structure which does not necessarily satisfy a required property. Then modify the sampled object until the properity is met. Thus this technique is called *sample-and-modify*.

Let $G = (V, E)$ be a graph. A set $I \subseteq V$ is called *independent* if $i, j \in I$ imply $ij \notin E$. It is NP-hard to determine the size of the largest independent set in a graph G. The sample-and-modify technique yields a lower bound. The method actually yields a (randomized) algorithm constructing an independent set of a certain size. By accident, the proof is *constructive*, as we say.

Theorem 4.4. *Let $G = (V, E)$ be a graph on n vertices and $m \geq n/2$ edges. Then G has an independent set of size at least $n^2/4m$.*

Proof. Let $d = 2m/n$ denote the average degree. Consider the following randomized algorithm:

(1) Delete each vertex of G (including its incident edges) independently with probability $1 - 1/d$.

(2) For each remaining edge, remove it and *one* of its incident vertices.

The vertices surviving step (1) are not necessarily independent. Those surviving step (2) are, since each edge was removed. How many are those (in expectation)?

Let X be the number of vertices surviving step (1). Since $m \geq n/2$ we have $1-1/d \geq 0$ is actually a legal probability. Each vertex survives with probability $1/d$. Thus $\mathbb{E}[X] = n/d$. Let Y be the number of edges that survive step (1). An edge survives if both its endpoints survive. Initially there are $m = nd/2$ edges in G. Thus $\mathbb{E}[Y] = nd/2 \cdot (1/d)^2 = n/2d$. Step (2) removes all remaining edges and one of its endpoints. That is, at most Y additional vertices are deleted.

Let Z denote the number of vertices surviving step (2). Then we obviously have $Z \geq X - Y$ and hence

$$\mathbb{E}[Z] \geq \mathbb{E}[X] - \mathbb{E}[Y] = \frac{n}{d} - \frac{n}{2d} = \frac{n}{2d} = \frac{n^2}{4m}.$$

The claim follows with Theorem 4.1. □

4.4.3 Random Graphs

One of the most successful topics for the first and second moment method is the area of *random graphs*. Let $n \in \mathbb{N}$ be a positive integer and let $0 \leq p \leq 1$. The *random graph* $G_{n,p}$ is a probability space over the set of graphs on the vertex set $V = \{1, \ldots, n\}$ determined by

$$\Pr[ij \in E] = p \quad \text{for all } i, j \in V,$$

where these events are mutally independent. That is, for each vertex-pair i, j, the edge ij is present in the graph with probability p, independently.

In this section, we demontrate the use of the first and second moment method, where we are interested in certain properties of $G_{n,p}$.

Let P be a graph property. Let $r(n)$ and $p(n)$ be functions of n. We say that $r(n)$ is a *threshold function* for property P if we have:

(i) If $p(n) \ll r(n)$ then

$$\lim_{n\to\infty} \Pr\left[G_{n,p(n)} \text{ has property } P\right] = 1.$$

(ii) If $p(n) \gg r(n)$ then

$$\lim_{n\to\infty} \Pr\left[G_{n,p(n)} \text{ has property } P\right] = 0.$$

Or vice versa. The notation $\ll$ means that $\lim_{n\to\infty} p(n)/r(n) = 0$, while similarly $\gg$ means $\lim_{n\to\infty} p(n)/r(n) = \infty$.

Isolated Vertices

Let $G = (V, E)$ be a graph on n vertices $V = \{1, \ldots, m\}$ and edges E. A vertex $i \in V$ is called *isolated* if $\{i, j\} \notin E$ for all $i \neq j \in V$.

Theorem 4.5. *The function $r(n) = \ln n/n$ is a threshold function for "has isolated vertices" in $G_{n,p(n)}$.*

Proof. For a graph $G = (V, E)$ let X_i indicate if a vertex $i \in V$ is isolated. Thus $X = \sum_{i \in V} X_i$ counts the number of isolated vertices. In $G_{n,p}$ with $p := p(n)$ we have

$$\mathbb{E}[X_i] = \Pr[X_i = 1] = (1-p)^{n-1} \leq e^{-(n-1)p},$$

having used $1 + x \leq e^x$ for any $x \in \mathbb{R}$. We first consider $p(n) \gg r(n) = \ln n/n$ and have

$$\mathbb{E}[X] \leq n \cdot e^{-(n-1)p} = e^{\ln n - (n-1)p} = e^{n(r(n)-p(n))+p(n)} \to 0 \quad \text{for } n \to \infty.$$

Hence, with Theorem 4.2

$$\lim_{n\to\infty} \Pr[X = 0] = 1,$$

i.e., $G_{n,p}$ does not have isolated vertices for $p(n) \gg r(n) = \ln n/n$ with probability tending to one if n tends to infinity.

For the other statement we have to estimate the variance $\mathrm{Var}[X] = \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2$, i.e., essentially $\mathbb{E}\left[X^2\right]$.

$$\begin{aligned}
\mathbb{E}\left[X^2\right] &= \mathbb{E}\left[\left(\sum_{i \in V} X_i\right)^2\right] \\
&= \sum_{i \neq j \in V} \mathbb{E}[X_i \cdot X_j] + \sum_{i \in V} \mathbb{E}[X_i] \\
&= \sum_{i \neq j \in V} \mathbb{E}[X_i \cdot X_j] + \mathbb{E}[X].
\end{aligned}$$

For $i \neq j$ we have $X_i \cdot X_j = 1$ if both vertices are isolated. That is, the edge ij is absent and the remaining $2 \cdot (n-2)$ edges connecting i and j with the rest of the vertices as well. Thus

$$\mathbb{E}[X_i \cdot X_j] = (1-p)^{2(n-2)+1} = (1-p)^{2n-3}$$

Therefore, as there are $n(n-1)$ such pairs we have

$$\begin{aligned}
\mathrm{Var}[X] &= n(n-1)(1-p)^{2n-3} + n(1-p)^{n-1} - n^2(1-p)^{2(n-1)} \\
&\leq n^2(1-p)^{2n-3}(1-(1-p)) + n(1-p)^{n-1} \\
&= n^2(1-p)^{2n-3}p + n(1-p)^{n-1}.
\end{aligned}$$

By using the expansion $\ln(1+x) = \sum_{i=1}^{\infty} 1/i \cdot (x/(1+x))^i$ for $x > -1/2$ we find that

$$\frac{\mathrm{Var}[X]}{\mathbb{E}[X]^2} \leq \frac{p}{1-p} + \frac{1}{n(1-p)^{n-1}} \to 0 \quad \text{for } n \to \infty,$$

since $p(n) \ll r(n) = \ln n/n$.

Now Theorem 4.2 implies

$$\lim_{n\to\infty} \Pr[X = 0] = 0,$$

i.e., $G_{n,p}$ does have isolated vertices for $p(n) \ll r(n) = \ln n/n$ with probability tending to one if n tends to infinity. □

Chapter 5

Randomized Rounding

Randomized rounding is a powerful method in the design and analysis of approximation algorithms. An approximation algorithm runs in polynomial time and determines a feasible solution (for a usually NP-hard problem), which is provably not "too far" away from a best-possible solution.

5.1 Basics

Many optmization problems can be formulated in terms of Integer Linear Programming (ILP). We are given an $m \times n$-matrix $A = (a_{i,j})_{1 \le i \le m, 1 \le j \le n}$ and vectors $b = (b_i)_{1 \le i \le m}$ and $c = (c_j)_{1 \le j \le n}$. Hence an instance is $I = (A, b, c)$. Furthermore, we have a vector $x = (x_j)_{1 \le j \le n}$ of *integer* variables, i.e., $x_j \in \mathbb{Z}$. The problem is to extremize, i.e., minimize or maximize, the linear function $\text{val}(x) = \sum_{j=1}^{n} c_j x_j$, subject to $A \cdot x \ge b$. val is also called the *objective function* and a vector x which satisfies $A \cdot x \ge b$ is called *feasible solution*. The set $F(A, b) = \{x \in \mathbb{Z}^n : A \cdot x \ge b\}$ is the *feasible region*. A feasible solution x^* where the desired extremum is attained is called *optimal*. We often write $\text{OPT}(I) = \text{val}(x^*)$.

Problem 5.1 Integer Linear Programming

Instance. Matrix $A \in \mathbb{R}^{m,n}$, vectors $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$

Task. Solve the problem

$$\begin{aligned} \text{minimize} \quad & \text{val}(x) = \sum_{j=1}^{n} c_j x_j, \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{i,j} x_j \ge b_i \quad i = 1, \ldots, m, \\ & x_j \in \mathbb{Z} \quad j = 1, \ldots, n. \end{aligned}$$

As it is in general NP-hard to solve ILP, we often resort to approximation algorithms. We consider a minimization problem w.l.o.g. here. A polynomial time algorithm ALG is called a *c-approximation algorithm* for Integer Linear Programming, if it produces for any instance $I = (A, b, c)$ a feasible solution $x \in F(A, b)$ with objective value $\text{ALG}(I) = \text{val}(x)$ satisfying

$$\frac{\text{ALG}(I)}{\text{OPT}(I)} \le c.$$

If we have a maximization problem, then we require $\text{ALG}(I)/\text{OPT}(I) \geq c$.

A common strategy for designing approximation algorithms for ILP is called *LP-relaxation*. If we replace each constraint $x_j \in \mathbb{Z}$ by $x_j \in \mathbb{R}$, then the resulting problem can be solved in polynomial time as it is an instance of (ordinary) Linear Programming (LP). Let z be an optimal solution for the relaxed problem. Observe that

$$\text{val}(z) \leq \text{val}(x^*),$$

if the original ILP problem was a minimization problem. However, the optimal solution z for the LP is in general infeasible for the ILP problem. Hence we still have to "round" z to a solution $x \in F(A, b)$ (but having an eye on solution quality). This means it is sufficient to show an inequality of the form

$$\text{val}(x) \leq c \cdot \text{val}(z).$$

Then the resulting algorithm $\text{ALG}(I) = \text{val}(x)$ is c-approximate because

$$\text{ALG}(I) = \text{val}(x) \leq c \cdot \text{val}(z) \leq c \cdot \text{val}(x^*) = c \cdot \text{OPT}(I).$$

The basic setting for *randomized rounding* is this: Suppose we have the requirements $x_j \in \{0, 1\}$ (instead of $x_j \in \mathbb{Z}$) for $j = 1, \ldots, n$. Suppose we replace each constraint $x_j \in \{0, 1\}$ by $x_j \in [0, 1]$. Then the values $z_j \in [0, 1]$ of any solution z can be *interpreted as probabilities*. Thus we do the following: We randomly produce a vector $X \in \{0, 1\}^n$, where $X_j = 1$ with probability z_j and $X_j = 0$ otherwise. By solving the obtained LP with solution z^*, say, we then have a randomized algorithm ALG with

$$\mathbb{E}\left[\text{ALG}(I)\right] = \mathbb{E}\left[\sum_{j=1}^{n} c_j X_j\right] = \sum_{j=1}^{n} c_j \mathbb{E}\left[X_j\right] = \sum_{j=1}^{n} c_j z_j^* = \text{val}(z^*).$$

However, we still have to ensure that the solution X is also in the feasible region $F(A, b)$ (with a certain probability). Sometimes this task is easy, sometimes not. It depends on the problem at hand.

5.2 Integer Linear Programs

Here we give a general approach for rounding a Linear Program (LP) relaxation of an Integer Linear Program (ILP). We especially obtain bounds on the infeasibility of the obtained solution for the ILP. The omitted proof is a simple application of a bound on concentration of measure, e.g., Azuma-Hoeffding.

Theorem 5.1. *Let $A \in [-\alpha, \alpha]^{n,n}$ be a matrix and let $b \in \mathbb{R}^n$. Let z be any fractional solution for the linear program*

$$\begin{aligned} Ax &= b, \\ x &\in [0, 1]^n. \end{aligned}$$

Define a vector $X \in \{0, 1\}^n$ randomly by

$$X_i = \begin{cases} 1 & \text{with probability } z_i, \\ 0 & \text{otherwise} \end{cases} \qquad \text{for all } i = 1, \ldots, n.$$

Then, with high probability, an outcome x of X satisfies

$$Ax - b \in [-c\sqrt{n \log n}, c\sqrt{n \log n}]^n,$$

for a suitable constant $c = c(\alpha) > 0$.

5.3 Minimum Set Cover

The Minimum Set Cover problem is a simple to state – yet quite general – NP-hard problem. It is widely applicable in sometimes unexpected ways. The problem is the following: We are given a set U (called *universe*) of n elements, a collection of sets $\mathcal{S} = \{S_1, \ldots, S_k\}$ where $S_i \subseteq U$ and $\cup_{S \in \mathcal{S}} S = U$, and a cost function $c : \mathcal{S} \to \mathbb{R}^+$. The task is to find a minimum cost subcollection $\mathcal{S}' \subseteq \mathcal{S}$ that *covers* U, i.e., such that $\cup_{S \in \mathcal{S}'} S = U$.

Example 5.2. Consider this instance: $U = \{1, 2, 3\}$, $\mathcal{S} = \{S_1, S_2, S_3\}$ with $S_1 = \{1, 2\}$, $S_2 = \{2, 3\}$, $S_3 = \{1, 2, 3\}$ and cost $c(S_1) = 10$, $c(S_2) = 50$, and $c(S_3) = 100$. These collections cover U: $\{S_1, S_2\}$, $\{S_3\}$, $\{S_1, S_3\}$, $\{S_2, S_3\}$, $\{S_1, S_2, S_3\}$. The cheapest one is $\{S_1, S_2\}$ with cost equal to 60.

For each set S, we associate a variable $x_S \in \{0, 1\}$ that indicates of we want to choose S or not. We may thus write solutions for Minimum Set Cover as a vector $x \in \{0, 1\}^k$. With this, we write Minimum Set Cover as a mathematical program.

Problem 5.2 Minimum Set Cover

Instance. Universe U with n elements, collection $\mathcal{S} = \{S_1, \ldots, S_k\}$, $S_i \subseteq U$, a cost function $c : \mathcal{S} \to \mathbb{R}$.

Task. Solve the problem

$$\begin{aligned} \text{minimize} \quad & \text{val}(x) = \sum_{S \in \mathcal{S}} c(S) x_S, \\ \text{subject to} \quad & \sum_{S : e \in S} x_S \geq 1 \quad e \in U, \\ & x_S \in \{0, 1\} \quad S \in \mathcal{S}. \end{aligned}$$

Randomized Rounding

We use give a randomized rounding method for this: For example, for the above relaxation, observe that the values z_S are between zero and one. We may thus interpret these values as probabilities for choosing a certain set S.

Here is the idea of the following algorithm: Solve the LP-relaxation optimally and call the solution z. With probability z_S include the set S into the cover.

This basic procedure yields a vector x with expected value equal to the optimal fractional solution value but might not cover all the elements. We thus repeat the procedure "sufficiently many" times and include a set into our cover if it was included in *any* of the iterations. We will show that $O(\log n)$ many iterations suffice for obtaining a feasible cover, thus yielding an $O(\log n)$-approximation algorithm.

Theorem 5.3. Randomized Rounding Set Cover *is* $2\lceil \ln n \rceil$*-approximate for* Minimum Set Cover*, in expectation.*

Proof. Let z be an optimal solution for the LP. In each iteration in Step 2, for each set S we let $x_S = 1$ with probability z_S. Then we have an increase in objective value in each iteration of at most

$$\sum_{S \in \mathcal{S}} \mathbb{E}[c(S) x_S] = \sum_{S \in \mathcal{S}} c(S) \Pr[x_S = 1] = \sum_{S \in \mathcal{S}} c(S) z_S = \text{val}(z).$$

Algorithm 5.1 RANDOMIZED ROUNDING SET COVER

Input. Universe U with n elements, collection $\mathcal{S} = \{S_1, \ldots, S_k\}$, $S_i \subseteq U$, a cost function $c : \mathcal{S} \to \mathbb{R}$.

Output. Vector $x \in \{0,1\}^k$

Step 1. Set $x = 0$, solve the LP relaxation below, and call the optimal solution z.

$$\begin{aligned} \text{minimize} \quad & \text{val}(x) = \sum_{S \in \mathcal{S}} c(S) x_S, \\ \text{subject to} \quad & \sum_{S : e \in S} x_S \geq 1 \quad e \in U, \\ & x_S \geq 0 \quad S \in \mathcal{S}. \end{aligned}$$

Step 2. Repeat $2\lceil \ln n \rceil$ times: For each set S let $x_S = 1$ with probability z_S, leave x_S unchanged otherwise.

Step 3. Return x.

We estimate the probability that an element $u \in U$ is covered in one iteration. Let u be contained in k sets and let $z_1, \ldots, z_k$ be the probabilities given in the solution z. Since u is fractionally covered we have $z_1 + \cdots + z_k \geq 1$. With easy but tedious calculus we see that – under this condition – the probability for u being covered is minimized when the z_i are all equal, i.e., $z_1 = \cdots = z_k = 1/k$:

$$\Pr[x_S = 1] = 1 - (1 - z_1) \cdot \cdots \cdot (1 - z_k) \geq 1 - \left(1 - \frac{1}{k}\right)^k \geq 1 - \frac{1}{e}.$$

Summing up this step: In each of the iterations in Step 2 the value of the solution x constructed increases by at most $\text{val}(z)$ in expectation. Each element is covered with probability at least $1 - 1/e$. But maybe we have not covered all elements after $2\lceil \ln n \rceil$ iterations. Here we show that we will have with high probability.

The probability that element u is not covered at the end of the algorithm, i.e., after $2\lceil \ln n \rceil$ iterations is

$$\Pr[u \text{ is not covered}] \leq \left(\frac{1}{e}\right)^{2\lceil \ln n \rceil} \leq \frac{1}{n^2}.$$

Thus the probability that there is an uncovered element is at most $n/n^2 = 1/n$.

Clearly,

$$\mathbb{E}[\text{val}(x)] \leq 2\lceil \ln n \rceil \cdot \text{val}(z) \leq 2\lceil \ln n \rceil \cdot \text{val}(x^*),$$

where x^* is an optimal solution for MINIMUM SET COVER. So, with high probability, the algorithm returns a feasible solution, whose expected value is at most $2\lceil \ln n \rceil$. □

5.4 Maximum Satisfiability

The SATISFIABILITY problem asks if a certain given Boolean formula has a satisfying assignment, i.e., one that makes the whole formula evaluate to true. There is a related

optimization problem called MAXIMUM SATISFIABILITY. The goal of this chapter is to develop a *deterministic* 3/4-approximation algorithm by first giving a *randomized algorithm*, which will then be *derandomized*.

We are given the Boolean *variables* $X = \{x_1, \ldots, x_n\}$, where each $x_i \in \{0, 1\}$. A *literal* ℓ_i of the variable x_i is either x_i itself, called a *positive* literal, or its negation $\bar{x}_i$ with truth value $1 - x_i$, called a *negative* literal. A *clause* is a disjunction $C = (\ell_1 \vee \cdots \vee \ell_k)$ of literals ℓ_j of X; their number k is called the *size* of C. For a clause C let S_C^+ denote the set of its positive literals; similarly S_C^- the set of its negative literals. Let $\mathcal{C}$ denote the set of clauses. A Boolean formula in *conjunctive form* is a conjunction of clauses $F = C_1 \wedge \cdots \wedge C_m$. Each vector $x \in \{0, 1\}^n$ is called a *truth assignment*. For any clause C and any such assignment x we say that x *satisfies* C if at least one of the literals of C evaluates to 1.

The problem MAXIMUM SATISFIABILITY is the following: We are given a formula F in conjunctive form and for each clause C a weight w_C, i.e., a weight function $w : \mathcal{C} \to \mathbb{N}$. The objective is to find a truth assignment $x \in \{0, 1\}^n$ that maximizes the total weight of the satisfied clauses. As an important special case: If we set all weights w_C equal to one, then we seek to maximize the number of satisfied clauses.

Now we introduce for each clause C a variable $z_C \in \{0, 1\}$ which takes the value one if and only if C is satisfied under a certain truth assignment x. Now we can formulate this as an ILP as done in Problem 5.3.

Problem 5.3 MAXIMUM SATISFIABILITY

Instance. Formula $F = C_1 \wedge \cdots \wedge C_m$ with m clauses over the n Boolean variables $X = \{x_1, \ldots, x_n\}$. A weight function $w : \mathcal{C} \to \mathbb{N}$.

Task. Solve the problem

$$\begin{aligned} \text{maximize} \quad & \text{val}(z) = \sum_{C \in \mathcal{C}} w_C z_C, \\ \text{subject to} \quad & \sum_{i \in S_C^+} x_i + \sum_{i \in S_C^-} (1 - x_i) \geq z_C \quad C \in \mathcal{C}, \\ & z_C \in \{0, 1\} \quad C \in \mathcal{C}, \\ & x_i \in \{0, 1\} \quad i = 1, \ldots, n. \end{aligned}$$

The algorithm we aim for is a combination of two algorithms. One works better for small clauses, the other for large clauses. Both are initially randomized but can be *derandomized* using the method of conditional expectation, i.e., the final algorithm is deterministic.

5.4.1 Randomized Algorithm

For each variable x_i we define the random variable X_i that takes the value one with a certain probability p_i and zero otherwise. This induces, for each clause C, a random variable Z_C that takes the value one if C is satisfied under a (random) assignment and zero otherwise.

Algorithm for Large Clauses

Consider this algorithm RANDOMIZED LARGE: For each variable x_i with $i = 1, \ldots, n$, set $X_i = 1$ independently with probability $1/2$ and $X_i = 0$ otherwise. Output $X = (X_1, \ldots, X_n)$.

Define the quantity

$$\alpha_k = 1 - 2^{-k}.$$

Lemma 5.4. *Let C be a clause. If* $\text{size}(C) = k$ *then*

$$\mathbb{E}[Z_C] = \alpha_k.$$

Proof. A clause C is not satisfied, i.e., $z_C = 0$ if and only if all its literals are set to zero. By independence, the probability of this event is exactly 2^{-k} and thus

$$\mathbb{E}[Z_C] = 1 \cdot \Pr[Z_C = 1] + 0 \cdot \Pr[Z_C = 0] = 1 - 2^{-k} = \alpha_k$$

which was claimed. □

Theorem 5.5. RANDOMIZED LARGE *is a* $1/2$*-approximate for* MAXIMUM SATISFIABILITY*, in expectation.*

Proof. By linearity of expectation, Lemma 5.4, and $\text{size}(C) \geq 1$ we have

$$\mathbb{E}[\text{val}(Z)] = \sum_{C \in \mathcal{C}} w_C \mathbb{E}[Z_C] = \sum_{C \in \mathcal{C}} w_C \alpha_{\text{size}(C)} \geq \frac{1}{2} \sum_{C \in \mathcal{C}} w_C \geq \frac{1}{2} \text{val}(z^*)$$

where (x^*, z^*) is an optimal solution for MAXIMUM SATISFIABILITY. We have used the obvious bound $\text{val}(z^*) \leq \sum_{C \in \mathcal{C}} w_C$. □

Algorithm for Small Clauses

Maybe the most natural LP-relaxation of the problem is:

$$\begin{aligned}
\text{maximize} \quad & \text{val}(z) = \sum_{C \in \mathcal{C}} w_C z_C, \\
\text{subject to} \quad & \sum_{i \in S_C^+} x_i + \sum_{i \in S_C^-} (1 - x_i) \geq z_C \quad C \in \mathcal{C}, \\
& 0 \leq z_C \leq 1 \quad C \in \mathcal{C} \\
& 0 \leq x_i \leq 1 \quad i = 1, \ldots, n.
\end{aligned}$$

In the sequel let (x, z) denote an optimum solution for this LP.

Consider this algorithm RANDOMIZED SMALL: Determine (x, z). For each variable x_i with $i = 1, \ldots, n$, set $X_i = 1$ independently with probability x_i and $X_i = 0$ otherwise. Output $X = (X_1, \ldots, X_n)$.

Define the quantity

$$\beta_k = 1 - \left(1 - \frac{1}{k}\right)^k.$$

Lemma 5.6. *Let C be a clause. If* $\text{size}(C) = k$ *then*

$$\mathbb{E}[Z_C] = \beta_k z_C.$$

Proof. We may assume that the clause C has the form $C = (x_1 \vee \cdots \vee x_k)$; otherwise rename the variables and rewrite the LP.

The clause C is satisfied if $X_1, \ldots, X_k$ are not all set to zero. The probability of this event is

$$\begin{aligned} 1 - \Pi_{i=1}^{k}(1 - x_i) &\geq 1 - \left(\frac{\sum_{i=1}^{k}(1 - x_i)}{k}\right)^k \\ &= 1 - \left(1 - \frac{\sum_{i=1}^{k} x_i}{k}\right)^k \\ &\geq 1 - \left(1 - \frac{z_C}{k}\right)^k. \end{aligned}$$

Above we firstly have used the arithmetic-geometric mean inequality, which states that for non-negative numbers $a_1, \ldots, a_k$ we have

$$\frac{a_1 + \cdots + a_k}{k} \geq \sqrt[k]{a_1 \cdot \cdots \cdot a_k}.$$

Secondly the LP guarantees the inequality $x_1 + \cdots + x_k \geq z_C$.

Now define the function $g(t) = 1 - (1 - t/k)^k$. This function is concave with $g(0) = 0$ and $g(1) = 1 - (1 - 1/k)^k$ which yields that we can bound

$$g(t) \geq t(1 - (1 - 1/k)^k) = t\beta_k$$

for all $t \in [0, 1]$.

Therefore

$$\Pr[Z_C = 1] \geq 1 - \left(1 - \frac{z_C}{k}\right)^k \geq \beta_k z_C$$

and the claim follows. □

Theorem 5.7. Randomized Small *is* $1 - 1/e$*-approximate for* Maximum Satisfiability*, in expectation.*

Proof. The function β_k is decreasing with k. Therefore if all clauses are of size at most k, then by Lemma 5.6

$$\mathbb{E}[\mathrm{val}(Z)] = \sum_{C \in \mathcal{C}} w_C \mathbb{E}[Z_C] \geq \beta_k \sum_{C \in \mathcal{C}} w_C z_C = \beta_k \mathrm{val}(z) \geq \beta_k \mathrm{val}(z^*),$$

where (x^*, z^*) is an optimal solution for Maximum Satisfiability. The claim follows since $(1 - 1/k)^k > 1/e$ for all $k \in \mathbb{N}$. □

3/4-Approximation Algorithm

Consider the algorithm Randomized Combine: With probability 1/2 run Randomized Large otherwise run Randomized Small.

Lemma 5.8. *Let C be a clause, then*

$$\mathbb{E}[Z_C] = \frac{3}{4} \cdot z_C.$$

Proof. Let the random variable B take the value zero if the first algorithm is run, one otherwise. For a clause C let size$(C) = k$. By Lemma 5.4 and $z_C \leq 1$

$$\mathbb{E}\left[Z_C \mid B = 0\right] = \alpha_k \geq \alpha_k z_C.$$

and by Lemma 5.4

$$\mathbb{E}\left[Z_C \mid B = 1\right] \geq \beta_k z_C.$$

Combining we have

$$\mathbb{E}\left[Z_C\right] = \mathbb{E}\left[Z_C \mid B = 0\right]\Pr\left[B = 0\right] + \mathbb{E}\left[Z_C \mid B = 1\right]\Pr\left[B = 1\right] \geq \frac{z_C}{2}(\alpha_k + \beta_k).$$

It is tedious but relatively easy to see that $\alpha_k + \beta_k \geq 3/2$ for all $k \in \mathbb{N}$. □

Theorem 5.9. Randomized Combine *is* $3/4$*-approximate for* Maximum Satisfiability*, in expectation.*

Proof. This follows from Lemma 5.8 and linearity of expectation. □

5.4.2 Derandomization

The notion of *derandomization* refers to "turning" a randomized algorithm into a deterministic one (possibly at the cost of additional running time or deterioration of approximation guarantee). One of the several available techniques is the method of *conditional expectation*.

We are given a Boolean formula $F = C_1 \wedge \cdots \wedge C_m$ in conjunctive form over the variables $X = \{x_1, \ldots, x_n\}$. Suppose we set $x_1 = 0$, then we get a formula F_0 over the variables $x_2, \ldots, x_n$ after simplification; if we set $x_1 = 1$ then we get a formula F_1.

Example 5.10. Let $F = (x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_3) \wedge (x_1 \vee \bar{x}_4)$ where $X = \{x_1, \ldots, x_4\}$.

$$\begin{aligned} x_1 = 0: &\quad F_0 = (x_2) \wedge (x_4) \\ x_1 = 1: &\quad F_1 = (x_3) \end{aligned}$$

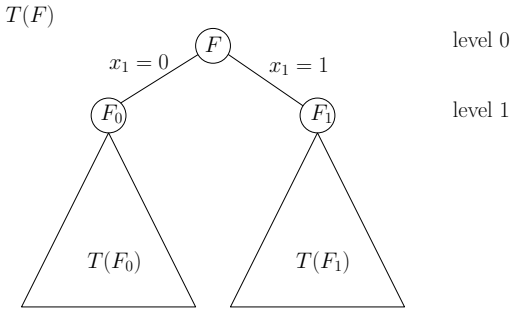

Figure 5.1: Derandomization tree for a formula F.

Applying this recursively, we obtain the tree $T(F)$ depicted in Figure 5.1. The tree $T(F)$ is a complete binary tree with height $n+1$ and $2^{n+1}-1$ vertices. Each vertex at level i corresponds to a setting for the Boolean variables $x_1, \ldots, x_i$. We label the vertices of $T(F)$ with their respective conditional expectations as follows. Let $X_1 = a_1, \ldots, X_i = a_i \in \{0, 1\}$

be the outcome of a truth assignment for the variables $x_1, \ldots, x_i$. The vertex corresponding to this assignment will be labeled

$$\mathbb{E}\left[\text{val}(Z) \mid X_1 = a_1, \ldots, X_i = a_i\right].$$

If $i = n$, then this conditional expectation is simply the total weight of clauses satisfied by the truth assignment $x_1 = a_1, \ldots, x_n = a_n$.

The goal of the remainder of the section is to show that we can find deterministically in polynomial time a path from the root of $T(F)$ to a leaf such that the conditional expectations of the vertices on that path are at least as large as $\mathbb{E}\left[\text{val}(Z)\right]$. Obviously, this property yields the desired: We can construct determistically a solution which is at least as good as the one of the randomized algorithm in expectation.

Lemma 5.11. *The conditional expectation*

$$\mathbb{E}\left[\text{val}(Z) \mid X_1 = a_1, \ldots, X_i = a_i\right]$$

of any vertex in $T(F)$ can be computed in polynomial time.

Proof. Consider a vertex $X_1 = a_1, \ldots, X_i = a_i$. Let F' be the Boolean formula obtained from F by setting $x_1, \ldots, x_i$ accordingly. F' is in the variables $x_{i+1}, \ldots, x_n$.

Clearly, by linearity of expectation, the expected weight of any clause of F' under any random truth assignment to the variables $x_{i+1}, \ldots, x_n$ can be computed in polynomial time. Adding to this the total weight of clauses satisfied by $x_1, \ldots, x_i$ gives the answer. □

Theorem 5.12. *We can compute in polynomial time a path from the root to a leaf in $T(F)$ such that the conditional expectation of each vertex on this path is at least $\mathbb{E}\left[\text{val}(Z)\right]$.*

Proof. Consider the conditional expectation at a certain vertex $X_1 = a_1, \ldots, X_i = a_i$ for setting the next variable X_{i+1}. We have that

$$\begin{aligned}
&\mathbb{E}\left[\text{val}(Z) \mid X_1 = a_1, \ldots, X_i = a_i\right] \\
&\qquad = \mathbb{E}\left[\text{val}(Z) \mid X_1 = a_1, \ldots, X_i = a_i, X_{i+1} = 0\right] \Pr\left[X_{i+1} = 0\right] \\
&\qquad\qquad + \mathbb{E}\left[\text{val}(Z) \mid X_1 = a_1, \ldots, X_i = a_i, X_{i+1} = 1\right] \Pr\left[X_{i+1} = 1\right].
\end{aligned}$$

We show that the two conditional expectations with X_{i+1} can *not* be both strictly smaller than $\mathbb{E}\left[\text{val}(Z) \mid X_1 = a_1, \ldots, X_i = a_i\right]$. Assume the contrary, then we have

$$\begin{aligned}
&\mathbb{E}\left[\text{val}(Z) \mid X_1 = a_1, \ldots, X_i = a_i\right] \\
&\qquad < \mathbb{E}\left[\text{val}(Z) \mid X_1 = a_1, \ldots, X_i = a_i\right] \left(\Pr\left[X_{i+1} = 0\right] + \Pr\left[X_{i+1} = 1\right]\right)
\end{aligned}$$

which is a contradiction since $\Pr\left[X_{i+1} = 0\right] + \Pr\left[X_{i+1} = 1\right] = 1$.

This yields the existence of such a path. And by Lemma 5.11 it can be computed in polynomial time. □

The derandomized version of a randomized algorithm now simply executes these proofs with the probability distribution as given by the randomized algorithm.

5.5 Plant Location

The Plant Location means the following problem: There are m possible plants and n customers. Customer j, say, has a *demand* of d_j units of some utility. Each plant i can be *constructed* at cost c_i. If a constructed plant i *serves* a customer j, it produces $u_{i,j}$ units of the *utility* at *service cost* $s_{i,j}$. The goal is to serve the demand of each customer at minimal total cost.

Plant Location is a prominent problem in operations research and has numerous applications since many economic decisions involve selecting and placing facilities to serve certain demands. Examples include manufacturing plants, storage facilities, depots, warehouses, libraries, fire stations and so on.

The Plant Location problem is NP-hard and can thus not be solved optimally in polynomial time, unless P = NP. Instead, we are interested in approximation algorithms. Recall that an algorithm is called *ρ-approximation* if the objective value of its returned solution is always within ρ times the optimal value.

We give a randomized expected $(4+\varepsilon)\cdot \ln n$-approximation algorithm for any $\varepsilon > 0$. The two main technical ingredients of our algorithm are:

(I) We give an LP-relaxation of the Plant Location problem strengthened by Knapsack Cover valid inequalities. This strengthened formulation allows us to round a fractional solution randomly thus yielding logarithmic integrality gap.

(II) The second tool we use (for the analysis of the rounding) is Bernstein's inequality, which is a Chernoff-type bound on the distribution of sums of independent random variables. A property of this bound is that it depends on the variance of the random sum under investigation and that the absolute values of the summands can be arbitrary constants. The dependence on the variance is useful: We use the Knapsack Cover inequalities to derive a rounding scheme in which some "large" variables are rounded deterministically and the other "small" ones are rounded randomly. For the latter we can prove that the variance of the associated sums is not "too large" and we derive a sufficiently strong bound on the concentration of measure.

We obtain the algorithm as follows: In Section 5.5.1 we consider the case $n = 1$, in which the Plant Location problem becomes the Knapsack Cover problem. As a first step, we give a randomized rounding algorithm for this special case. In Section 5.5.2, we observe that the Plant Location problem can be seen as n simultaneous Knapsack Cover problems with the additional requirement of constructing plants. By solving all of these problems feasibly with high probability we obtain which plant shall serve which customer. Having these decisions made, we construct a plant if it serves some customer. With the help of the ingredients (I) and (II) we are able to prove that the algorithm is expected $(4+\varepsilon)\cdot \ln n$-approximate.

The possible plants are indexed by the set $P = \{1,\dots,m\}$ and the customers are indexed by $C = \{1,\dots,n\}$. We are given a *demand vector* $d = (d_1,\dots,d_j,\dots,d_n)$ with $d_j \geq 0$ for all $j \in C$. Furthermore, we are given a *cost vector* $c = (c_1,\dots,c_i,\dots,c_m)$ and *constructing* a plant $i \in P$ incurs cost $c_i \geq 0$. For each plant $i \in P$ there is a *service cost vector* $s_i = (s_{i,1},\dots,s_{i,j},\dots,s_{i,n})$. A constructed plant i can *serve* a customer j at cost $s_{i,j}$. Each plant $i \in P$ is associated a *utility vector* $u_i = (u_{i,1},\dots,u_{i,j},\dots,u_{i,n})$ with $u_{i,j} \geq 0$ for all $j \in C$. We may assume that $\sum_{i\in P} u_{i,j} \geq d_j$ for all $j \in C$, since the problem is otherwise clearly infeasible. If plant i serves customer j, $u_{i,j}$ units of the demanded utility are produced.

Our goal is to decide which plants to construct and which plants shall serve which customer in order to cover the demand at minimal total cost. More precisely, introducing the variables $x_i \in \{0,1\}$ of a vector $x = (x_1, \ldots, x_i, \ldots, x_m)$ that indicate if plant i is constructed and the variables $y_{i,j} \in \{0,1\}$ of a vector $y = (y_{1,1}, \ldots, y_{i,j}, \ldots, y_{m,n})$ that indicate if plant i serves customer j, we define PLANT LOCATION:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{m} c_i x_i + \sum_{i=1}^{m} \sum_{j=1}^{n} s_{i,j} y_{i,j} && (5.1)\\
\text{subject to} \quad & \sum_{i=1}^{m} u_{i,j} y_{i,j} \geq d_j \qquad j = 1, \ldots, n, && (5.2)\\
& x_i - y_{i,j} \geq 0 \qquad i = 1, \ldots, m, j = 1, \ldots, n, && (5.3)\\
& x_i \in \{0,1\} \qquad i = 1, \ldots, m, && (5.4)\\
& y_{i,j} \in \{0,1\} \qquad i = 1, \ldots, m, j = 1, \ldots, n. && (5.5)
\end{aligned}$$

As a shorthand we write $\text{cost}(x,y) = \sum_{i=1}^{m} c_i x_i + \sum_{i=1}^{m} \sum_{j=1}^{n} s_{i,j} y_{i,j}$.

5.5.1 Knapsack Cover

Using similar notation, in the KNAPSACK COVER problem we are given a knapsack with *demand* $d \geq 0$ and items $P = \{1, \ldots, m\}$. Item $i \in P$ has *utility* $u_i \geq 0$ and *cost* $c_i \geq 0$. Find a subset of items with total utility covering the demand at minimal total cost. That is, KNAPSACK COVER is the problem:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{m} c_i x_i && (5.6)\\
\text{subject to} \quad & \sum_{i=1}^{m} u_i x_i \geq d, && (5.7)\\
& x_i \in \{0,1\} \qquad i = 1, \ldots, m. && (5.8)
\end{aligned}$$

We write $\text{cost}(x) = \sum_{i=1}^{m} c_i x_i$. Using relaxed variables $x_i \in [0,1]$ instead of the $x_i \in \{0,1\}$ yields an arbitrarily large integrality gap. For any set $Q \subseteq P$ define

$$d(Q) = d - \sum_{i \in Q} u_i \quad \text{and} \quad u_i(Q) = \min\{u_i, d(Q)\}$$

as the *residual demand* $d(Q)$ and *residual utility* $u_i(Q)$, respectively. An alternative formulation with bounded integrality gap is:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{m} c_i x_i && (5.9)\\
\text{subject to} \quad & \sum_{i \in P-Q} u_i(Q) x_i \geq d(Q) \qquad \text{for all } Q \subseteq P, && (5.10)\\
& x_i \in [0,1] \qquad i = 1, \ldots, m. && (5.11)
\end{aligned}$$

The constraint $\sum_{i \in P-Q} u_i(Q) x_i \geq d(Q)$ states that, even if we choose the items in Q, the remaining items $P - Q$ must still cover the residual demand $d(Q)$. It can be proved

that the integrality gap of this relaxation is 2. We are not aware of an algorithm that solves this relaxation exactly. However, define the following type of solution, which is sufficient for our purpose and which can be found in polynomial time with the ELLIPSOID algorithm. For $c > 0$, call a vector $\bar{x}$ a *c-relaxed solution* for (5.9) if $\text{cost}(\bar{x}) \leq \text{cost}(x)$, where x is an optimal (fractional) solution for (5.9), and $\bar{x}$ satisfies (5.11) and the KNAPSACK COVER inequalities (5.10) for the set $Q = \{i \in P : \bar{x}_i \geq c\}$. A c-relaxed solution can be found in polynomial time, because the separation problem of the ELLIPSOID algorithm can be solved in polynomial time: Given a solution candidate $\bar{x}$ already satisfying (5.11), we can compute the set Q in linear time and check if (5.10) is also satisfied for this particular set. If so, we can stop, if not, we have found a violated constraint, as needed by the ELLIPSOID algorithm.

Algorithm 5.2 ROUNDKC

Input. $c \in (0, 1]$.

Output. $X = (X_1, \ldots, X_m) \in \{0, 1\}^m$.

Step 1. Find a c-relaxed solution for (5.9) $\bar{x} = (\bar{x}_1, \ldots, \bar{x}_m) \in [0, 1]^m$.

Step 2. For $i = 1, \ldots, m$ let $C_i \sim \text{Uni}\,(0, c)$ and let

$$X_i = \begin{cases} 1 & \text{if } \bar{x}_i \geq C_i, \\ 0 & \text{otherwise.} \end{cases}$$

Step 3. Return $X = (X_1, \ldots, X_m)$.

Theorem 5.13. *Let $c \in (0, 1]$. The algorithm* ROUNDKC(c) *runs in polynomial time and returns an expected $1/c$-approximate solution, which is feasible for* KNAPSACK COVER *with probability at least $1 - 2\exp(-(1-c)^2/4c)$.*

The proof of Theorem 5.13 can be found below and uses the following intermediate result.

Lemma 5.14. *Let $c \in (0, 1]$. With probability at least $1-2\exp(-(1-c)^2/4c)$,* ROUNDKC(c) *returns a feasible solution for* KNAPSACK COVER*.*

Proof. The proof uses Bernstein's inequality.

Theorem 5.15 (Bernstein)**.** *Let $Z_1, \ldots, Z_m$ be independent random variables with $\mathbb{E}[Z_i] = 0$ and $|Z_i| \leq \delta$. For $Z = \sum_{i=1}^m Z_i$ and $\lambda > 0$ it holds that*

$$\Pr[|Z| > \lambda] \leq 2\exp\left(-\frac{1}{2}\frac{\lambda^2}{\text{Var}[Z] + \lambda\delta}\right).$$

We have to show that $U = \sum_{i=1}^m u_i X_i \geq d$ with probability at least $1 - 2\exp(-(1 - 1/c)^2/4c)$, or equivalently $\Pr[U < d] \leq 2\exp(-(1 - 1/c)^2/4c)$. First observe that the items $i \in Q \subseteq P$ with $\bar{x}_i \geq c$ are rounded to one deterministically, i.e., $X_i = 1$. Now, if $\sum_{i \in Q} u_i X_i \geq d$, there is nothing to show as the solution is already feasible. Thus we assume $\sum_{i \in Q} u_i X_i < d$. For the set Q, since $\bar{x}$ is c-relaxed, we have the constraint

$$\sum_{i \in P - Q} u_i(Q)\bar{x}_i \geq d(Q).$$

For the items $i \in P - Q$ with $\bar{x}_i \leq c$ we have

$$\mathbb{E}[X_i] = \Pr[X_i = 1] = \Pr[\bar{x}_i \geq C_i] = \bar{x}_i/c.$$

Therefore, by defining $U(Q) = \sum_{i \in P-Q} u_i(Q) X_i$ we have

$$\mathbb{E}[U(Q)] = \sum_{i \in P-Q} u_i(Q)\mathbb{E}[X_i] = \frac{1}{c} \cdot \sum_{i \in P-Q} u_i(Q)\bar{x}_i \geq \frac{1}{c} \cdot d(Q).$$

Define $Z_i = u_i(Q)(X_i - \mathbb{E}[X_i])$ for all $i \in P - Q$ and $Z(Q) = \sum_{i \in P-Q} Z_i = U(Q) - \mathbb{E}[U(Q)]$. Observe that the Z_i are independent, $\mathbb{E}[Z_i] = 0$, and $|Z_i| \leq \max_{i \in P-Q} u_i(Q) =: \delta$. Furthermore, by independence, we have $\operatorname{Var}[Z(Q)] = \operatorname{Var}[U(Q)] = \sum_{i \in P-Q} u_i^2(Q)\operatorname{Var}[X_i] \leq \delta \cdot \mathbb{E}[U(Q)]$, since $\operatorname{Var}[X_i] = 1/c \cdot \bar{x}_i(1 - 1/c \cdot \bar{x}_i) \leq 1/c \cdot \bar{x}_i = \mathbb{E}[X_i]$. Using $\mathbb{E}[U(Q)] \geq 1/c \cdot d(Q) \geq 1/c \cdot \delta$ and Bernstein's inequality yields

$$\begin{aligned}
\Pr[U(Q) < d(Q)] &\leq \Pr[U(Q) < c \cdot \mathbb{E}[U(Q)]] = \Pr[\mathbb{E}[U(Q)] - U(Q) > \mathbb{E}[U(Q)](1-c)] \\
&\leq \Pr[|Z(Q)| > \mathbb{E}[U(Q)](1-c)] \\
&\leq 2\exp\left(-\frac{1}{2}\frac{\mathbb{E}[U(Q)]^2(1-c)^2}{\operatorname{Var}[Z(Q)] + \mathbb{E}[U(Q)](1-c)\delta}\right) \\
&\leq 2\exp\left(-\frac{(1-c)^2}{4c}\right).
\end{aligned}$$

With $X_i = 1$ for $i \in Q$ we find

$$\begin{aligned}
\Pr[U < d] &= \Pr\left[\sum_{i \in P-Q} u_i X_i < d - \sum_{i \in Q} u_i\right] \leq \Pr\left[\sum_{i \in P-Q} u_i(Q) X_i < d - \sum_{i \in Q} u_i\right] \\
&= \Pr[U(Q) < d(Q)] \leq 2\exp\left(-\frac{(1-c)^2}{4c}\right)
\end{aligned}$$

completing the proof. □

Proof of Theorem 5.13. Using the Ellipsoid algorithm for solving the fractional relaxation of Knapsack Cover, RoundKC(c) runs in polynomial time. The probability that X is feasible is stated in Lemma 5.14. Let $\bar{x}$ be the c-relaxed solution found by the algorithm and let x^* be an optimal solution for Knapsack Cover. We clearly have $\text{cost}(\bar{x}) \leq \text{cost}(x^*)$.

Let $Q = \{i \in P : \bar{x}_i \geq c\}$. Recall that variables X_i with $i \in Q$ are rounded to one deterministically. Hence $X_i = 1 = c/c \leq \bar{x}_i/c$. For the variables X_i with $i \in P - Q$ we have $X_i = 1$ with probability equal to $\bar{x}_i/c$. Therefore, for any X found by the algorithm we have

$$\begin{aligned}
\mathbb{E}[\text{cost}(X)] &= \mathbb{E}\left[\sum_{i=1}^{m} c_i X_i\right] = \sum_{i \in Q} c_i \mathbb{E}[X_i] + \sum_{i \in P-Q} c_i \mathbb{E}[X_i] \leq \sum_{i \in Q} \frac{1}{c} \cdot c_i \bar{x}_i + \sum_{i \in P-Q} \frac{1}{c} \cdot c_i \bar{x}_i \\
&= \frac{1}{c} \cdot \text{cost}(\bar{x}) \leq \frac{1}{c} \cdot \text{cost}(x^*)
\end{aligned}$$

as claimed. □

5.5.2 General Case

Here we give an LP relaxation of Plant Location, which also uses Knapsack Cover inequalities. For any $Q \subseteq P$ and customer j define

$$d_j(Q) = d_j - \sum_{i \in Q} u_{i,j} \quad \text{and} \quad u_{i,j}(Q) = \min\{u_{i,j}, d_j(Q)\}$$

as the *residual demand* $d_j(Q)$ and *residual utility* $u_{i,j}(Q)$ for customer j, respectively. This yields the following relaxation for Generalized Plant Location:

$$\text{minimize} \quad \sum_{i=1}^{m} c_i x_i + \sum_{i=1}^{m} \sum_{j=1}^{n} s_{i,j} y_{i,j} \tag{5.12}$$

$$\text{subject to} \quad \sum_{i=1}^{m} u_{i,j}(Q) y_{i,j} \geq d_j(Q) \qquad \text{for all } Q \subseteq P, j = 1, \ldots, n, \tag{5.13}$$

$$x_i - y_{i,j} \geq 0 \qquad i = 1, \ldots, m, j = 1, \ldots, n, \tag{5.14}$$

$$x_i \in [0,1] \qquad i = 1, \ldots, m, \tag{5.15}$$

$$y_{i,j} \in [0,1] \qquad i = 1, \ldots, m, j = 1, \ldots, n. \tag{5.16}$$

Consider the algorithm RoundPL. It is important to note that it uses the random variables $C_1, \ldots, C_m$, i.e., *one* variable per potential plant for the *whole* set of customers. For $c > 0$, call a vector $(\bar{x}, \bar{y})$ a *c-relaxed solution* for (5.12) if $\text{cost}(\bar{x}, \bar{y}) \leq \text{cost}(x, y)$, where (x, y) is a (fractional) optimum solution for (5.12), and $(\bar{x}, \bar{y})$ satisfies (5.14), (5.15), (5.16), and for each j the Knapsack Cover inequalities (5.13) for the set $Q = \{i \in P : \bar{y}_{i,j} \geq c\}$. A c-relaxed solution can be found in polynomial time with the Ellipsoid algorithm, see Section 5.5.1.

Algorithm 5.3 RoundPL

Input. $c \in (0, 1]$.

Output. $X = (X_1, \ldots, X_m) \in \{0,1\}^m$, $Y = (Y_{1,1}, \ldots, Y_{m,n}) \in \{0,1\}^{m \times n}$.

Step 1. Find a c-relaxed solution for (5.12)

$$\bar{x} = (\bar{x}_1, \ldots, \bar{x}_m) \in [0,1]^m, \bar{y} = (\bar{y}_{1,1}, \ldots, \bar{y}_{m,n}) \in [0,1]^{m \times n}.$$

Step 2. For $i = 1, \ldots, m$ let $C_i \sim \text{Uni}\,(0, c)$.

Step 3. For $i = 1, \ldots, m$ and $j = 1, \ldots, n$ let

$$Y_{i,j} = \begin{cases} 1 & \text{if } \bar{y}_{i,j} \geq C_i, \\ 0 & \text{otherwise.} \end{cases}$$

Step 4. For $i = 1, \ldots, m$ let $X_i = \max_{j=1,\ldots,n} Y_{i,j}$.

Step 5. Return $X = (X_1, \ldots, X_m)$, $Y = (Y_{1,1}, \ldots, Y_{m,n})$.

Theorem 5.16. *Let $c \in (0,1]$. The algorithm* RoundPL(c) *runs in polynomial time and returns an expected $1/c$-approximate solution, which is feasible for* Generalized Plant Location *with probability at least $1 - 2n\exp(-(1-c)^2/4c)$.*

Proof. By ignoring the x_i for the moment consider the following problem:

$$\text{minimize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} s_{i,j}y_{i,j} \tag{5.17}$$

$$\text{subject to} \quad \sum_{i=1}^{m} u_{i,j}(Q)y_{i,j} \geq d_j(Q) \qquad \text{for all } Q \subseteq P, j = 1,\ldots,n, \tag{5.18}$$

$$y_{i,j} \in \{0,1\} \qquad i = 1,\ldots,m, j = 1,\ldots,n. \tag{5.19}$$

Observe that this problem consists of n many Knapsack Cover problems – one for each customer. Let $\bar{y}$ be a c-relaxed solution over the variables $\bar{y}_{i,j} \in [0,1]$. Let $C_i \sim \text{Uni}\,(0,c)$ and round the $Y_{i,j}$ as given in the algorithm.

Let the variable $I_j \in \{0,1\}$ indicate if the variables $Y_{i,j}$ with $i \in P$ are *infeasible* for the Knapsack Cover problem of customer j. Thus $I = \sum_{j=1}^{n} I_j$ counts the number of infeasibly solved problems. We have $\Pr[I > 0] \leq \mathbb{E}[I] = \sum_{j=1}^{n}\mathbb{E}[I_j] = \sum_{j=1}^{n}\Pr[I_j = 1] \leq 2n\exp(-(1-c)^2/4c)$.

For each *fixed* customer j, the $Y_{i,j}$ are independent by construction. Thus Lemma 5.14 also applies to each individual Knapsack Cover problem and we have the bound $\Pr[I_j = 1] \leq 2\exp(-(1-c)^2/4c)$. As there are n such problems, the union bound gives that the vector Y is feasible for the whole problem with probability at least $1 - 2n\exp(-(1-c)^2/4c)$.

For any fixed j let $Q = \{i \in P : \bar{y}_{i,j} \geq c\}$. Thus the $Y_{i,j} = 1 = c/c \leq \bar{y}_{i,j}/c$ deterministically for $i \in Q$ and $Y_{i,j} = 1$ with probability $\bar{y}_{i,j}/c$ for $i \in P - Q$. Hence $\mathbb{E}[\sum_{i=1}^{m} s_{i,j}Y_{i,j}] = \sum_{i=1}^{m} s_{i,j}\mathbb{E}[Y_{i,j}] \leq \frac{1}{c}\cdot\sum_{i=1}^{m} s_{i,j}\bar{y}_{i,j}$.

Now return to the formulation of the problem including the x_i. Let $(\bar{x},\bar{y})$ be the c-relaxed solution found by RoundPL(c). Let $\bar{y}_{i,\max} = \max_{j=1,\ldots,n}\bar{y}_{i,j}$. Since there is *one* random variable C_i per plant i, we have

$$\mathbb{E}[X_i] = \Pr[X_i = 1] = \Pr[\bar{y}_{i,\max} \geq C_i] = \begin{cases} 1 \leq \bar{y}_{i,\max}/c & \text{if } \bar{y}_{i,\max} > c, \\ \bar{y}_{i,\max}/c & \text{if } \bar{y}_{i,\max} \leq c. \end{cases}$$

By the constraints $x_i - y_{i,j} \geq 0$ we have $\bar{y}_{i,\max} \leq \bar{x}_i$.

Let (x^*,y^*) be an optimum solution for the Generalized Plant Location problem. We have that $\text{cost}(\bar{x},\bar{y}) \leq \text{cost}(x^*,y^*)$. Now we calculate and obtain

$$\begin{aligned}\mathbb{E}[\text{cost}(X,Y)] &= \mathbb{E}\left[\sum_{i=1}^{m} c_iX_i + \sum_{i=1}^{m}\sum_{j=1}^{n} s_{i,j}Y_{i,j}\right] = \sum_{i=1}^{m} c_i\mathbb{E}[X_i] + \sum_{j=1}^{n}\mathbb{E}\left[\sum_{i=1}^{m} s_{i,j}Y_{i,j}\right] \\ &\leq \frac{1}{c}\cdot\sum_{i=1}^{m} c_i\bar{x}_i + \frac{1}{c}\cdot\sum_{j=1}^{n}\sum_{i=1}^{m} s_{i,j}\bar{y}_{i,j} = \frac{1}{c}\cdot\text{cost}(\bar{x},\bar{y}) \leq \frac{1}{c}\cdot\text{cost}(x^*,y^*)\end{aligned}$$

as claimed. □

The notion *high probability* refers to probability converging to one as n tends to infinity.

Corollary 5.17. *Choose $c = 1/((4+\varepsilon)\cdot\ln n)$ for any $\varepsilon > 0$ and* RoundPL(c) *is expected $(4+\varepsilon)\cdot\ln n$-approximate and feasible with high probability.*

Part II

Online Algorithms

Chapter 6

Introduction

In *online computation*, an algorithm receives its input as it proceeds, i.e., not necessarily entirely in the beginning. Any decision the algorithm makes can not be revoked later on. That is, the algorithm must decide under incomplete information about the future.

These situations occur naturally in various circumstances:

(1) Investment decisions, e.g., stock trading.

(2) Caching, e.g., in the area of operating systems.

(3) Scheduling, e.g., when tasks arrive over time.

Nevertheless, the algorithm achieves some objective value. In the area of *competitive analysis*, this value is compared against the *offline optimum*, i.e., the best-possible objective value that can be achieved if the entire input is known initially. This comparison may seem unfair, but it captures the price of having to decide under uncertainty.

The *competitive ratio* (to be defined shortly) is the worst-case ratio between the objective values obtained by the online and the best offline algorithm over all inputs. We say that an online algorithm is ρ-*competitive* if its objective value is at most a factor of ρ away from the offline optimal value. If the online algorithm is randomized, we will consider the expected objective value in comparison to the offline optimum value.

It is natural to assume that the input is chosen by a malicious *adversary*. In order to compensate for the unfairness mentioned above, we will sometimes restrict the power of the adversary; either by reducing the resources of the offline algorithm or by limiting the choice of inputs.

6.1 Ski Rental

A very basic example is the Ski Rental problem: Renting skiing equipment costs 1 Euro per day and buying costs $b > 1$ Euro. Of course, after the equipment is bought, no further costs arise. The online nature of the problem is this: On each day, you will either go skiing or not; and you do not know how many skiing-days are yet to come.

If you knew the number d of skiing days, the best-possible (offline) algorithm Opt is clear: If $d > b$ you buy the equipement initially, otherwise you rent on every skiing-day.

The two most obvious algorithms are probably Rent-Always and Buy-At-Once. The problems with these are: If there are many skiing-days, Rent-Always spends an unbounded factor more than Opt. If there are few skiing-days, Buy-At-Once is also unbounded compared to Opt.

A natural online algorithm is Break-Even: Rent for the first $b-1$ days and buy on the b-th skiing day. The name Break-Even is motivated from the strategy of the algorithm to rent until the renting cost accumulate to the buying cost. This algorithm tries to balance between the cases when Rent-Always and Buy-At-Once behave good, while avoiding the cases when they behave poorly.

Consider the algorithm Alg(k) that buys on the k-th skiing day and observe that we get Buy-At-Once = Alg(1), Break-Even = Alg(b), and Rent-Always = Alg(∞). Also observe that any deterministic algorithm is Alg(k) for some value of k.

Theorem 6.1. Break-Even *is* $2-1/b$*-competitive. There is no deterministic online algorithm that is less than* $2-1/b$*-competitive.*

Proof. Let d denote the number of skiing-days. The optimal cost is

$$c^* = \min\{b, d\}.$$

The cost of algorithm Alg(k) is

$$c_k = \begin{cases} d & \text{if } d < k, \\ b+k-1 & \text{if } d \geq k. \end{cases}$$

The competitive ratio of Alg(k) and Opt is

$$\rho_k = \frac{c_k}{c^*} = \begin{cases} d/\min\{b,d\} & \text{if } d < k, \\ (b+k-1)/\min\{b,d\} & \text{if } d \geq k. \end{cases}$$

We distinguish the following cases:

Case 1. $d < k$**:** If $d \leq b$, then $r_k = 1$. If $d > b$, then $r_k = d/b$. This function is maximized for $d = k-1$. In either case we have

$$\rho_k = \max\{1, (k-1)/b\}.$$

The best (minimizing) choice for the algorithm is any value $k \leq b+1$.

Case 2. $d \geq k$**:** If $d \leq b$ we have $\rho_k = (b+k-1)/d$. This function takes its maximum at $d = k$ yielding $\rho_k = (b+k-1)/k$. If $d > b$ we have $\rho_k = (b+k-1)/b$, which does not depend on d. Hence, in either case

$$\rho_k = \max\{1 + (b-1)/k, 1 + (k-1)/b\}.$$

The minimizing choice for the algorithm is $k = b$.

If we take both cases into account, we see that the unique best choice for the algorithm is $k = b$ yielding a competitive ratio of

$$\rho_b = 2 - 1/b.$$

as claimed. □

A natural idea is to randomize between the algorithms Alg(k). That is, Alg(k) is chosen with probability p_k. Notice that the worst-case competitive ratio ρ_k is the larger, the smaller k is; e.g. the proof above yields $r_1 = b$. This means that p_k should be chosen

the smaller, the smaller k is. It turns out that (essentially) the Geometric distribution with success probability $1/b$ is a good choice.

Define the algorithm Randomized-Break-Even by: Choose algorithm Alg(k) with probability

$$p_k = \begin{cases} \left(\frac{b-1}{b}\right)^{b-k} \cdot \frac{1}{b(1-((b-1)/b)^b)} & \text{if } 1 \le k \le b, \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 6.2. Randomized-Break-Even *is expected* $e/(e-1)$*-competitive.*

Proof. The proof is based on calculating the expected competitive value directly. The following known identities, derived from the Geometric series, are useful. Firstly,

$$\sum_{k=1}^{\ell} (b+k-1) \cdot \left(\frac{b-1}{b}\right)^{(b-k)} = b \cdot \ell \cdot \left(\frac{b-1}{b}\right)^{b-\ell}.$$

and secondly

$$\sum_{k=1}^{\ell} \left(\frac{b-1}{b}\right)^{b-k} = b \cdot \left(\frac{b-1}{b}\right)^{b} \cdot \left(1 - \left(\frac{b}{b-1}\right)^{\ell}\right).$$

Recall that b denotes the buying cost and that d denotes the number of skiing-days. We distinguish two cases.

The first (simpler) case is $d \ge b$: The optimal cost is $c^* = b$ and the expected value of the cost C of the algorithm is

$$\begin{aligned} \mathbb{E}[C] &= \frac{1}{b(1-((b-1)/b)^b)} \cdot \sum_{k=1}^{b} (b+k-1) \cdot \left(\frac{b-1}{b}\right)^{(b-k)} \\ &= \frac{b^2}{b(1-((b-1)/b)^b)} \\ &= \frac{b}{1-((b-1)/b)^b}. \end{aligned}$$

Hence we have an expected competitive ratio of

$$\rho = \frac{1}{1-((b-1)/b)^b}.$$

The second case is $d < b$: The optimal cost is $c^* = d$. For the algorithm we get

$$\begin{aligned}
\mathbb{E}[C] &= \frac{1}{b(1-((b-1)/b)^b)} \cdot \left(\sum_{k=1}^{d-1} (b+k-1) \cdot \left(\frac{b-1}{b}\right)^{(b-k)} + \sum_{k=d}^{b} d \cdot \left(\frac{b-1}{b}\right)^{(b-k)} \right) \\
&= \frac{1}{b(1-((b-1)/b)^b)} \cdot \left(b \cdot (d-1) \cdot \left(\frac{b-1}{b}\right)^{b-d+1} \right. \\
&\quad \left. + d \cdot b \cdot \left(\left(\frac{b-1}{b}\right)^b - 1 \right) - d \cdot b \cdot \left(\frac{b-1}{b}\right)^b \cdot \left(1 - \left(\frac{b}{b-1}\right)^{d-1} \right) \right) \\
&= \frac{1}{1-((b-1)/b)^b} \cdot \left((d-1) \cdot \left(\frac{b-1}{b}\right)^{b-d+1} \right. \\
&\quad \left. + d \cdot \left(\left(\frac{b-1}{b}\right)^b - 1 \right) - d \cdot \left(\frac{b-1}{b}\right)^b \cdot \left(1 - \left(\frac{b}{b-1}\right)^{d-1} \right) \right) \\
&= \frac{1}{1-((b-1)/b)^b} \cdot \left((d-1) \cdot \left(\frac{b-1}{b}\right)^{b-d+1} \right. \\
&\quad \left. + d \cdot \left(\left(\frac{b-1}{b}\right)^b - 1 \right) - d \cdot \left(\left(\frac{b-1}{b}\right)^b - \left(\frac{b-1}{b}\right)^{b-d+1} \right) \right) \\
&= \frac{1}{1-((b-1)/b)^b} \cdot \left((d-1) \cdot \left(\frac{b-1}{b}\right)^{b-d+1} + d \cdot \left(\left(\frac{b-1}{b}\right)^{b-d+1} - 1 \right) \right) \\
&= \frac{1}{1-((b-1)/b)^b} \cdot \left((2d-1) \cdot \left(\frac{b-1}{b}\right)^{b-d+1} - d \right)
\end{aligned}$$

Hence we have an expected competitive ratio of

$$\rho = \frac{1}{1-((b-1)/b)^b} \cdot \left((2-1/d) \cdot \left(\frac{b-1}{b}\right)^{b-d+1} - 1 \right)$$

We use the facts $((b-1)/b)^{b-d+1} < 1$ and $2 - 1/d < 2$ to yield

$$\rho \leq \frac{1}{1-((b-1)/b)^b}.$$

By using that $(1 + x/n)^n \to e^x$ from below, we get that

$$\rho \leq \frac{1}{1-((b-1)/b)^b} \leq \frac{1}{1-1/e} = \frac{e}{e-1} \simeq 1.58$$

as claimed. □

6.2 Competitive Analysis

Recall the notions of combinatorial optimization defined in Section 1.2, especially that the ratio

$$\rho_{\text{ALG}}(I) = \frac{\text{ALG}(I)}{\text{OPT}(I)}.$$

of an algorithm ALG and the optmimum OPT on an input I is called approximation ratio. Let us concentrate on minimization problems here. An algorithm is ρ-approximative if

$$\rho_{\text{ALG}}(I) \leq \rho$$

holds for all inputs I. In this setting, the entire input is given to an algorithm before it starts.

As already mentioned, in *online computation*, the input $I = i_1, i_2, \ldots$ is revealed to an algorithm as it proceeded. Any decision the algorithm takes can not be altered later on. However, the objective value attained by such an online algorithm is compared against the (offline) optimal value. Analogously to the above, we say that an online algorithm ALG is *ρ-competitive* if

$$\rho_{\text{ALG}}(I) = \frac{\text{ALG}(I)}{\text{OPT}(I) + \alpha} \leq \rho$$

holds for all inputs I. It is important to note that ALG is an online algorithm, while OPT is the best-possible offline algorithm. The additive constant α in the bound is introduced for the following technical reasons and must *not* depend on the input: In many online settings, the input can be thought of as unbounded and hence we are interested in the asymptotic behaviour of an online algorithm. Allowing for a fixed additive constant α will make the analysis often much simpler and elegant, while not affecting the asymptotics.

The smallest possible value such that some algorithm ALG is still ρ-competitive is called the *competitive ratio* of the algorithm. Notice that ρ need not be a constant – it is allowed to depend on input parameters of the problem at hand. Two important special cases: If ρ is actually a constant, then we say that the algorithm is (constant) competitive. If ρ grows with the length of the input, then we say that the algorithm is *not* competitive.

If the algorithm is randomized, we simply consider

$$\rho_{\text{ALG}}(I) = \frac{\mathbb{E}\left[\text{ALG}(I)\right]}{\text{OPT}(I) + \alpha} \leq \rho,$$

where the expectation is taken over the random choices of the algorithm and the input is fixed (but unknown to the algorithm) before the algorithm starts. (That is, the input can not depend on the random choices of the algorithm.)

Chapter 7

Online Scheduling

7.1 Load Balancing

In this section, we consider the classical Load Balancing problem. We are given m machines for scheduling, indexed by the set $M = \{1, \ldots, m\}$. There are furthermore given n jobs, indexed by the set $J = \{1, \ldots, n\}$, where job j takes $p_{i,j}$ units of time if scheduled on machine i. Let J_i be the set of jobs scheduled on machine i. Then $\ell_i = \sum_{j \in J_i} p_{i,j}$ is the *load* of machine i. The maximum load $\ell_{\max} = c_{\max} = \max_{i \in M} \ell_i$ is called the *makespan* of the schedule.

In the online-version of the problem that we consider here, we assume that the jobs arrive one-by-one. This means that an algorithm must assign a job to some machine before it is presented the next job. The assignment-decision can not be revoked later.

However, we will derive several constant competitive algorithms. In the first version, we consider identical machines, which means that the machines have the same speed, i.e. $p_{i,j} = p_j$ for all $i \in M$. In the case of related machines, the machines have speeds that scale the lengths of assigned jobs. That is, machine i has speed s_i and the processing time of job j is $p_{i,j} = p_j / s_i$ for any $i \in M$.

Identical Machines

In the special case of *identical machines*, we have that $p_{i,j} = p_j$ for all $i \in M$ and all $j \in J$. Here p_j is called the *length* of job j.

List Scheduling

As a warm-up we consider the following two heuristics for Load Balancing. The List Scheduling algorithm works as follows: Determine any ordering of the job set J, stored in a list L. Starting with all machines empty, determine the machine i with the currently least load and schedule the respective next job j in L on i. The load of i before the assignment of j is called the *starting time* s_j of job j and the load of i after the assignment is called the *completion time* c_j of job j. In the Sorted List Scheduling algorithm we execute List Scheduling, where the list L consists of the jobs in decreasing order of length.

Theorem 7.1. *The* List Scheduling *algorithm is a* 2*-approximation for* Load Balancing *on identical machines.*

Proof. Let T^* be the optimal makespan of the given instance. We show that $s_j \leq T^*$ for all $j \in J$. This implies $c_j = s_j + p_j \leq T^* + p_j \leq 2 \cdot T^*$ for all $j \in J$, since we clearly must have $T^* \geq p_j$ for all $j \in J$.

Assume that $s_j > T^*$ for some $j \in J$. Then we have that the load *before* the assignment of j is $\ell_i > T^*$ for all $i \in M$. Thus the jobs $J' \subseteq J$ scheduled before j by the algorithm have total length $\sum_{j' \in J'} p_{j'} > m \cdot T^*$. On the other hand, since the optimum solution schedules all jobs J until time T^* we have $\sum_{j \in J} p_j \leq m \cdot T^*$. A contradiction and the List Scheduling algorithm must start all jobs not later than time T^*. □

Let us assume the special-case that we know that the jobs are presented in non-increasing order of length. Then, List Scheduling becomes the Sorted List Scheduling algorithm. Here we show that Sorted List Scheduling is a 3/2-approximation. (This is not best-possible, because one can actually prove that the algorithm is a 4/3-approximation.)

Theorem 7.2. *The* Sorted List Scheduling *algorithm is a* 3/2*-approximation for* Load Balancing *on identical machines.*

Proof. Let T^* be the optimal makespan of the given instance. Partition the jobs $J_L = \{j \in J : p_j > T^*/2\}$ and $J_S = J - J_L$, called *large* and *small* jobs. Notice that there can be at most m large jobs: Assume that there are more than m such jobs. Then, in any schedule, including the optimal one, there must be at least two such jobs scheduled on some machine. Since the length of a large job is more than $T^*/2$, this contradicts that T^* is the optimal makespan.

Since there are at most m large jobs and the algorithm schedules those first and hence on individual machines, we have that each large job completes not later than T^*, i.e., $c_j \leq T^*$ for all $j \in J_L$. Thus, if a job completes later than T^* it must be a small job having length at most $T^*/2$. Since each job starts not later than T^* we have $c_j \leq T^* + p_j \leq 3/2 \cdot T^*$ for every small job $j \in J_S$. □

7.2 Bin Packing

Here we consider the classical Bin Packing problem: We are given a set $I = \{1, \ldots, n\}$ of *items*, where item $i \in I$ has *size* $s_i \in (0, 1]$ and a set $B = \{1, \ldots, n\}$ of *bins* with *capacity* one. Find an assignment $a : I \to B$ such that the number of non-empty bins is minimal. As a shorthand, we write $s(J) = \sum_{j \in J} s_j$ for any $J \subseteq I$.

We will show that there are constant competitive approximations for Bin Packing. Firstly we consider the probably most simple Next Fit algorithm, which can be shown to be 2-competitive. Secondly, we give the First Fit Decreasing algorithm and show that it is 3/2-competitive. The latter algorithm is online, but requires that the items are given in non-increasing size.

Next Fit

The Next Fit algorithm works as follows: Initially all bins are empty and we start with bin $j = 1$ and item $i = 1$. If bin j has residual capacity for item i, assign item i to bin j, i.e., $a(i) = j$, and consider item $i + 1$. Otherwise consider bin $j + 1$ and item i. Repeat until item n is assigned.

Theorem 7.3. Next Fit *is* 2*-competitive for* Bin Packing*. The algorithm runs in* $O(n)$ *time.*

Proof. Let k be the number of non-empty bins in the assignment a found by Next Fit. Let k^* be the optimal number of bins. We show the slightly stronger statement that

$$k \leq 2 \cdot k^* - 1.$$

Firstly we observe the lower bound $k^* \geq \lceil s(I) \rceil$. Secondly, for bins $j = 1, \ldots, \lfloor k/2 \rfloor$ we have

$$\sum_{i:a(i) \in \{2j-1, 2j\}} s_i > 1.$$

Adding these inequalities we get

$$\left\lfloor \frac{k}{2} \right\rfloor < s(I).$$

Since the left hand side is an integer we have that

$$\frac{k-1}{2} \leq \left\lfloor \frac{k}{2} \right\rfloor \leq \lceil s(I) \rceil - 1.$$

This proves $k \leq 2 \cdot \lceil s(I) \rceil - 1 \leq 2 \cdot k^* - 1$ and hence the claim. □

The analysis is tight for the algorithm, which can be seen with the following instance with $2n$ items. For some $\varepsilon > 0$ let $s_{2i-1} = 2 \cdot \varepsilon$, $s_{2i} = 1 - \varepsilon$ for $i = 1, \ldots, n$.

First Fit Decreasing

The algorithm Next Fit never considers bins again that have been left behind. Thus the wasted capacity therein leaves room for improvement. A natural way is First Fit: Initially all bins are empty and we start with current number of bins $k = 0$ and item $i = 1$. Consider all bins $j = 1, \ldots, k$ and place item i in the first bin that has sufficient residual capacity, i.e., $a(i) = j$. If there is no such bin increment k and repeat until item n is assigned. One can prove that First Fit uses at most $k \leq \lceil 17/10 \cdot k^* \rceil$ many bins, where k^* is the optimal number.

There is a further natural heuristic improvement of First Fit, called First Fit Decreasing: Reorder the items such that $s_1 \geq \cdots \geq s_n$ and apply First Fit. The intuition behind considering large items first is the following: "Large" items do not fit into the same bin anyway, so we already use unavoidable bins and try to place "small" items into the residual space.

Theorem 7.4. First Fit Decreasing *is* 3/2*-competitive for* Bin Packing. *The algorithm runs in* $O\left(n^2\right)$ *time.*

Proof. Let k be the number of non-empty bins of the assignment a found by First Fit Decreasing and let k^* be the optimal number.

Consider bin number $j = \lceil 2/3k \rceil$. If it contains an item i with $s_i > 1/2$, then each bin $j' < j$ did not have space for item i. Thus j' was assigned an item i' with $i' < i$. As the items are considered in non-increasing order of size we have $s_{i'} \geq s_i > 1/2$. That is, there are at least j items of size larger than $1/2$. These items need to be placed in individual bins. This implies

$$k^* \geq j \geq \frac{2}{3}k.$$

Otherwise, bin j and any bin $j' > j$ does not contain an item with size larger than $1/2$. Hence the bins $j, j+1, \ldots, k$ contain at least $2(k-j)+1$ items, none of which fits into the bins $1, \ldots, j-1$. Thus we have

$$\begin{aligned} s(I) &> \min\{j-1, 2(k-j)+1\} \\ &\geq \min\{\lceil 2/3k \rceil - 1, 2(k-(2/3k+2/3))+1\} \\ &= \lceil 2/3k \rceil - 1 \end{aligned}$$

and $k^* \geq s(I) > \lceil 2/3k \rceil - 1$. This even implies

$$k^* \geq \left\lceil \frac{2}{3}k \right\rceil \geq \frac{2}{3}k$$

and hence the claim. □

Chapter 8

List Update

In this chapter, we consider the LIST UPDATE problem. Suppose we have a *list* that contains unsorted data-items and we receive a sequence I of *requests* to these items. A *request* can either be an insert, an access, or a delete operation. The cost of a request is equal to the position of the item in the list. In the sequel, we will assume that all the requests are accesses. This restricted model already captures the underlying problem of reorganizing the datastructure while allowing for simplified analysis.

We are allowed to reorganize the list. After the access to an item, we can bring it to any position further to the front at no additional cost. Such a reordering is called a *free-exchange*. Furthermore, we assume that we can swap the positions of any two adjacent items at unit-cost. This is called a *paid-exchange*.

The objective is to serve the entire sequence of requests at preferably minimal costs. In the *offline* version, the entire request sequence is known to the serving algorithm. In the *online* variant, the requests become known one-by-one, that is, the next request becomes available only after the current one is served.

We will compare the cost of any online algorithm against the optimal offline cost in terms of competitive ratio.

8.1 Online Algorithms

We will consider the following natural online algorithms:

MOVE-TO-FRONT: After accessing an item, move it to the front of the list without changing the relative order of the other items.

TRANSPOSE: After accessing an item, move it one position further to the front.

FREQUENCY-COUNT: Maintain an access counter of each item, which is initialized to zero. Upon an access, increment the counter of the requested item. Reorganize the list such that the items are always in non-increasing order of counters.

Theorem 8.1. *Let I be a sequence of n requests. Let* MOVE-TO-FRONT *and* OPT *start with the same configuration of the list.* $\text{MTF}(I)$ *denotes the cost of* MOVE-TO-FRONT. $\text{OPT}_A(I)$ *denotes the access cost of the optimal algorithm, while* $\text{OPT}_P(I)$ *and* $\text{OPT}_F(I)$ *denote the cost of the paid and free exchanges. Then it holds that*

$$\text{MTF}(I) \leq 2 \cdot \text{OPT}_A(I) + \text{OPT}_P(I) - \text{OPT}_F(I) - n.$$

Proof. Both algorithms start in the same initial configuration and both process the request sequence I. The notion of an *inversion* is crucial for the analysis: Let a and b denote two

distinct items. If the relative ordering of a and b is different in the lists of MOVE-TO-FRONT and OPT, these items define an inversion.

Consider the moment when k items are requested. Call i_k the number of inversions in the list of MOVE-TO-FRONT after the k-th request. Furtehrmore, let c_k denote the actual cost for serving the k-th request. Define the *amortized cost* of MOVE-TO-FRONT by $a_k = c_k + i_k - i_{k-1}$. Observe that we can write

$$\text{MTF}(I) = \sum_{k=1}^{n} c_k$$

and

$$\sum_{k=1}^{n} a_k = \sum_{k=1}^{n} c_k + i_k - i_{k-1} = \text{MTF}(I) - i_0 + i_n$$

or in other words

$$\text{MTF}(I) = i_0 - i_n + \sum_{k=1}^{n} a_k.$$

This means, by bounding the amortized cost, we also bound the actual cost of the algorithm. Notice that $i_0 = 0$, as MOVE-TO-FRONT and OPT start with the same initial configuration. Clearly, i_n is never negative.

For $j = 1, \ldots, \ell$, let x_j denote the item that is positioned at location j in the list of OPT just after processing request $i-1$. We show that the amortized cost to process request i is at most $(2a-1)+p-f$, where a is the access cost (not including transpositions) incurred by OPT and p and f are the cost for paid and free exchanges, respectively. Once this is established, the theorem is proved by summing over all requests.

The item x_j is positioned at j in the list of OPT and at k in MOVE-TO-FRONT. Let v denote the items that are after x_j in the list of OPT but before x_j in the list of MOVE-TO-FRONT. Then $k-1-v$ items precede x_j in both lists. This means $k-1-v \leq j-1$, since x_j is at position j in the list of OPT. When MOVE-TO-FRONT moves the item x_j to the front, it creates $k-1-v$ new inversions with respect to the list of OPT before it processes the request. MOVE-TO-FRONT eliminates v inversions. Hence, the contribution to the amortized cost is

$$k + (k-1-v) - v = 2(k-v) - 1 \leq 2j - 1 = 2a - 1,$$

as the access cost of OPT is exactly j. Each paid exchange contributes at most 1 and each free exchange contributes -1. Since OPT performs p paid and f free exchanges the claim is established. □

Corollary 8.2. MOVE-TO-FRONT *is* 2*-competitive.*

Theorem 8.3. TRANSPOSE *is not competitive.*

Proof. Consider a list with ℓ items and let x the item that is initially at the end. We will approximate OPT by MOVE-TO-FRONT at the expense of a factor of at most 2.

Access item x for $\ell/2$ many times. MOVE-TO-FRONT incurs a total cost of $\ell+(\ell/2-1) \leq 3/2\ell$. TRANSPOSE has a total cost of $\ell+(\ell-1)+\cdots+\ell/2 \geq \ell^2/4$. This ratio is unbounded, the construction can be repeated, and TRANSPOSE is hence not competitive. □

Theorem 8.4. FREQUENCY-COUNT *is not competitive.*

This is left to the reader as an exercise.

8.2 Lower Bound

Theorem 8.5. *For the* List Update *problem with a list on ℓ items, any deterministic online algorithm has a competitive ratio of at least $2 - 2/(\ell + 1)$.*

Proof. Consider the adversary that always requests the last item in the list of the deterministic online algorithm. There are n requests and hence the cost of the online algorithm is $n\ell$.

There are $\ell!$ permutations of the list. Hence there are $\ell!$ offline strategies that first use paid exchanges to establish one of these permutations and then serve the entire sequence in the fixed permutation. The cost to reach any fixed permutation is at most ℓ^2.

We look at the total cost of serving all requests in all of these strategies. For a single request, this cost is

$$\sum_{i=1}^{\ell} i(\ell-1)! = (\ell-1)!\frac{\ell(\ell+1)}{2},$$

since there are $(\ell - 1)!$ permutations of the remaining items and the item is at position $i = 1, \ldots, \ell$ in either strategy.

The total cost for the entire sequence is at most

$$n(\ell-1)!\frac{\ell(\ell+1)}{2} + \ell!\ell^2.$$

Hence the average cost over all offline strategies is at most $1/2 \cdot n(\ell + 1) + \ell$ and at least one of the strategies has at most average cost.

The ratio $n\ell/(1/2 \cdot n(\ell + 1) + \ell)$ approaches $2 - 2/(\ell + 1)$ as claimed. □

Appendix A

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.

A.1 Basics

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 A.1. *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 A.2. *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 A.3. *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 A.4. *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 A.5 (Linearity of Expectation)**.** *For random variables $X_1, \ldots, X_n$ and constants $a_1, \ldots, a_n, b$ we have that*

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

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 A.6. *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

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

A.2 Bounds on Probabilities

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

Theorem A.7 (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 A.8 (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 $t > 0$.

A.3 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 $\mathrm{Var}[X] = ((b-a+1)^2 - 1)/12$.