Lecture Notes

Randomized Algorithms and Probabilistic Analysis

Prof. Dr. Heiko Röglin

Prof. Dr. Melanie Schmidt

with edits by Dr. Daniel Schmidt

Department of Computer Science

University of Bonn

April 10, 2024

Contents

Please send comments and corrections to `roeglin@cs.uni-bonn.de`.

Part I

Randomized Algorithms

Part I is based on the books [MU05] and [MR95]. Section 3.3.2 on routing in hypercubes was inspired by lecture notes by Thomas Worsch [Wor15] (in German).

Chapter 1

Discrete Event Spaces and Probabilities

In Part I, we are interested in algorithmic problems with *discrete* solution spaces that we solve by *randomized* algorithms. We thus need the formal basis of analyzing random processes for discrete event spaces. Section 1.1 and Section 1.2 correspond to 1.1 and 1.2 in [MU05]. Section 1.3.1 corresponds to 1.4 in [MU05] and 10.2 in [MR95].

We define $\mathbb{N} = \{1, 2, 3, \ldots\}$ to be the set of natural numbers and $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$ to be the set of natural numbers with zero. For $n \in \mathbb{N}$, we denote the set $\{1, \ldots, n\}$ by $[n]$. We denote the power set of a set M by 2^M. We use $\mathbb{R}_{\geq 0}$ to denote the set $\{x \in \mathbb{R} \mid x \geq 0\}$. Given a vector $x \in \mathbb{R}^n$, we use $x_1, \ldots, x_n$ to denote its entries, i.e., $x = (x_1, \ldots, x_n)^\mathsf{T}$. The norm $\|x\|$ of a vector $x \in \mathbb{R}^n$ is always meant to be its Euclidean norm, i.e., $\|x\| = \sqrt{x_1^2 + \cdots + x_n^2} = \sqrt{x^\mathsf{T} x}$.

1.1 Discrete Probability Spaces

We define what we mean by probabilities and introduce basic notation and rules. Intuitively, we can think of probabilities as frequencies when observing a process for a long time. For example, the probability that we roll a six with a die is 1/6 – we expect that one sixth of all rolls will be six if we observe the die infinitely long. Notice that we express probabilities by numbers between zero and one.

To define probabilities, we first need a formal notion of *events*: Everything that can happen in the process that we observe. The most basic type of event is the *elementary event*. When we roll a die, one of six elementary events happens. Based on these, we can define more complicated events like „rolling an even number". These can be described by sets of elementary events.

Definition 1.1. *Let* Ω *be a finite or countable set that we call* sample space*. The elements of* Ω *are* elementary events*. An* event *is a subset* $A \subseteq \Omega$*. The* complementary event *of* A *is* $\overline{A} = \Omega \setminus A$*.*

Recall that the set of all subsets of Ω is 2^Ω, the power set. Thus, every event A is an element of 2^Ω. If $A, B \in 2^\Omega$ are two events, then $A \cup B$ is the event that at least one of A and B occurs, and $A \cap B$ is the event that both events occur. Recall that A and B are disjoint if $A \cap B = \emptyset$.

A *probability measure* assigns a value to every event – its probability. When events are disjoint, then their probability has to add up to the probability of the joint event.

Definition 1.2. *A* probability measure *or* probability *on Ω is a mapping $\mathbf{Pr}\colon 2^\Omega \to [0,1]$ that satisfies $\mathbf{Pr}(\Omega) = 1$ and is σ-additive. The latter means that*

$$\mathbf{Pr}\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mathbf{Pr}(A_i)$$

holds for every countably infinite sequence $A_1, A_2, \ldots$ of pairwise disjoint events. We say that $(\Omega, \mathbf{Pr})$ is a discrete probability space*.*

We observe that $\mathbf{Pr}(\emptyset)$ is always zero because $\mathbf{Pr}(\Omega) = \mathbf{Pr}(\Omega \cup \emptyset) = 1 + \mathbf{Pr}(\emptyset)$. More generally, we have $\mathbf{Pr}(\overline{A}) = 1 - \mathbf{Pr}(A)$ as $1 = \mathbf{Pr}(\Omega) = \mathbf{Pr}(A \cup \overline{A})$. In the same manner, we can prove that

$$\mathbf{Pr}(A) = \sum_{a \in A} \mathbf{Pr}(\{a\}). \tag{1.1}$$

Thus, the probability of an event A is the sum of the probabilities of the elementary events that A consists of. In fact, probability measures on countable sets can equivalently be defined by setting $\mathbf{Pr}(\{a\})$ for all $A \in \Omega$, extending it to events by (1.1) and demanding that $\mathbf{Pr}(\Omega) = 1$. Finally, we observe that for any events A, B, we have $\mathbf{Pr}(A \setminus B) = \mathbf{Pr}(A) - \mathbf{Pr}(A \cap B)$ because A is the disjoint union of $A \setminus B$ and $A \cap B$.

Example 1.3. *We look at some examples for sample spaces and probability measures.*

- *When we model one roll of a die, we set $\Omega = \{1,2,3,4,5,6\}$ and $\mathbf{Pr}(i) = 1/6$ for all $i \in \Omega$. The event to roll an even number is $A = \{2,4,6\}$, and its probability is $\mathbf{Pr}(A) = \mathbf{Pr}(2) + \mathbf{Pr}(4) + \mathbf{Pr}(6) = 1/2$.*
- *When we model a fair coin toss, we set $\Omega = \{H, T\}$ for the elementary events that we get heads or tail, and let $\mathbf{Pr}(H) = \mathbf{Pr}(T) = 1/2$. We can also model multiple fair coin tosses. For two turns, we set $\Omega = \{(H,H),(H,T),(T,H),(T,T)\}$ for the four possible outcomes. Intuitively, they all have the same probability, i.e., we set $\mathbf{Pr}((H,H)) = \mathbf{Pr}((H,T)) = \mathbf{Pr}((T,H)) = \mathbf{Pr}((T,T)) = 1/4$. We observe that the event A to get heads in the first turn is still $1/2$: It is $A = \{(H,H),(H,T)\}$ and thus $\mathbf{Pr}(A) = \mathbf{Pr}((H,H)) + \mathbf{Pr}((H,T)) = 1/4 + 1/4 = 1/2$.*
- *Now we want to model two fair coin tosses, but we cannot distinguish the coins and throw them at the same time. Thus, we model $\Omega = \{\{H,H\},\{H,T\},\{T,T\}\}$ for the three elementary events that both coins come up heads, one shows heads and one tail, or both show tails. We still want to model fair coin tosses, so we now need different probabilities for the elementary events: We set $\mathbf{Pr}(\{H,H\}) = \mathbf{Pr}(\{T,T\}) = 1/4$ and $\mathbf{Pr}(\{H,T\}) = 1/2$.*

In two of the examples in 1.3, we assigned the same probability to every elementary event. In this case, the elementary events occur *uniformly at random*. When we say that we choose an element uniformly at random from t choices/events, we mean that each of the t events has probability $1/t$.

Application: Polynomial Tester (Part I: The Power of Random Decisions) Our first randomized algorithm tests whether two polynomials $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ are equal. The degree of the polynomials will be important, so let d be the maximum degree of f and g.

Our algorithm cannot see the formulas that describe f and g. Instead, it can send an $x \in \mathbb{R}$ to a ***black box*** and receives $f(x)$ and $g(x)$. The simplest randomized algorithm we can think of is to send an x chosen uniformly at random from $\mathbb{R}$ and to check if $f(x) = g(x)$ is true. If yes, we output that f and g are equal, if no, then we output that f and g are not equal. In the latter case, our algorithm has no error because we found a proof that f and g are not equal. But what is the probability that f and g are not equal, but we still output yes?

We need a bit more knowledge about polynomials. Since f and g are polynomials, we know that $f - g$ is a polynomial, too. Furthermore, the degree of $f - g$ is bounded by d as well. The fact that we now crucially need is that a polynomial of degree at most d (that is not the zero polynomial) has at most d roots. Thus, $f(x) - g(x) = 0$ can only be true for d different values of x. That means that $f(x) = g(x)$ can only be true for d different values of x as well!

What is the probability that we pick one of these d values when choosing an x uniformly at random? Is it even possible to choose an element uniformly at random from $\mathbb{R}$? We resolve our problems by changing the algorithm. Now we choose x uniformly at random from a set of t possible values. Formally, we set $\Omega := \{1, \ldots, t\}$ and $\mathbf{Pr}(x) = 1/t$. We do ***worst-case analysis***, so we assume that Ω contains as many roots as possible. To have a chance to give the right answer, we thus need $t \geq d + 1$.

Now the probability that we choose a root and falsely output that f and g are equal is d/t. If we set $t = 100d$, then the failure probability of our algorithm is $1/100$. This is true even though our algorithm only checks a single x! A deterministic algorithm could decide the question without any error by checking for $d+1$ values whether $f(x) = g(x)$ is true. Our randomized version saves d of these checks at the cost of a small error.

Recall that a negative answer of our algorithm has no error, only a positive answer might be incorrect. We call algorithms of this type ***randomized algorithms with one-sided error***.

Union bound and product spaces We often want to analyze the probability that one of some events occurs. This is also helpful when we want to analyze that none of a set of events occurs. For two events, we get the following lemma.

Lemma 1.4. *Let* $(\Omega, \mathbf{Pr})$ *be a discrete probability space and let* $A, B \in 2^{\Omega}$ *be events. It holds that*

$$\mathbf{Pr}(A \cup B) = \mathbf{Pr}(A) + \mathbf{Pr}(B) - \mathbf{Pr}(A \cap B).$$

Proof. We write $A \cup B$ as the disjoint union of A and $B \setminus A$ and apply σ-additivity:

$$\mathbf{Pr}(A \cup B) = \mathbf{Pr}(A) + \mathbf{Pr}(B \setminus A) = \mathbf{Pr}(A) + \mathbf{Pr}(B) - \mathbf{Pr}(A \cap B). \qquad \square$$

We should memorize two things. First, the probability for $A \cup B$ *can* be smaller than $\mathbf{Pr}(A)+\mathbf{Pr}(B)$. The two terms are only equal if A and B are disjoint. Second, $\mathbf{Pr}(A \cup B)$ *cannot* be greater than $\mathbf{Pr}(A)+\mathbf{Pr}(B)$. It is always $\mathbf{Pr}(A \cup B) \leq \mathbf{Pr}(A)+\mathbf{Pr}(B)$. This simple fact can be extremely useful. It can be generalized to the union of a family $(A_i)_{i \in \mathbb{N}}$ of events.

Lemma 1.5 (Union Bound)**.** *For a finite or countably infinite sequence $A_1, A_2, \ldots$ of events, it holds that*

$$\mathbf{Pr}\left(\bigcup_{i=1}^{\infty} A_i\right) \leq \sum_{i=1}^{\infty} \mathbf{Pr}(A_i).$$

Proof. Recall that $A \setminus B = \{a \in \Omega \mid a \in A, a \notin B\}$. Observe that $\mathbf{Pr}(A \setminus B) \leq \mathbf{Pr}(A)$ holds for any two events $A, B \in 2^{\Omega}$ by definition. Thus, we can use σ-additivity to obtain that

$$\mathbf{Pr}\Big(\bigcup_{i=1}^{\infty} A_i\Big) = \mathbf{Pr}\left(\bigcup_{i=1}^{\infty}\Big(A_i \setminus \Big(\bigcup_{j=1}^{i-1} A_j\Big)\Big)\right) = \sum_{i=1}^{\infty} \mathbf{Pr}\Big(A_i \setminus \Big(\bigcup_{j=1}^{i-1} A_j\Big)\Big) \leq \sum_{i=1}^{\infty} \mathbf{Pr}(A_i).$$

□

Let $B_1, B_2, \ldots, B_\ell$ be bad events. We want *none* of $B_1, \ldots, B_\ell$ to occur. The probability that at least one of them occurs is bounded by

$$\mathbf{Pr}\left(\bigcup_{i=1}^{\ell} B_i\right) \leq \sum_{i=1}^{\ell} \mathbf{Pr}(B_i)$$

by the Union Bound. Thus, the probability that no bad event occurs, which is the complementary event, has a probability of at least

$$1 - \sum_{i=1}^{\ell} \mathbf{Pr}(B_i).$$

Similarly, assume that we have a set of good events $G_1, G_2, \ldots, G_\ell$. Here, we want all of $G_1, \ldots, G_\ell$ to occur. For any $i \in \{1, \ldots, \ell\}$, the complementary event $\overline{G_i}$ has probability $1 - \mathbf{Pr}(G_i)$. We first compute the probability that any of the $\overline{G_1}, \ldots, \overline{G_\ell}$ occurs.

$$\mathbf{Pr}\left(\bigcup_{i=1}^{\ell} \overline{G_i}\right) \leq \sum_{i=1}^{\ell}(1 - \mathbf{Pr}(G_i)).$$

The event that all of $G_1, \ldots, G_\ell$ occur is the complementary event of $\bigcup_{i=1}^{\ell} \overline{G_i}$ and thus has a probability of at least $1 - \sum_{i=1}^{\ell}(1 - \mathbf{Pr}(G_i))$.

Application: Polynomial Tester (Part II: More than one bad event) Again, we want to test if two polynomials f and g are equal. This time, we only have access to a black box *with error*. Our algorithm can send a value $x \in \mathbb{R}$ to the black box and ask whether $f(x) = g(x)$ is true. With probability 0.9, the black box gives a correct answer, and with probability 0.1, it replies incorrectly. Our algorithm does not change: It still chooses one of t possible values for x, sends x to the black box and then repeats the answer of the black box.

We model this situation with two probability spaces. The probability space $(\Omega_1, \mathbf{Pr}_1)$ models the random behavior of our algorithm. It is defined by $\Omega_1 := \{1, \ldots, t\}$ and $\mathbf{Pr}_1(x) = 1/t$ for all $x \quad \Omega_1$. The random behavior of the black box is described by $(\Omega_2, \mathbf{Pr}_2)$, $\Omega_2 = \{R, W\}$ and $\mathbf{Pr}_2(R) = 0.9$, $\mathbf{Pr}_2(W) = 0.1$.

To analyze the whole process, we need to combine $(\Omega_1, \mathbf{Pr}_1)$ and $(\Omega_2, \mathbf{Pr}_2)$. We assume that the error of the black box is *independent* of the random behavior of our algorithm. We model this by using the *product space* $(\Omega, \mathbf{Pr})$.

Fact 1.6. *Let* $(\Omega_1, \mathbf{Pr}_1)$ *and* $(\Omega_2, \mathbf{Pr}_2)$ *be discrete probability spaces. Define the* product space $(\Omega, \mathbf{Pr})$ *by* $\Omega = \Omega_1 \times \Omega_2$ *and* $\mathbf{Pr}(x, y) = \mathbf{Pr}_1(x) \cdot \mathbf{Pr}_2(y)$ *for all* $(x, y) \quad \Omega$*. Then* $(\Omega, \mathbf{Pr})$ *is a discrete probability space. Furthermore,*

$$\mathbf{Pr}_1(x) = \mathbf{Pr}(\{x\} \times \Omega_2) \text{ and } \mathbf{Pr}_2(y) = \mathbf{Pr}(\Omega_1 \times \{y\})$$

is true for all $(x, y) \quad \Omega$*.*

The random behavior of our algorithm and the black box is jointly described by the product space of $(\Omega_1, \mathbf{Pr}_1)$ and $(\Omega_2, \mathbf{Pr}_2)$. Notice that the outcome of our algorithm now has *two-sided error*. When f and g are equal, $f(x) = g(x)$ is always true, independent of the x that we choose. However, the event $W \quad \Omega_2$ that the black box replies incorrectly might occur. Thus, the error probability of the algorithm in this case is $\mathbf{Pr}(\Omega_1 \times \{W\}) = \mathbf{Pr}_2(\{W\}) = 0.1$.

If f and g are not equal, then the probability that we choose one of the $k \leq d$ roots $r_1, \ldots, r_k$ as x from $\{1, \ldots, t\}$ is bounded by d/t. Let $A = \{r_1, \ldots, r_k\} \times \Omega_2$ be the event that this happens. Furthermore, let $B = \Omega_1 \times \{W\}$ be the event that the black box answers incorrectly. The events A and B are bad events, we want both of them to not occur[1]. As we argued above, we can use the Union Bound to obtain

$$\mathbf{Pr}(A \quad B) \leq \mathbf{Pr}(A) + \mathbf{Pr}(B) = \mathbf{Pr}_1(\{r_1, \ldots, r_k\}) + \mathbf{Pr}_2(\{W\}) = \frac{k}{100d} + 0.1 \leq 0.11,$$

where we again set $t = 100d$.

In this example, we could have computed the failure probability exactly. However, we have learned a simple yet powerful tool that is often helpful when bounding the error probability of more complex randomized algorithms.

1.2 Independent Events & Conditional Probability

In the last section, we intuitively used the term *independent events*. We now formally define what we mean by independence.

[1] If A and B both happen, we “accidentially” give the correct answer. We neglect this case in our analysis as we are only interested in an upper bound on the failure probability.

Definition 1.7. *Let* $(\Omega, \mathbf{Pr})$ *be a discrete probability space. We say that two events* $A \quad 2^{\Omega}$ *and* $B \quad 2^{\Omega}$ *are* independent *if* $\mathbf{Pr}(A \cap B) = \mathbf{Pr}(A) \cdot \mathbf{Pr}(B)$ *holds. A sequence* $A_1, \ldots, A_k$ *of events is independent if*

$$\mathbf{Pr}\left(\bigcap_{i \quad I} A_i\right) = \prod_{i \quad I} \mathbf{Pr}(A_i)$$

holds for every $I \quad \{1, \ldots, k\}$*. The sequence* $A_1, \ldots, A_k$ *is* pairwise independent *if* A_i *and* A_j *are independent for every* $i, j \quad \{1, \ldots, k\}$ *with* $i = j$*.*

Intuitively, the independence of two events A and B means that we gain no information about B when we get to know whether A occurred or not, and vice versa.

Example 1.8. *Consider the probability space* $(\Omega, \mathbf{Pr})$ *which models two independent rolls of a die where we set* $\Omega = \{(i, j) \mid i, j \quad \{1, \ldots, 6\}\}$ *and* $\mathbf{Pr}(i, j) = 1/36$*.*

- *Intuitively, the outcome of the first roll of a fair die has no consequence for the outcome of the second roll. Assume that we know that event* $A = \{(2, j) \mid j \quad \{1, \ldots, 6\}\} \quad \{(4, j) \mid j \quad \{1, \ldots, 6\}\} \quad \{(6, j) \mid j \quad \{1, \ldots, 6\}\}$ *occurred, which is that the first roll gives an even number. The probability of this event is* $\mathbf{Pr}(A) = 1/2$ *because* $|A| = 18$ *and all elementary events have the same probability. Let* B *be the event that the second roll gives a* 3*, which has probability* $1/6$*. We observe that* $A \cap B = \{(2, 3), (4, 3), (6, 3)\}$ *has probability* $1/12$*. This confirms our intuitive idea that* A *and* B *are independent because*

$$\mathbf{Pr}(A \cap B) = 1/12 = (1/2) \cdot (1/6) = \mathbf{Pr}(A) \cdot \mathbf{Pr}(B).$$

- *Consider the event* C *that the sum of the two rolls is* 8*. This event is not independent of the event* D *that the first roll is a* 1*, since the occurrence of* D *means that* C *has probability zero. Consider the event* E *that the first roll is a* 5*. Again,* E *and* C *are not independent, because rolling a* 5 *and thus not rolling a* 1 *increases the probability that the sum is* 8*. We verify this intuition. We have* $E \cap C = \{(5, 3)\}$ *with probability* $1/36$*. Event* E *has probability* $1/6$*. Finally, the tuples* $(i, 8 - i)$ *for* $i \quad \{2, \ldots, 6\}$ *are the possible outcomes with sum* 8*, so* $\mathbf{Pr}(C) = 5/36$*. We see that*

$$\mathbf{Pr}(E \cap C) = \frac{1}{36} > \frac{5}{6 \cdot 36} = \mathbf{Pr}(E) \cdot \mathbf{Pr}(C).$$

- *Now assume that* F *is the event that the sum is* 7*, and* G *is the event that we rolled a* 6 *with the first die. From the last example, we might get the intuition that these events are not independent. However, at a second glance we see that* $\mathbf{Pr}(F) = 1/6$*,* $\mathbf{Pr}(G) = 1/6$ *and* $\mathbf{Pr}(F \cap G) = 1/36$*. Events* F *and* G *are indeed independent. We should always verify our intuition about independence by computing the probabilities, in particular in more complex scenarios.*

When events are dependent, then knowledge about the occurrence of one of the events changes the probability of the other. This is made clear by the definition of *conditional probabilities*.

Definition 1.9. *Let* $(\Omega, \mathbf{Pr})$ *be a discrete probability space and let* A, B 2^Ω *be events with* $\mathbf{Pr}(B) > 0$*. The* conditional probability of A given B *is defined as*

$$\mathbf{Pr}(A \mid B) = \frac{\mathbf{Pr}(A \cap B)}{\mathbf{Pr}(B)}.$$

The definition is consistent with our notion of independent events: If A, B 2^Ω with $\mathbf{Pr}(B) > 0$ are independent, then

$$\mathbf{Pr}(A \mid B) = \frac{\mathbf{Pr}(A \cap B)}{\mathbf{Pr}(B)} = \frac{\mathbf{Pr}(A) \cdot \mathbf{Pr}(B)}{\mathbf{Pr}(B)} = \mathbf{Pr}(A).$$

The occurrence of B does not influence the probability of A, thus the conditional probability $\mathbf{Pr}(A \mid B)$ is just $\mathbf{Pr}(A)$.

Example 1.10. *We continue Example 1.8. Events* C *and* E *are not independent. We argued that* E *influences* C *because its occurrence ensures that we did not roll a* 1*. Indeed, the conditional probability of* C *given* E *is greater than* $\mathbf{Pr}(C)$*:*

$$\mathbf{Pr}(C \mid E) = \frac{\mathbf{Pr}(C \cap E)}{\mathbf{Pr}(E)} = \frac{1/36}{1/6} = \frac{6}{36} > \frac{5}{36} = \mathbf{Pr}(C).$$

In Example 1.10, we computed the conditional probability from the data that we are given by the modeling of our random experiment. However, if the conditional probability of an event is *part of the modeling*, then we can use Definition 1.9 reversely to compute $\mathbf{Pr}(A \cap B)$ or $\mathbf{Pr}(B)$. We see an example for that as well.

Example 1.11. *Assume that we know the following facts about a group of people.*

- *In this group of people, 30% play the piano.*
- *Among the piano players, 90% like the composer Bach.*
- *Among the people that like Bach, 65% play the piano.*

Question: How many people in the whole group like Bach?

We define the events B *for 'a person likes Bach' and* P *for 'a person plays the piano'. From the data given to us, we know that* $\mathbf{Pr}(P) = 0.3$*,* $\mathbf{Pr}(B \mid P) = 0.9$ *and* $\mathbf{Pr}(P \mid B) = 0.65$ *(if we choose a person uniformly at random from the group). By the definition of conditional probability,*

$$\mathbf{Pr}(B \cap P) = \mathbf{Pr}(P) \cdot \mathbf{Pr}(B \mid P) = 0.27 \text{ and}$$

$$\mathbf{Pr}(B) = \frac{\mathbf{Pr}(B \cap P)}{\mathbf{Pr}(P \mid B)} = \frac{0.27}{0.65} \approx 0.42.$$

Thus, 42% of the people in this group like the composer Bach.

Now we want to reduce the error probability of our polynomial tester by repetitions. Independent repetitions can be modeled by product spaces. Recall the definition of product spaces from Fact 1.6. A product space of two probability spaces can be used to model two independent runs of an algorithm. We can verify that events that correspond to different components of this product space are always independent.

Fact 1.12. *Let $(\Omega_1, \mathbf{Pr}_1)$ and $(\Omega_2, \mathbf{Pr}_2)$ be discrete probability spaces and let $(\Omega, \mathbf{Pr})$ be the product space. Then the events $A \times \Omega_2$ and $\Omega_1 \times B$ are independent for all choices of $A \quad 2^{\Omega_1}$ and $B \quad 2^{\Omega_2}$.*

Notice that the definition of product spaces can be easily extended to products of t copies. If $(\Omega_1, \mathbf{Pr}_1), \ldots (\Omega_t, \mathbf{Pr}_t)$ are discrete probability spaces, then the product space is defined by $\Omega = \Omega_1 \times \ldots \times \Omega_t$ and $\mathbf{Pr}(x_1, \ldots, x_t) = \mathbf{Pr}_1(x_1) \cdot \ldots \cdot \mathbf{Pr}_t(x_t)$. Fact 1.6 and Fact 1.12 extend to this slightly more general case.

We will also see an example for an algorithm that is modeled with dependent runs. Then we will use a conditional modeling in the design of the algorithm, so that we can analyze it with conditional probabilities.

Application: Polynomial Tester (Part III: Multiple runs reduce error) We return to our randomized algorithm with one-sided error for the scenario where a black box answers whether $f(x)$ and $g(x)$ are equal for a given value $x \quad$ R, and where the black box has no error. We observed that the error probability of the algorithm which chooses one of t values for x uniformly at random is at most d/t. We extend this algorithm. It now chooses n values $x_1, \ldots, x_n$ from $\{1, \ldots, t\}$ uniformly at random. It is possible that the algorithm chooses the same value multiple times. If $f(x_i) = g(x_i)$ for at least one $i \quad \{1, \ldots, n\}$, then we output no, otherwise, we output yes. If f and g are equal, then the algorithm has no error. We assume that f and g are not equal and analyze the failure probability. Let A_i be the event that x_i satisfies $f(x_i) = g(x_i)$. We know that $\mathbf{Pr}(A_i) \leq d/t$, or, for $t = 100d$, $\mathbf{Pr}(A_i) \leq 1/100$. The event that the algorithm fails is $A_1 \cap \ldots \cap A_n$. The events A_i are independent because they correspond to different components of the product space that models our independent runs. Thus, the failure probability is

$$\mathbf{Pr}(A_1 \cap \ldots \cap A_n) = \mathbf{Pr}(A_1) \cdot \ldots \cdot \mathbf{Pr}(A_n) \leq \left(\frac{1}{100}\right)^n.$$

This probability decreases exponentially in the number of runs, demonstrating the power of independent runs for reducing the failure probability. Assume that we want to decrease the failure probability below δ, i.e., we want $(\frac{1}{100})^n \leq \delta$. Solving this inequality tells us that we need $(\log \frac{1}{\delta})/(\log 100)$ independent runs to achieve this. For example, we need 4 runs to reduce the failure probability to 1/100000000.

To make the events A_i independent, we had to allow that the randomized algorithm chooses the same value more than once. We can decrease the failure probability further by dropping the independence of different runs, at the cost of a more complex analysis. Assume that the algorithm picks x_i uniformly at random from $\{1, \ldots, t\} \setminus \{x_1, \ldots, x_{i-1}\}$ where we assume that $n \leq t$. Again, we analyze the probability that the algorithm outputs yes even though f and g are not equal. Observe that the algorithm discovers $f = g$ if $n \geq d + 1$, thus we now assume that $n \leq d$. Define A_i as before, but notice that these events are no longer independent. We rewrite the failure probability by using Definition 1.9:

$$\mathbf{Pr}(A_1 \cap \ldots \cap A_n) = \mathbf{Pr}(A_n \mid A_1 \cap \ldots \cap A_{n-1}) \cdot \mathbf{Pr}(A_1 \cap \ldots \cap A_{n-1}).$$

Applying the definition of conditional probability iteratively, we get that

$$\mathbf{Pr}(A_1 \cap \ldots \cap A_n) = \mathbf{Pr}(A_1) \cdot \mathbf{Pr}(A_2 \mid A_1) \cdot \mathbf{Pr}(A_3 \mid A_1 \cap A_2) \cdot \ldots \cdot \mathbf{Pr}(A_n \mid A_1 \cap \ldots \cap A_{n-1}).$$

Let $\mathbf{Pr}(A_i \mid A_1 \cap \ldots \cap A_{i-1})$ be one of the n factors. The condition $A_1 \cap \ldots \cap A_{i-1}$ means that we already chose $i-1$ values, and all of them satisfied $f(x) = g(x)$. Then there are now $i-1$ fewer values with this property available, and the probability that x_i is chosen as one of them is at most $(d-(i-1))/(t-(i-1))$. For $t = 100d$, we get

$$\mathbf{Pr}(A_1 \cap \ldots \cap A_n) \leq \prod_{i=1}^{n} \frac{d-(i-1)}{100d-(i-1)}.$$

This term is always bounded from above by $(1/100)^n$. For $i \geq 2$ it is strictly smaller, so the error probability does indeed decrease when we avoid to choose the same value more than once.

1.3 Applications

Our first applications were to test the equality of two polynomials in different settings. Next, we see randomized algorithms for an important combinatorial optimization problem, the *minimum cut problem*. After that, we use our knowledge about discrete probability spaces to sample uniformly at random from an infinite stream of numbers. This is an important subroutine for the development of *data stream algorithms*.

1.3.1 The Minimum Cut Problem

This section is devoted to an application, *the (global) minimum cut problem*. For this problem, we want to cut a graph into (at least) two pieces (connected components) by deleting as few edges as possible. Let $G = (V, E)$ be an undirected graph. We assume that $n = |V| \geq 2$, $m = |E|$ and that G is connected (otherwise, it already has two connected components).

It is convenient to define a cut based on nodes instead of edges. So let $S \quad V$ with $S = \quad$ and $S = V$ be a non-trivial subset of the nodes. To separate S from $V \setminus S$, we need to delete all edges that have one endpoint in S and one endpoint in $V \setminus S$. These are exactly the edges in the set $\delta(S) \quad E$ that is defined as follows:

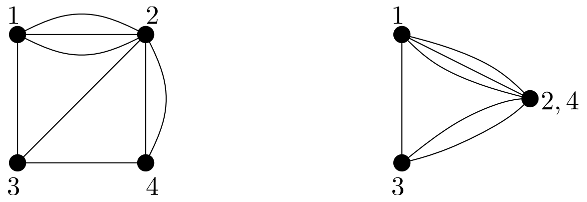
$$\delta(S) = \{\{u, v\} \quad E \mid u \quad S, v \ / \ S\}.$$

We want to minimize the number of edges in $\delta(S)$.

Problem 1.13 (MinCut)**.** *Given a graph* $G = (V, E)$*,* the (global) minimum cut (MinCut) problem *is to compute a set* $S \quad V$ *with* $S = \quad$ *and* $S = V$ *that minimizes* $|\delta(S)|$*.*

We discuss two randomized algorithms for this problem. Our motivation is two-fold: Both algorithms are conceptually simple with short pseudo code descriptions. Additionally, the second algorithm is very efficient. Of course, deterministic algorithms for the MinCut problem are also known, the best one achieving a running time of $O(nm + n^2 \log n)$, see [NI92] and [SW97]. For dense graphs (where $m \quad \Theta(n^2)$), this is $O(n^3)$. The second randomized algorithm that we learn about achieves a running time of $O(n^2 (\log n)^{O(1/\delta)})$, where δ is the desired error probability. This is asymptotically faster.

Contracting edges Both algorithms are based on a contraction operation that we name `Contract-Edge(`G, e`)`. For an edge $e = \{u, v\}$ this operation contracts e. It replaces u and v by one node uv. Edges that had either u or v as an endpoint are rerouted to uv, edges between u and v are deleted. The operation can create parallel edges. Formally, this means that the algorithm works with ***multigraphs*** where parallel edges are allowed. In the following picture, one of the edges between 2 and 4 is contracted:



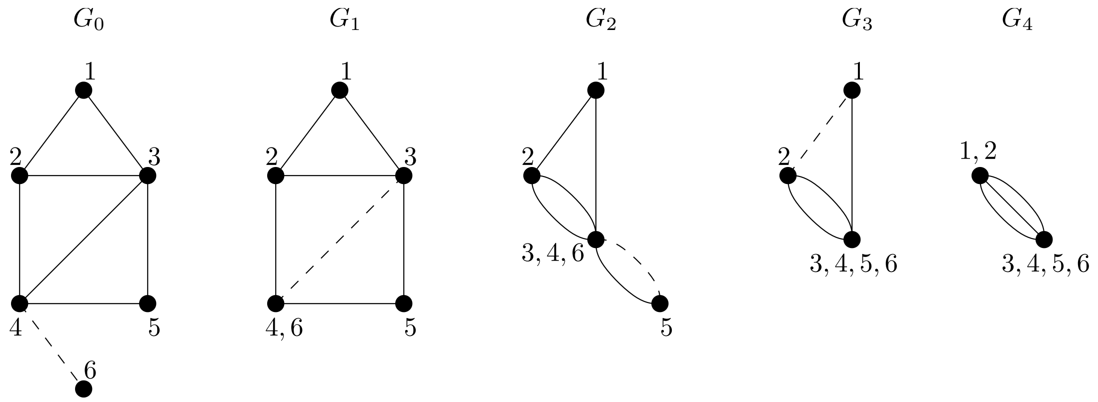
More precisely, the operation `Contract-Edge(`G, e`)` replaces a multigraph G by a multigraph $G \; = (V \; , E \;)$ with $V \; = V \setminus \{u, v\} \quad \{uv\}$ and $E \; := \{\{w, uv\} \mid w \; / \; \{u, v\}, \{w, v\} \quad E \quad \{w, u\} \quad E\} \quad \{\{w, x\} \mid \{w, x\} \quad E, w, x \; / \; \{u, v\}\}$.

Karger's Contract Algorithm The first randomized algorithm was published by Karger [Kar93]. It does $n - 2$ iterations. In every iteration, it contracts a random edge. The edge is chosen uniformly at random from the edges that are still present. Thus, parallel edges are important: The more parallel edges connect u and v, the more likely it is that u and v are merged.

To describe the algorithm, we number the nodes arbitrarily, i.e., we assume that $V = \{1, \ldots, n\}$. Then we assign the label $\ell(v) = \{v\}$ to node v for all $v \quad V$. The resulting graph with labels is the graph $G_0 = (V_0, E_0)$. Iteration $i \quad \{1, \ldots, n-2\}$ now creates the multigraph $G_i = (V_i, E_i)$. It chooses an edge $e = \{u, v\}$ uniformly at random from the edges in G_{i-1} and calls `Contract-Edge(`G_{i-1}, e`)`. It also sets the label of the new node uv to $\ell(u) \quad \ell(v)$. We get the following pseudocode:

```
Contract($\mathbf{G} = (\mathbf{V}, \mathbf{E})$)
  For all $v$   $V$: label $v$ with $\ell(v) = \{v\}$
  While $|V| > 2$:
      choose $e$   $E$ uniformly at random, $e = \{u, v\}$
      Contract-Edge($G, e$)        % creates node $uv$
      Set  $\ell(uv) = \ell(u)$   $\ell(v)$
  Let  $G = (\{x, y\}, E\ )$ be the current graph
  Return $\ell(x)$
```

Every iteration reduces the number of nodes by one. The resulting graph G_{n-2} has two nodes, and the labels of the two nodes correspond to two disjoint subsets S_1, S_2 V with S_1 $S_2 = V$. The algorithm outputs one of them (observe that $\delta(S_1) = \delta(S_2)$, so S_1 or S_2 are solutions with the same quality). The following picture shows an example run of the algorithm. The dashed line is the edge that is contracted in the next step.



In this example, the algorithm outputs $\{1, 2\}$ or $\{3, 4, 5, 6\}$, and the value of this solution is three because there are three edges between $\{1, 2\}$ and $\{3, 4, 5, 6\}$ (observe that these three edges exist in G_4, but also in G_3, G_2, G_1 and the original graph G_0). Of course, this example only shows one possible outcome of the algorithm. We want to know how likely it is that the algorithm outputs a minimum cut. Recall that a pair u, v is more likely to be contracted if a lot of edges connect u and v. This increases the probability that we end up with a good solution. We now prove that the probability that the algorithm indeed outputs a minimum cut is significantly larger than 0.

Theorem 1.14. *The algorithm* `Contract` *always outputs a cut. With a probability of at least* $\frac{2}{n(n-1)}$*, this cut is a minimum cut.*

Proof. To see that the output S is always a cut, observe that a label can never be empty, and that it cannot be V as long as there are at least two nodes. We can say something more about the multigraphs that the algorithm creates. Recall that the ***degree*** of a node v V is the number of edges $e = \{x, y\}$ E with v $\{x, y\}$. For any G_i, i $\{0, \ldots, n-2\}$, the degree of a node v V_i is equal to the number of edges that leave the cut that the label of v represents. In other words, the degree of v

is always $|\delta(\ell(v))|$. Formally, this statement can be shown by induction. We observe that the statement is true for G_0: Then the label of v is just $\{v\}$, and the value of this cut is exactly the degree of v. Whenever we merge two vertices u and v, we keep exactly those edges that have one endpoint in the label sets $\ell(u)$ and $\ell(v)$ and the other endpoint outside. Exactly these edges are cut by $\ell(u) \cup \ell(v)$.

There can be many minimum cuts in G, but that is not guaranteed, so we don't want to consider multiple minimum cuts in our analysis. Instead we want to compare S to a fixed minimum cut. We let S^* be an arbitrary optimal solution and now always compare to this solution. Let $C^* = \delta(S^*)$ be the edges that are cut by S^*.

If the algorithm never contracts an edge from C^*, then it outputs C^* and thus succeeds. We observe that the probability that it fails in the first iteration is $|C^*|/|E_0|$. So in order to see that this probability cannot be arbitrarily close to 1, we need to know something about the number of edges in G_0.

We have the following insight. If a vertex in G_0 has degree d, then the cut $\{v\}$ has value d as we observed already. So if S^* is a minimum cut, then all vertices have a degree of at least $|C^*|$, because otherwise, there would be a better cut than S^*. We can now use a fact about graphs. If one sums up the degrees of all nodes, then every edge is counted exactly twice, thus the sum is $2 \cdot |E|$. That implies the following:

Fact 1.15. *If every node in a graph G with n nodes has degree at least d, then the graph has at least $(n \cdot d)/2$ edges.*

We have argued that every vertex in G_0 has degree at least $|C^*|$. Thus, Fact 1.15 says that G_0 has at least $(n \cdot |C^*|)/2$ edges. We can now bound the probability that the algorithm makes a mistake in the first iteration by

$$\frac{|C^*|}{|E_0|} \leq \frac{|C^*|}{(n \cdot |C^*|)/2} = \frac{2}{n}.$$

Let A_i be the event that the algorithm contracts a good edge in iteration i, i.e., an edge that is not in C^*. We have just shown that $\mathbf{Pr}(A_1) = 1 - \mathbf{Pr}(\bar{A}_1) \geq 1 - \frac{2}{n}$. The algorithm succeeds if A_i occurs for all $i \in \{1, \ldots, n-2\}$, i.e., the algorithm always contracts an edge that is not in C^*. We have a similar situation as when we analyzed dependent runs of our polynomial tester. The success probability of the `Contract` algorithm is:

$$\mathbf{Pr}(A_1 \cap \ldots \cap A_{n-2}) = \mathbf{Pr}(A_1) \cdot \mathbf{Pr}(A_2 \mid A_1) \cdot \ldots \cdot \mathbf{Pr}(A_{n-2} \mid A_1 \cap \ldots \cap A_{n-3})$$

where we use the definition of conditional probability. If we pick good edges in the first $i-1$ iterations, then there are still $|C^*|$ bad edges in iteration i, even though the total number of edges decreased. More precisely, the probability to choose an edge from C^* if we never chose an edge from C^* before is

$$\mathbf{Pr}(\bar{A}_i \mid A_1 \cap \ldots \cap A_{i-1}) = \frac{|C^*|}{|E_{i-1}|}.$$

Now we need to show that E_{i-1} still contains enough edges. So far, we only showed that $|E_0|$ is large. However, we can show a lower bound on $|E_{i-1}|$ in a similar way. We observed in the beginning of the proof that the degree of every vertex in G_i is exactly the value of the cut that we get from the label $\ell(v)$. We know that no cut can cut fewer than $|C\ |$ edges. Thus, the degree of every node in G_{i-1} is *still* at least $|C\ |$. The number of nodes in G_{i-1} is $n-(i-1)$ because we contracted $i-1$ edges. Thus, we know that $|E_{i-1}| \geq |C\ | \cdot (n-i+1)/2$ if we again use Fact 1.15. So we now know that

$$\mathbf{Pr}(\bar{A}_i \mid A_1 \cap \ldots \cap A_{i-1}) = \frac{|C\ |}{|E_{i-1}|} \leq \frac{|C\ |}{|C\ | \cdot (n-i+1)/2} = \frac{2}{n-i+1}.$$

We can now compute

$$\mathbf{Pr}(A_i \mid A_1 \cap \ldots \cap A_{i-1}) = 1 - \mathbf{Pr}(\bar{A}_i \mid A_1 \cap \ldots \cap A_{i-1}) \geq 1 - \frac{2}{n-i+1} = \frac{n-i-1}{n-i+1}.$$

Luckily, when we multiply the different terms, we observe that a lot of terms cancel out. We get:

$$\begin{aligned}
&\mathbf{Pr}(\text{Algo } \texttt{Contract} \text{ succeeds})\\
\geq\ &\mathbf{Pr}(\text{Algo } \texttt{Contract} \text{ outputs } S\ )\\
=\ &\mathbf{Pr}(A_1 \cap \ldots \cap A_{n-2}) = \mathbf{Pr}(A_1) \cdot \mathbf{Pr}(A_2 \mid A_1) \cdot \ldots \cdot \mathbf{Pr}(A_{n-2} \mid A_1 \cap \ldots \cap A_{n-3})\\
\geq\ &\left(\frac{n-2}{n}\right) \cdot \left(\frac{n-3}{n-1}\right) \cdot \left(\frac{n-4}{n-2}\right) \cdot \ldots \cdot \left(\frac{4}{6}\right) \cdot \left(\frac{3}{5}\right) \cdot \left(\frac{2}{4}\right) \cdot \left(\frac{1}{3}\right)\\
=\ &\frac{2}{n(n-1)}
\end{aligned}$$

That shows the theorem. □

A success probability of roughly $1/n^2$ does not look impressive. However, we can increase the success probability by independent runs as we did before. Assume that we repeat the algorithm n^2 times, and return the best solution that we found. Then the output is not a minimum cut if no run found a minimum cut. Thus, the probability that the algorithm fails is at most

$$\prod_{i=1}^{n^2} \mathbf{Pr}(\text{run } i \text{ fails}) \leq \prod_{i=1}^{n^2} \left(1 - \frac{2}{n(n-1)}\right) \leq \prod_{i=1}^{n^2} \left(1 - \frac{2}{n^2}\right) = \left(\left(1 - \frac{2}{n^2}\right)^{\frac{n^2}{2}}\right)^2 \leq \left(\frac{1}{e}\right)^2 = \frac{1}{e^2}.$$

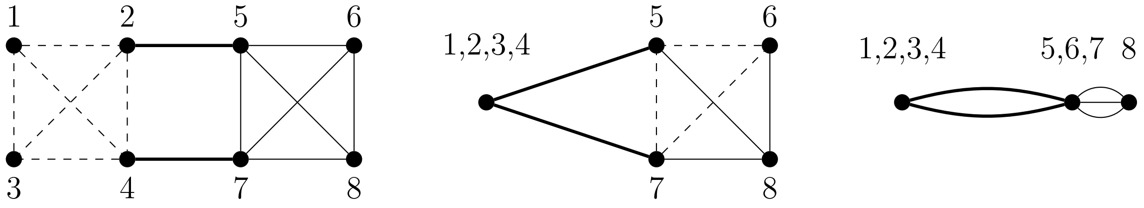
For the last step, we used the useful inequality $(1-\frac{t}{x})^x \leq \frac{1}{e^t}$ which holds for $x, t \quad \mathbb{R}$ with $x \geq 1$ and $|t| \leq x$ (see 3.6.2 in [Mit70]). The failure probability is now $1/e^2$, which is less than 0.14. We can decrease it to any given δ by increasing the number of independent runs appropriately, as we did for the polynomial tester.

By using appropriate data structures, one edge contraction of the `Contract` algorithm can be implemented in time $O(n)$. It is also possible to select an edge from the remaining edges in time $O(n)$. Since there are $O(n)$ edge contractions, the overall running time of `Contract` is bounded by $O(n^2)$. More detailed instructions how to show these statements are given in Problem 10.9 and Problem 10.10 in [MR95].

Fact 1.16. *Algorithm `Contract`*$(\mathbf{G})$ *can be implemented to run in time* $O(n^2)$.

We conclude that we can get an algorithm with failure probability $1/e^2$ and running time $O(n^4)$. Next, we see how to significantly decrease the running time.

The FastCut algorithm We discuss an improvement of the `Contract` algorithm that we call `FastCut`. It was published by Karger and Stein in [KS96]. The following pictures show snapshots (G_0, G_3 and G_5) of an example run of the `Contract` algorithm. The dashed edges are contracted in the steps that happen between the pictures (either because the edge itself is contracted or because it is parallel to an edge that is contracted).



An optimal solution is $\{1, 2, 3, 4\}$ which cuts the two bold edges C . We observe that in G_0, there are 14 edges in total, and the probability to choose one of the two bold edges is only 1/7. However, after contracting five edges not in C , the example run obtains G_5 with only five edges. Now the probability to contract an edge from C is 2/5. Additionally, the probability to get to G_5 is already small because the algorithm had to make five good choices already. The failure probability gets higher and higher the longer the `Contract` algorithm runs.

We also see this effect when we look at the proof of Theorem 1.14 again. Assume that we stop the `Contract` algorithm when the graph has t nodes left, i.e., at G_{n-t}. Then the probability that the algorithm contracted no edge from C is

$$\begin{aligned}
&\mathbf{Pr}(A_1 \cap \ldots \cap A_{n-t}) = \mathbf{Pr}(A_1) \cdot \mathbf{Pr}(A_2 \mid A_1) \cdot \ldots \cdot \mathbf{Pr}(A_{n-t} \mid A_1 \cap \ldots \cap A_{n-t-1})\\
\geq\ & \left(\frac{n-2}{n}\right) \cdot \left(\frac{n-3}{n-1}\right) \cdot \left(\frac{n-4}{n-2}\right) \cdot \ldots \cdot \left(\frac{t+2}{t+4}\right) \cdot \left(\frac{t+1}{t+3}\right) \cdot \left(\frac{t}{t+2}\right) \cdot \left(\frac{t-1}{t+1}\right)\\
=\ & \frac{t(t-1)}{n(n-1)}.
\end{aligned}$$

We see that the bound gets very small when the number of remaining nodes is small, e.g., for $t = 2$, which matches the original `Contract` algorithm. The key observation is that we can reduce the number of nodes significantly, for example to around $(3/4)n$, without having a high chance of failure: For $t = 1 + (3/4)n$, the success probability is still bounded from below by

$$\frac{t(t-1)}{n(n-1)} \geq \frac{(t-1)^2}{n^2} \geq \frac{(3/4)^2 n^2}{n^2} = \frac{9}{16} \geq \frac{1}{2}.$$

The value of t can be made a little smaller. For $t = 1 + n/\sqrt{2} \approx 0.7n$, we get

$$\frac{t(t-1)}{n(n-1)} \geq \frac{(t-1)^2}{n^2} \geq \frac{n^2}{\sqrt{2}^2 n^2} = \frac{1}{2}.$$

Setting $t = 1 + n/\sqrt{2}$ will turn out beneficial for the running time, so we choose this value. The algorithm `Contract(`G, t`)` is the same as `Contract(`G`)` except that the 2 in step 2 is replaced by t, that step 6 is deleted, and that the algorithm returns the current graph G instead of a cut. We just argued that the probability that Contract(G, t) does not contract an edge from C is at least $1/2$ for $t = 1 + n/\sqrt{2}$.

The idea behind the algorithm FastCut(G) is to include the repetitions into the algorithm instead of simply repeating Karger's Contract(G) algorithm as a whole. This makes sense because the failure probability increases during the algorithm. Entangling the repetitions with the algorithm opens the possibility to repeat the more dangerous steps more often than the safer steps when the graph still has many edges. The algorithm uses Contract(G, t) as a subroutine. It uses two independent calls to this procedure to reduce G to graphs H_1 and H_2 with $1+ n/\sqrt{2}$ nodes. Then it recursively solves the minimum cut problem first on H_1 and second on H_2 by calling FastCut(H_1) and FastCut(H_2), the better result is then returned. Observe that each recursive call starts with two independent reduction steps. Thus, in the first level of the recursion, the previously computed H_1 is twice reduced to roughly 70 % of its nodes, which is 49% of the original number of nodes of G. The deeper we are in the recursion tree, the more independent reduced graphs are computed in different branches. When the number of nodes n falls below 7, then $1 + n/\sqrt{2}$ is no longer smaller than n (observe that $1 + 6/\sqrt{2} \geq 1 + 4.24 = 6$). When the recursion reaches this point, the algorithm simply solves the problem optimally. For graphs with less than 7 nodes, this can be done in constant time.

```
FastCut($\mathbf{G} = (\mathbf{V}, \mathbf{E})$)

if $n := |V| \geq 7$
    Set $t := 1 + n/\sqrt{2}$
    $H_1 =$ Contract$(G, t)$; $H_2 =$ Contract$(G, t)$;
    $S_1 =$ FastCut$(H_1)$; $S_2 =$ FastCut$(H_2)$;
    if $|\delta(S_1)| \leq |\delta(S_2)|$ then
        return $S_1$
    else
        return $S_2$
else
    Compute an optimal solution $S$ by enumerating all solutions
    return $S$
```

Since the algorithm is recursive, determining its running time requires the analysis of a recurrence. Let $T(n)$ be the running time for a graph with n nodes. We know from Fact 1.16 that there is a constant $c > 0$ such that the original Contract(G) needs at most $c \cdot n^2$ time. Since Contract(G,t) only stops earlier, its running time is bounded by $c \cdot n^2$ as well.

Thus, FastCut(G) spends at most $2cn^2$ time for the calls to Contract(G, t). Aside from that, all steps need some constant $c > 0$ time, except for the recursive calls. When $n < 7$, then the running time is bounded by some constant c . We get that the running time is bounded by the following recurrence:

$$T(n) \leq \begin{cases} c & \text{if } n < 7, \\ 2 \cdot T(\; n/\sqrt{2} \;) + 2cn^2 + c & \text{if } n \geq 7. \end{cases}$$

Solving this recurrence is a standard task and we leave the details as an exercise.

Lemma 1.17. *The running time of algorithm* `FastCut(`G`)` *is* $O(n^2 \log n)$.

Proof Sketch. To understand how the running time builds up, we do a proof sketch where we ignore the ceiling function and prefer intuition over a formal proof. We look at the workload that the algorithm does by looking at the computation tree. The source node corresponds to the initial call with n nodes. On level i, every call gets a graph of size $n/(\sqrt{2})^i$ as input. The height h of the tree satisfies h $\Theta(\log n)$. There are 2^i calls on level i, each of them consisting of two recursive calls plus $cn^2/((\sqrt{2})^i)^2 = cn^2/2^i$ overhead for some constant $c > 0$. Thus, the total overhead on level i is $2^i \cdot cn^2/2^i = cn^2$. This is true for all levels except the leaf level. On the leaf level, there are less than n nodes, each of them having a workload of c . The leaf level thus has a workload of $O(n)$ $O(n^2)$. So, the workload added up for all h levels lies in $\Theta(n^2 \log n)$.

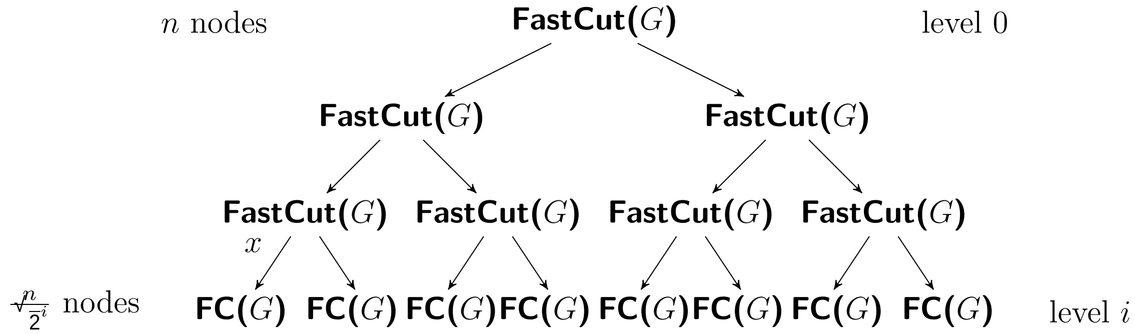
Observe that it is crucial that the number of nodes is reduced to $1/\sqrt{2}$ time the input number before the recursive calls. If it was reduced to a larger fraction, for example (3/4) times the number of input nodes, then the computation done by one level would increase for increasing i and the overall running time would lie in $\omega(n^2 \log n)$. □

Lemma 1.17 gives a much better bound than the running time bound we got for the repeated `Contract` algorithm. Now the important part is to show that the `FastCut` algorithm has a good success probability.

Theorem 1.18. *The algorithm* `FastCut` *always outputs a cut. With a probability of at least* $\Omega(1/\log n)$*, this cut is a minimum cut.*

Proof. As before, we fix an optimal solution S and let C $:= \delta(S)$ be the edges that are cut by S . We show an upper bound on the probability that the `FastCut` algorithm does not output a minimum cut.

The algorithm is recursive. Thus, the success probability follows a recurrence relation. Assume that we are at a call of `FastCut` on level i of the recursion, and C is still intact (no edge from C has been contracted). By level i of the recursion we mean all calls where the input graph has already been reduced i times. The first call `FastCut(`G`)` is (the only call) on level 0.


n nodes
FastCut(G)
level 0
FastCut(G)
FastCut(G)
FastCut(G) FastCut(G) FastCut(G) FastCut(G)
x
$\frac{n}{\sqrt{2^i}}$ nodes
FC(G) FC(G) FC(G) FC(G) FC(G) FC(G) FC(G) FC(G)
level i

We want to analyze the probability that a `FastCut` call on level i outputs an optimal solution, under the assumption that this is still possible. We let $P(i)$ be a lower bound on the probability that a call on level i outputs a minimum cut under the assumption that C is in its input graph.

`FastCut` computes a graph H_1 by calling `Contract(`G, t`)` with $t = 1 + n/\sqrt{2}$. We already argued that this call contracts no edge from C with probability at least 1/2 because of the choice of t. Later, it recursively computes a solution S_1 based on H_1. The probability that H_1 still contains C and then the recursive call is successful is at least $P(i+1)/2$ because these two events are independent. The same is true for S_2. The probability that both S_1 and S_2 fail to be a minimum cut is now bounded from above by $(1 - P(i+1)/2)^2$, which means that

$$P(i) \geq 1 - \left(1 - \frac{P(i+1)}{2}\right)^2.$$

It might be a bit unintuitive that the recurrence goes 'up'. To change this, let h be the maximum level, i.e., the level where all calls are executed without any further recursion. We redefine $P(i)$ as the probability that a call on level $h - i$ outputs a minimum cut under the assumption that C is in its input graph. Then $P(0) = 1$ because of the assumption that C is still a feasible solution and because the problem is solved optimally. Furthermore, by our above argumentation,

$$P(i) \geq 1 - \left(1 - \frac{P(i-1)}{2}\right)^2$$

for $i \quad \{1, \ldots, h\}$.

We show that $P(i) \geq 1/(i+1)$ for all $i \quad \{0, \ldots, h\}$ by induction. The base case is $i = 0$, and the statement holds in this case because $P(0) = 1 = 1/(0+1)$. For $i \quad \{1, \ldots, h\}$ we get

$$\begin{aligned} P(i) \geq & 1 - \left(1 - \frac{P(i-1)}{2}\right)^2 \geq 1 - \left(1 - \frac{1}{2i}\right)^2 = 1 - \left(\frac{2i-1}{2i}\right)^2 \\ = & \frac{4i^2}{4i^2} - \frac{4i^2 - 4i + 1}{4i^2} = \frac{4i-1}{4i^2} = \frac{1}{i} - \frac{1}{4i^2} \overset{i\geq 1}{\geq} \frac{1}{i} - \frac{1}{i(i+1)} = \frac{1}{i+1}. \end{aligned}$$

The success probability of the first `FastCut` call is $P(h)$. On level i, the calls get input graphs with $n/(\sqrt{2})^i$ nodes. Thus, on level $\log_{\sqrt{2}} n/6$, the number of nodes is at most

$$\frac{n}{\sqrt{2}^{\log_{\sqrt{2}} n/6}} \leq \frac{n}{n/6} = 6.$$

We conclude that h $O(\log n)$ which proves the theorem. □

As before, we can additionally increase the success probability by independent repetitions. For a constant success probability, we need $\Theta(\log n)$ repetitions, yielding an overall running time of $O(n^2 \log^2 n)$.

1.3.2 Reservoir Sampling

We now do an excursion to the data stream model to see a somewhat different setting. Data streams frequently occur in today's algorithmic challenges, when monitoring internet traffic or the output of a sensor, the stock market or another source that never stops to produce more and more information. Data streams cannot be stored (completely), and the length of the stream is unknown (or thought of as being infinite). In particular, this means that algorithms working with a data stream as the input data have no random access to the data. They can read the data once, store some information about it, but they cannot go back and forth.

There are many models for data stream algorithms. A detailed introduction and overview on different models is for example given in [Mut05]. We do worst-case analysis in this part of the lecture, thus we also assume a worst-case scenario here: The data arrives in an arbitrary order, possibly the worst order for the algorithm we design. In this section, we consider the following simple data stream model: The stream consists of distinct integers

$$a_1, a_2, a_3, \ldots$$

in arbitrary order and we do not know the length of the stream (it might be infinite).

Sampling an element from a stream. In the data stream model, easy tasks can be a challenge. In this section, we want to solve the basic problem of choosing elements uniformly at random from the stream (choosing elements uniformly at random from some ground set is also called 'sampling'). More precisely, our first task is to design an algorithm that accomplishes the following: It always stores one number s from the stream. After seeing the ith element a_i, the probability that s is equal to a_j should be $1/i$ for all j $\{1, \ldots, i\}$. It is a bit surprising that we can actually achieve this goal, and in a fairly straightforward manner. Our algorithm does the following:

```
ReservoirSampling
After seeing $a_1$, set $s = a_1$.
When seeing $a_i$:
    With probability $1/i$, replace $s$ by $a_i$.
    Else (thus, with probability $(i-1)/i$), do nothing.
```

Lemma 1.19. *ReservoirSampling satisfies for any $i \geq 1$: After seeing and processing a_i, it holds that*

$$\mathbf{Pr}(s = a_j) = \frac{1}{i} \quad j \quad \{1, \ldots, i\}.$$

Proof. The event '$s = a_j$' occurs if a_j is chosen to replace s after seeing a_j and it is never replaced by the $i - j$ subsequent elements (i.e., the algorithm chose to do nothing $i - j$ times). Thus,

$$\mathbf{Pr}(s = a_j) = \frac{1}{j} \cdot \frac{j}{j+1} \cdot \frac{j+1}{j+2} \cdot \frac{j+2}{j+3} \cdot \ldots \cdot \frac{i-2}{i-1} \cdot \frac{i-1}{i} = \frac{1}{i},$$

which proves the lemma. □

Sampling more than one element. Our second task is to sample s elements from the stream, where s can be greater than one. First we observe that this reduces to our previous algorithm if we allow that an element is sampled multiple times. In order to get a set S with s elements from the stream where multiples are allowed, we simply run s copies of our above algorithm. Then every multi-set S with $|S| = s$ has equal probability to be the current multi-set after seeing a_i, namely probability $1/i^s$.

The challenge of this paragraph is to always have a set S of distinct elements which is chosen uniformly at random/sampled from the set of all possible subsets of $\{a_1, \ldots, a_i\}$. The algorithm that achieves this is the algorithm `R` published by Vitter in [Vit85]. We still call the algorithm `ReservoirSampling` (for s elements):

```
ReservoirSampling
After seeing the first $s$ elements, set $S = \{a_1, a_2, \ldots, a_s\}$.
When seeing $a_i$:
    With probability $\frac{i-s}{i}$, do nothing.
    Else (thus, with probability $\frac{s}{i}$):
        Choose an item $x$ uniformly at random from $S$.
        Replace $x$ in $S$ by $a_i$.
```

We show that every set with s elements has equal probability to be S at any point in the stream. Observe that there are $\binom{i}{s}$ subsets of size s of $\{a_1, \ldots, a_i\}$. Thus, after seeing a_i, the probability for any specific subset should be $1/\binom{i}{s}$.

Theorem 1.20. *`ReservoirSampling` (for s elements) satisfies for any $i \geq s$: After seeing and processing a_i, it holds that*

$$\mathbf{Pr}(S = T) = \frac{1}{\binom{i}{s}}$$

for every $T \quad \{a_1, \ldots, a_i\}$ with $|T| = s$.

Proof. We show the statement by induction over i.

$i = s$: Since the algorithm chooses the first s elements as the initial set S, we have $\mathbf{Pr}(S = T) = 1 = 1/\binom{s}{s}$ for $T = \{a_1, \ldots, a_s\}$, the only possible subset of $\{a_1, \ldots, a_s\}$ that satisfies $|T| = s$.

$i > s$: We want to show $\mathbf{Pr}(S = T) = 1/\binom{i}{s}$ for all possible sets, so let $T \quad \{a_1, \ldots, a_i\}$ be an arbitrary fixed set. We let S be the set that the algorithm had after seeing a_{i-1} and before seeing a_i. It then processed a_i, maybe replaced a uniformly chosen element x by a_i and the current set is set S. We analyze the probability that $S = T$ by distinguishing two cases.

a) $a_i \quad T$.
If a_i is not in T, then the algorithm can only have $S = T$ if it already had T after seeing a_{i-1}. Thus let A be the event that $S \ = T$. By the induction hypothesis, we know that

$$\mathbf{Pr}(A) = \frac{1}{\binom{i-1}{s}}.$$

If A occurred, it might still be that $S = T$ if the algorithm chose to replace an item. Thus let B be the event that the algorithm replaces an item. By the algorithm definition, it is

$$\mathbf{Pr}(B) = \frac{s}{i}.$$

We observe that $S = T$ occurs exactly if A occurs and B does not occur. Since the events A and $\bar{B}$ are independent, we can conclude that

$$\begin{aligned}\mathbf{Pr}(S = T) &= \mathbf{Pr}(A \cap \bar{B}) = \mathbf{Pr}(A) \cdot \mathbf{Pr}(\bar{B}) \\ &= \frac{1}{\binom{i-1}{s}} \cdot \frac{i-s}{i} = \frac{s!(i-s-1)!}{(i-1)!} \cdot \frac{i-s}{i} = \frac{s!(i-s)!}{i!} = \frac{1}{\binom{i}{s}}.\end{aligned}$$

b) $a_i \quad T$.
In this case, we can only get T if we had all elements except a_i in S already. Thus, let $R = T \setminus \{a_i\}$ be these elements and let C be the event that $R \quad S$. Notice that there are $(i-1) - (s-1) = i - s$ subsets of $\{a_1, \ldots, a_{i-1}\}$ with s elements that contain R. For each of these subsets, the probability that it is S is $1/\binom{i-1}{s}$ by the induction hypothesis. Thus we get that

$$\mathbf{Pr}(C) = \frac{i-s}{\binom{i-1}{s}}.$$

We again use the event B that the algorithm chose to replace an item by a_i, we recall that $\mathbf{Pr}(B) = s/i$. Finally, we need one event more. Let D be the event that $C \cap B$ occurs and that additionally, the item x is the one item in $T \setminus R$. Thus, $\mathbf{Pr}(S = T) = \mathbf{Pr}(D)$. Observe that the probability $\mathbf{Pr}(D \mid B \cap C)$ is $1/s$: If we already know that an element is replaced, and we also already know that C occurred, then the probability that the replaced item is one specific item is just $1/|S| = 1/s$. We can now conclude that

$$\begin{aligned}\mathbf{Pr}(S = T) &= \mathbf{Pr}(B \cap C \cap D) = \mathbf{Pr}(B \cap C) \cdot \mathbf{Pr}(D \mid B \cap C) \\ &= \mathbf{Pr}(B) \cdot \mathbf{Pr}(C) \cdot \mathbf{Pr}(D \mid B \cap C) \\ &= \frac{s}{i} \cdot \frac{i-s}{\binom{i-1}{s}} \cdot \frac{1}{s} \\ &= \frac{1}{i} \cdot \frac{i-s}{\binom{i-1}{s}} \\ &= \frac{1}{\binom{i}{s}}.\end{aligned}$$

For the last step, observe that we computed the same term above as well. We have shown that the theorem is true.

□

Chapter 2

Evaluating Outcomes of a Random Process

A ***random variable*** is a very useful concept: It allows us to concentrate our attention to a numerical evaluation of a random process that we are interested in, and to compute properties of this evaluation, for example the expected value. When analyzing randomized algorithms, we are often interested in the running time, which can be a random variable, or in the expected quality (in case of an optimization problem). We define the basics for using random variables and computing expected values. An interesting special case are integer valued random variables. We will see a useful way to compute expected values when the random variable is non-negative and attains integer values. Then, we discuss two even more specialized but very important types of integer valued random variables: Geometrically and binomially distributed random variables.

Section 2.1 and Section 2.2 cover selected content from 2.1-2.4 in [MU05]. Section 2.3.1 corresponds to 2.5 in [MU05].

2.1 Random Variables and Expected Values

A random variable is a mapping that assigns a value to every elementary event. We have already used random variables implicitly: the value of a die, the sum of multiple dice or the number of times that a coin flip shows heads.

Definition 2.1. *Let* $(\Omega, \mathbf{Pr})$ *be a discrete probability space. A* discrete random variable *is a mapping* $X : \Omega \rightarrow$ R *that assigns a real number to every elementary event.*

We also already used random variables in the definition of events, for example that the sum of the values of two dice is eight. The event that a random variable X takes value a is usually abbreviated by $X = a$. The precise definition of this event is $\{\omega \quad \Omega \mid X(\omega) = a\}$. We will also use this abbreviation, in particular we use

$$\mathbf{Pr}(X = a) = \mathbf{Pr}(\{\omega \quad \Omega \mid X(\omega) = a\}).$$

Analogously, we use for $R \quad$ R:

$$\mathbf{Pr}(X \quad R) = \mathbf{Pr}(\{\omega \quad \Omega \mid X(\omega) \quad R\}).$$

Example 2.2. *We have a look at three examples for random variables.*

1. *The sum of two dice, modeled by the probability space $(\Omega, \mathbf{Pr})$ with $\Omega = \{1, \ldots, 6\}^2$ and $\mathbf{Pr}(\omega) = 1/36$ for all $\omega \quad \Omega$, is formally defined by $X : \Omega \to \{1, \ldots, 12\}$ with $X(\omega) = \omega_1 + \omega_2$ for all $\omega = (\omega_1, \omega_2) \quad \Omega$. We observe that $\mathbf{Pr}(X = 1) = 0$, $\mathbf{Pr}(X = 2) = 1/36$, and $\mathbf{Pr}(X = 7) = 1/6$.*
2. *Let $(\Omega, \mathbf{Pr})$ be a probability space for modeling ten independent coin flips, i.e., $\Omega = \{H, T\}^{10}$, $\mathbf{Pr}(\omega) = 1/1024$ for all $\omega \quad \Omega$. Let $X : \Omega \to \{0, 1, 2, \ldots, 10\}$ be the number of times that the flip came up heads. We observe that $\mathbf{Pr}(X = 0) = 1/1024$ and $\mathbf{Pr}(X = 1) = 10/1024$. A little more consideration lets us observe that $\mathbf{Pr}(X = 2) = 45/1024$.*
3. *We model the following random experiment: A coin is tossed until the first time that we see tails. We have $\Omega = \{T, HT, HHT, HHHT, \ldots\} = \{H^{i-1}T \mid i \geq 1\}$ as a countably infinite discrete probability space. The probability of the elementary event to get $i - 1$ times head and then tails for the first time is $\mathbf{Pr}(H^{i-1}T) = 1/2^i$. Observe that*

   $$\sum_{i=1}^{\infty} \frac{1}{2^i} = -1 + \sum_{i=0}^{\infty} \frac{1}{2^i} = 2 - 1 = 1.$$

   We define the random variable $X : \Omega \to$ N by setting $X(H^{i-1}T) = i$ for all $H^{i-1}T \quad \Omega$. By definition, it is clear that $\mathbf{Pr}(X = i) = 1/2^i$. Assume that Alice and Bob bet whether the first tails will occur after an odd or even number of coin tosses. If the number is odd, then Alice wins i points, and if it is even, she loses i points; Bobs result is inverse. Thus Alice's gain is described by the random variable $Y : \Omega \to$ N with

   $$Y(H^{i-1}T) = \begin{cases} i & \text{if } i \text{ is odd} \\ -i & \text{if } i \text{ is even,} \end{cases}$$

   and Bobs gain is described by $-Y$.

We call random variables independent if knowing the value of one random variable gives no information about the value of the other variable. Given Definition 1.7, it is straightforward to obtain the following definition.

Definition 2.3. *Let $(\Omega, \mathbf{Pr})$ be a discrete probability space, let $X, Y : \Omega \to$ R be discrete random variables. We say that X and Y are* independent *if*

$$\mathbf{Pr}((X = x) \cap (Y = y)) = \mathbf{Pr}(X = x) \cdot \mathbf{Pr}(Y = y)$$

holds for all $x, y \quad$ R. A family $X_1, \ldots, X_\ell : \Omega \to$ R of discrete random variables is independent *if*

$$\mathbf{Pr}\Big(\bigcap_{i \quad I}(X_i = x_i)\Big) = \prod_{i \quad I} \mathbf{Pr}(X_i = x_i)$$

holds for every $I \quad \{1, \ldots, \ell\}$ and all $x_i \quad$ R$, i \quad I$.

Since Ω is finite or countably infinite, the image of X is also finite or countably infinite. We name the image $R(X)$. When $R(X)$ is finite, then the expected value is the (weighted) average of the values in $R(X)$. When $R(X)$ is countably infinite, we can think of the expected value in a similar way. However, in this case it is possible that the expected value diverges.

Definition 2.4. *Let* $(\Omega, \mathbf{Pr})$ *be a discrete probability space and let* $X : \Omega \to \mathrm{R}$ *be a discrete random variable. We say that the expected value of* X exists *iff the series*

$$\sum_{\omega\ \Omega} |X(\omega)| \cdot \mathbf{Pr}(\{\omega\}) = \sum_{x\ R(X)} |x| \cdot \mathbf{Pr}(X = x)$$

converges. If the expected value exists, then it is defined as

$$\mathbf{E}[X] = \sum_{\omega\ \Omega} X(\omega) \cdot \mathbf{Pr}(\{\omega\}) = \sum_{x\ R(X)} x \cdot \mathbf{Pr}(X = x).$$

We observe that if $\mathbf{E}[X]$ exists, the (potentially infinite) sum $\sum_{\omega\ \Omega} X(\omega) \cdot \mathbf{Pr}(\{\omega\})$ converges absolutely. It is thus well-defined even though we did not specify the ordering of its terms.

Example 2.5. *We continue Example 2.2.*

1. *When* X *is the sum of two independent dice, then its expected value is*

$$\mathbf{E}[X] = \sum_{x=2}^{12} x \cdot \mathbf{Pr}(X = x) = 2 \cdot \frac{1}{36} + 3 \cdot \frac{2}{36} + 4 \cdot \frac{3}{36} + \ldots + 11 \cdot \frac{2}{36} + 12 \cdot \frac{1}{36} = 7.$$

2. *For the random variable* X *that models the number of heads in ten independent coin flips, computing the expected value is somewhat tedious. Intuitively, we expect to see heads half of the time. Indeed, it turns out that*

$$\mathbf{E}[X] = 0 \cdot \frac{1}{1024} + 1 \cdot \frac{10}{1024} + 2 \cdot \frac{45}{1024} + \ldots + 10 \cdot \frac{1}{1024} = 5.$$

3. *What is the expected value of Alice's gain? Does the expected value exist? We can write* $Y(H^{i-1}T)$ *as* $(-1)^{i-1} \cdot \frac{1}{2^i}$. *To decide if the expected value exists, we have to decide whether*

$$\sum_{i=1}^{\infty} |(-1)^{i-1} i| \cdot \frac{1}{2^i} = \sum_{i=1}^{\infty} \frac{i}{2^i}$$

converges. We might know that this series converges or we can apply a convergence test to the series $\sum_{i=1}^{\infty} a_i$ *with* $a_i = \frac{i}{2^i}$ *to prove it (for example, to apply the ratio test, observe that* $\frac{i+1}{2^{i+1}} / (\frac{i}{2^i}) = 1/2 + 1/(2i)$ *and thus* $\lim_{i\to\infty} |a_{i+1}/a_i| = 1/2 < 1$*), and obtain that the expected value exists. When we actually compute the expected value, it might be surprising that it is not zero. We can compute some intermediate terms to get an intuition that the alternating series*

$$\mathbf{E}[Y] = \sum_{i=1}^{\infty} (-1)^{i-1} i \cdot \frac{1}{2^i}$$

converges to a value above zero:

$$
\begin{aligned}
&a_1 = 1/2, a_2 = 2/4, a_3 = 3/8, a_4 = 4/16, a_5 = 5/17, a_6 = 6/18\\
&a_1 = 1/2, a_1 - a_2 = 0, a_1 - a_2 + a_3 = 3/8, a_1 - a_2 + a_3 - a_4 = 1/8,\\
&a_1 - a_2 + a_3 - a_4 + a_5 = 9/32, a_1 - a_2 + a_3 - a_4 + a_5 - a_6 = 3/16.
\end{aligned}
$$

In fact, the sum of the series is $2/9$! Thus, Alice wins $2/9$ points in expectation, and Bob loses $2/9$ points.

Assume that we have two random variables $X, Y : \Omega \to \mathsf{R}$ in $(\Omega, \mathbf{Pr})$. By stating $X \leq Y$ we mean that $X(\{\omega\}) \leq Y(\{\omega\})$ for all $\omega \quad \Omega$.

Observation 2.6. *If $X, Y : \Omega \to \mathsf{R}$ satisfy $X \leq Y$, then $\mathbf{E}[X] \leq \mathbf{E}[Y]$ given that the expected values of X and Y exist.*

Proof. The observation follows by the definition of the expected value since

$$\mathbf{E}[X] = \sum_{\omega \quad \Omega} X(\omega) \cdot \mathbf{Pr}(\{\omega\}) \leq \sum_{\omega \quad \Omega} Y(\omega) \cdot \mathbf{Pr}(\{\omega\}) = \mathbf{E}[Y]. \qquad \square$$

It is tempting to say that if $X \leq Y$ and the expected value of X does not exist, then also the expected value of Y does not exist. However, this assertion is not true: For instance, it could be that the expected value of X diverges to $-\infty$ while Y is zero everywhere.

Computations with the expected value often become a lot easier when we use the fact that the expected value is linear. We show this and more important rules for the expected value below. Intuitively, we know what the sum or product of a random variable is, or how we multiply constants or add constants. Formally, we define three types of composed random variables. Let $X, Y : \Omega \to \mathsf{R}$ be two random variables in a discrete probability space $(\Omega, \mathbf{Pr})$.

- For all $c \quad \mathsf{R}$, the random variable $(cX) : \Omega \to \mathsf{R}$ is defined by $(cX)(\omega) = c \cdot X(\omega)$ for all $\omega \quad \Omega$.
- The random variable $(X + Y) : \Omega \to \mathsf{R}$ is defined by $(X + Y)(\omega) = X(\omega) + Y(\omega)$ for all $\omega \quad \Omega$.
- The random variable $(X \cdot Y) : \Omega \to \mathsf{R}$ is defined by $(X \cdot Y)(\omega) = X(\omega) \cdot Y(\omega)$ for all $\omega \quad \Omega$.

In all three cases, we usually omit the brackets for a shorter notation when possible.

Theorem 2.7. *Let $(\Omega, \mathbf{Pr})$ be a discrete probability space, let $X, Y : \Omega \to \mathsf{R}$ be discrete random variables. If the expected values of X and Y exist, then it also holds that:*

1. *The expected value of cX exists for all $c \quad \mathsf{R}$ and $\mathbf{E}[cX] = c\mathbf{E}[X]$.*
2. *The expected value of $X + Y$ exists and $\mathbf{E}[X + Y] = \mathbf{E}[X] + \mathbf{E}[Y]$.*
3. *If X and Y are independent, then the expected value of $X \cdot Y$ exists and $\mathbf{E}[X \cdot Y] = \mathbf{E}[X] \cdot \mathbf{E}[Y]$.*

Proof. Statements 1 and 2 are proven similarly, we prove 1 as an example. First we observe that by the definition of cX, we have

$$\begin{aligned}\sum_{\omega \in \Omega} |(cX)(\omega)| \cdot \mathbf{Pr}(\{\omega\}) &= \sum_{\omega \in \Omega} |c \cdot X(\omega)| \cdot \mathbf{Pr}(\{\omega\}) \\ &= \sum_{\omega \in \Omega} (|c| \cdot |X(\omega)|) \cdot \mathbf{Pr}(\{\omega\}) \\ &= |c| \cdot \sum_{\omega \in \Omega} |X(\omega)| \cdot \mathbf{Pr}(\{\omega\}).\end{aligned}$$

The expected value of X exists, so $\sum_{\omega \in \Omega} |X(\omega)| \cdot \mathbf{Pr}(\{\omega\})$ converges. Since c is a constant, this means that $\sum_{\omega \in \Omega} |(cX)(\omega)| \cdot \mathbf{Pr}(\{\omega\})$ converges as well, so the expected value of cX exists. We can compute it in a similar fashion and see that

$$\sum_{\omega \in \Omega} (cX)(\omega) \cdot \mathbf{Pr}(\{\omega\}) = c \cdot \sum_{\omega \in \Omega} X(\omega) \cdot \mathbf{Pr}(\{\omega\}) = c\mathbf{E}[X].$$

For statement 3, we apply the definition of $X \cdot Y$ and the fact that X and Y are independent to obtain that

$$\begin{aligned}\sum_{\omega \in \Omega} |(X \cdot Y)(\omega)| \cdot \mathbf{Pr}(\{\omega\}) &= \sum_{\omega \in \Omega} |X(\omega) \cdot Y(\omega)| \cdot \mathbf{Pr}(\{\omega\}) \\ &= \sum_{x \in R(X)} \sum_{y \in R(Y)} |xy| \cdot \mathbf{Pr}((X = x) \cap (Y = y)) \\ &\leq \sum_{x \in R(X)} \sum_{y \in R(Y)} |x| \cdot |y| \cdot \mathbf{Pr}(X = x) \cdot \mathbf{Pr}(Y = y) \\ &= \sum_{x \in R(X)} |x| \cdot \mathbf{Pr}(X = x) \sum_{y \in R(Y)} |y| \cdot \mathbf{Pr}(Y = y).\end{aligned}$$

We know that the expected value of Y exists, thus $\sum_{y \in R(Y)} |y| \cdot \mathbf{Pr}((Y = y))$ converges to a constant g. Thus,

$$\begin{aligned}\sum_{\omega \in \Omega} |(X \cdot Y)(\omega)| \cdot \mathbf{Pr}(\{\omega\}) &\leq \sum_{x \in R(X)} |x| \cdot \mathbf{Pr}(X = x) \cdot g \\ &= g \cdot \sum_{x \in R(X)} |x| \cdot \mathbf{Pr}(X = x).\end{aligned}$$

The expected value of X exists, so $\sum_{x \in R(X)} |x| \mathbf{Pr}((X = x))$ converges. Thus, $\sum_{\omega \in \Omega} |(X \cdot Y)(\omega)| \cdot \mathbf{Pr}(\{\omega\})$ converges and the expected value of $X \cdot Y$ exists. We can compute it in a similar fashion and obtain that $\mathbf{E}[X \cdot Y] = \mathbf{E}[X] \cdot \mathbf{E}[Y]$. Notice that we crucially needed the fact that X and Y are independent. □

Statement 2 from Theorem 2.7 says that the expected value is additive, the rule by itself is often referred to as ***linearity of expectation***. Statement 1 says that the expected value is also homogeneous, so the expected value is indeed a linear function. Notice that rule 2 holds for all random variables over the same probability space, even for dependent random variables! Statement 3 is very useful when random variables are independent. Observe that for dependent random variables X and Y, it is possible that $\mathbf{E}[X]$ and $\mathbf{E}[Y]$ exist, but $\mathbf{E}[X \cdot Y]$ does not exist.

Example 2.8. *Let us construct an example where* $\mathbf{E}[X]$ *exists, but* $\mathbf{E}[X \cdot X]$ *does not exist. We need a probability space with a countably infinite set* Ω *and choose* $\Omega = \mathbb{N}$*. Our random experiment will be to draw a random number from* $\mathbb{N}$*. Over the infinite set of all positive integers, we cannot choose uniformly at random. If we choose* i *with probability* $1/(2^i)$*, then we know that this defines a probability measure (this space naturally arises in our coin flip examples). An alternative is to pick* i *with a probability proportional to* $1/i^4$*. In the following, we will use some facts from mathematical analysis without worrying about how to obtain them. The series*

$$\sum_{i=1}^{\infty} \frac{1}{i^4}$$

converges to $\pi^4/90$*, so in order to get a probability measure, we choose number* i *with probability* $90/(\pi^4 i^4)$*, because then we have* $\mathbf{Pr}(\Omega) = \sum_{i=1}^{\infty} \frac{90}{\pi^4 i^4} = 1$*. Now consider the random variable* X *defined by* $X(i) = i^2$*. The expected value of* X *exists because*

$$\mathbf{E}[X] = \sum_{i=1}^{\infty} \frac{90 i^2}{\pi^4 i^4} = \frac{90}{\pi^4} \sum_{i=1}^{\infty} \frac{1}{i^2}$$

converges (to $15/\pi^2$*). However, the expected value of the random variable* $X \cdot X$ *does not exist because the series*

$$\sum_{i=1}^{\infty} \frac{90 i^4}{\pi^4 i^4} = \sum_{i=1}^{\infty} \frac{90}{\pi^4}$$

diverges.

2.1.1 Non-negative Integer Valued Random Variables

We are often interested whether a specific event occurs or not. Is my algorithm successful? Did the die roll give a six? It can be very useful to express this by an *indicator random variable*. Such a variable can only take the values 0 and 1 depending on whether the event in question occurred or not – it 'indicates' whether the event happened. For example, we could use a random variable that is 0 if one run of the algorithm at hand fails, and 1 if it succeeds. We will often define indicator random variables in the following shorter notation instead of referring to the actual elementary events:

$$X = \begin{cases} 1 & \text{if event } A \text{ occurs,} \\ 0 & \text{else.} \end{cases}$$

We observe that the expected value of an indicator variable X is equal to the probability that we get the value 1:

$$\mathbf{E}[X] = 0 \cdot \mathbf{Pr}(X = 0) + 1 \cdot \mathbf{Pr}(X = 1) = \mathbf{Pr}(X = 1). \tag{2.1}$$

A random variable X with values 0 and 1 and $p = \mathbf{Pr}(X = 1)$ is also called a *Bernoulli random variable with parameter* p.

Example 2.9. *We continue 2 from Example 2.5 where our random experiment consists of ten independent coin flips. We define the indicator variable X_i*

$$X_i = \begin{cases} 1 & \text{if the } i\text{th coin flip comes up heads,} \\ 0 & \text{else.} \end{cases}$$

for every i $\{1,\ldots,10\}$. We immediately observe that $\mathbf{E}[X_i] = \mathbf{Pr}(X_i = 1) = 1/2$. Before, we considered the random variable X which is the number of heads during the ten coin flips. This is just the sum of the indicator variables. By using linearity of expectation, the computation of the expected value of X now becomes a much easier task:

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{i=1}^{10} X_i\right] = \sum_{i=1}^{10} \mathbf{E}[X_i] = \sum_{1=1}^{10} \frac{1}{2} = 5.$$

For random variables that map to non-negative integers $\mathsf{N}_0 = \{0, 1, 2, 3, \ldots\}$, there is a useful generalization of (2.1):

Lemma 2.10. *Let $(\Omega, \mathbf{Pr})$ be a discrete probability space and let $X : \Omega \to \mathsf{N}_0$ be a discrete random variable. Assume that $\mathbf{E}[X]$ exists. Then it holds that*

$$\mathbf{E}[X] = \sum_{i=1}^{\infty} \mathbf{Pr}(X \geq i).$$

Proof. The statement is true because

$$\begin{aligned} \mathbf{E}[X] = \sum_{i=0}^{\infty} i \cdot \mathbf{Pr}(X = i) &= \sum_{i=1}^{\infty} i \cdot \mathbf{Pr}(X = i) \\ &= \sum_{i=1}^{\infty} \sum_{j=1}^{i} \mathbf{Pr}(X = i). \end{aligned}$$

Now we use that the existence of $\mathbf{E}[X]$ means that all sums in the equation converge and even converge absolutely. For absolutely convergent series, we can reorder the terms arbitrarily. The term $\mathbf{Pr}(X = 1)$ appears one time, the term $\mathbf{Pr}(X = 2)$ twice, the term $\mathbf{Pr}(X = 3)$ three times, and so on. We get the same terms in different order for

$$\sum_{j=1}^{\infty} \sum_{i=j}^{\infty} \mathbf{Pr}(X = i) = \sum_{i=1}^{\infty} \mathbf{Pr}(X \geq i).$$

□

2.1.2 Conditional Expected Values

We know from Example 2.5 that the expected value of the sum of two dice is 7. Assume that we observe the outcome of the first die roll. Then the expected value of the sum will change. We already know the concept of conditional probabilities and now define conditional expectations.

Definition 2.11. *Let* $(\Omega, \mathbf{Pr})$ *be a discrete probability space, let* $A \quad 2^{\Omega}$ *be an event and let* $X : \Omega \rightarrow \mathsf{R}$ *be a discrete random variable. The* conditional expected value of X under A exists *if* $\sum_{x \quad R(X)} |x| \cdot \mathbf{Pr}(X = x \mid A)$ *converges and is then defined as*

$$\mathbf{E}[X \mid A] = \sum_{x \quad R(X)} x \cdot \mathbf{Pr}(X = x \mid A).$$

In particular, if $Y : \Omega \rightarrow \mathsf{R}$ *is another discrete random variable and* $\mathbf{E}[X \mid Y = y]$ *for* $y \quad \mathsf{R}$ *exists, then we get:*

$$\mathbf{E}[X \mid Y = y] = \sum_{x \quad R(X)} x \cdot \mathbf{Pr}(X = x \mid Y = y).$$

Example 2.12. *We continue 1. from Example 2.5. Let* A *be the event that the first die roll is* 2*, let* X *be the random variable for the sum of the two dice. Then* $\mathbf{E}[X \mid A] = \sum_{x=3}^{8} x \cdot \frac{1}{6} = \frac{11}{2}$*. We can also say that* X_1 *is the random variable for the first roll,* X_2 *for the second roll, and then observe that*

$$\mathbf{E}[X \mid X_1 = 2] = \mathbf{E}[X_1 + X_2 \mid X_1 = 2] = \sum_{x=3}^{8} x \cdot \frac{1}{6} = \frac{11}{2}.$$

We can also use information about X *to obtain information about the first roll:*

$$\begin{aligned} \mathbf{E}[X_1 \mid X = 5] &= \sum_{x=1}^{4} x \cdot \mathbf{Pr}(X_1 = x \mid X = 5) \\ &= \sum_{x=1}^{4} x \cdot \frac{\mathbf{Pr}((X_1 = x) \cap (X = 5))}{\mathbf{Pr}(X = 5)} \\ &= \sum_{x=1}^{4} x \cdot \frac{\mathbf{Pr}((X_1 = x) \cap (X_2 = 5 - x))}{\mathbf{Pr}(X = 5)} \\ &= \sum_{x=1}^{4} x \cdot \frac{1/36}{4/36} = \frac{1}{4} \cdot 10 = \frac{5}{2} \end{aligned}$$

We do not want to miss the following rules when computing conditional expected values. They are proven similarly to Theorem 2.7.

Lemma 2.13. *Let* $(\Omega, \mathbf{Pr})$ *be a discrete probability space, let* $X, Y : \Omega \rightarrow \mathsf{R}$ *be discrete random variables and let* $A \quad 2^{\Omega}$ *be an event. If the expected values of* X *and* Y *exist, then it also holds that:*

1. *The conditional expected value of* cX *under* A *exists for all* $c \quad \mathsf{R}$ *and it is* $\mathbf{E}[cX \mid A] = c\mathbf{E}[X \mid A]$*.*
2. *The conditional expected value of* $X + Y$ *under* A *exists and* $\mathbf{E}[X + Y \mid A] = \mathbf{E}[X \mid A] + \mathbf{E}[Y \mid A]$*.*

Also observe that if X and Y are independent, then

$$\mathbf{E}[X \mid Y = y] = \sum_{x \quad R(X)} x \cdot \mathbf{Pr}(X = x \mid Y = y) = \sum_{x \quad R(X)} x \cdot \frac{\mathbf{Pr}((X = x) \cap (Y = y))}{\mathbf{Pr}(Y = y)}$$

$$= \sum_{x \ \ R(X)} x \cdot \frac{\mathbf{Pr}(X = x) \cdot \mathbf{Pr}(Y = y)}{\mathbf{Pr}(Y = y)} = \sum_{x \ \ R(X)} x \cdot \mathbf{Pr}(X = x) = \mathbf{E}[X] .$$

Example 2.14. *We continue 2. from Example 2.5. Let Y be the random variable that is equal to the number of heads we see in the first six of ten independent coin flips, let Z be the random variable that is equal to the number of heads we see in the last four of ten independent coin flips and let $X = Y + Z$ be the number of heads in all ten coin flips. Then $\mathbf{E}[X \mid Y = 4] = \mathbf{E}[Y + Z \mid Y = 4] = \mathbf{E}[Y \mid Y = 4] + \mathbf{E}[Z \mid Y = 4] = 4 + 2 = 6$.*

2.2 Binomial and Geometric Distribution

When we have n independent Bernoulli random variables $Y_1, \ldots, Y_n$ with the same parameter p and add them, then we get a binomially distributed random variable $X = Y_1 + Y_2 + \ldots + Y_n$ with parameters n and p. For example, X could describe the number of successful runs of an algorithm or the number of times that we see heads in a series of n coin flips. The random variable X in Example 2.9 is binomially distributed with parameters 10 and 1/2. Analogously to that example, we can use linearity of expectation to confirm our suspicion that the expected value of a binomially distributed variable with parameters n and p should be np.

Lemma 2.15. *Let $X = Y_1 + \ldots + Y_n$ be a binomially distributed random variable with parameters n and p. Then $\mathbf{E}[X] = np$.*

Proof. We know that Y_i is a Bernoulli random variable with parameter p for all i $\{1, \ldots, n\}$. We already observed that this implies that $\mathbf{E}[Y_i] = \mathbf{Pr}(Y_i = 1) = p$. By linearity of expectation, we get that

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{i=1}^{n} Y_i\right] = \sum_{i=1}^{n} \mathbf{E}[Y_i] = np.$$

□

The expected value of a binomially distributed random variable is easy to memorize. The probabilities for obtaining a specific number describe the distribution more precisely, but are a little more complex to compute.

Lemma 2.16. *Let $X = Y_1 + \ldots + Y_n$ be a binomially distributed random variable with parameters n and p. Then*

$$\mathbf{Pr}(X = j) = \binom{n}{j} \cdot p^j (1 - p)^{n-j}.$$

Proof. Observe that there are $m = \binom{n}{j}$ ways how j ones can occur in the sum of the n random variables. Each possibility has probability $p^j(1 - p)^{n-j}$. Formally, we can define an event A_i for each of the m possibilities and observe that these events are disjoint and their union is the event $X = j$. Each event has probability $p^j(1-p)^{n-j}$, so we have

$$\mathbf{Pr}(X = j) = \mathbf{Pr}\Big(\bigcup_{i=1}^{m} A_i\Big) = \sum_{i=1}^{m} \mathbf{Pr}(A_i) = m \cdot p^j(1-p)^{n-j} = \binom{n}{j} \cdot p^j(1-p)^{n-j}.$$

□

Another related type of random variables are *geometrically distributed* random variables. A geometrically distributed random variable models how long it takes until a Bernoulli experiment returns 1 for the first time. Let $Y_1, Y_2, Y_3, \ldots$ be independent Bernoulli random variables with the same parameter p. Then the corresponding geometrically distributed random variable $X : \Omega \to \mathsf{N}$ with parameter p has value i iff $Y_j = 0$ for $j < i$ and $Y_i = 1$. With such a random variable we can for example model how long we have to wait until independent repetitions of the same randomized algorithm lead to the first success.

The probability that $X = i$ is exactly $(1-p)^{i-1}p$ because the event Y_i happens if $i-1$ tries return 0 and the ith try returns 1. We know from Lemma 2.15 that the binomially distributed random variable with parameters n and p has expected value np, thus the number of tries that return 1 within the first $1/p$ tries is 1 (assuming that $1/p$ is an integer). This gives us the intuition that the corresponding geometrically distributed variable with parameter p should have expected value $1/p$. We see that this is indeed the case.

Lemma 2.17. *Let X be a geometrically distributed random variable with parameter $p > 0$. Then $\mathbf{E}[X] = 1/p$.*

Proof. We recall that Lemma 2.10 says that $\mathbf{E}[X] = \sum_{j=1}^{\infty} \mathbf{Pr}(X \geq j)$. What is the probability that at least j tries are necessary to get the first 1? This happens if and only if the first $j-1$ tries fail, so $\mathbf{Pr}(X \geq j) = (1-p)^{j-1}$. Now we can use that $\sum_{k=0}^{\infty} r^k = \frac{1}{1-r}$ for any r with $|r| < 1$ (this is a geometric *series*) to obtain that

$$\mathbf{E}[X] = \sum_{j=1}^{\infty} \mathbf{Pr}(X \geq j) = \sum_{j=1}^{\infty} (1-p)^{j-1} = \sum_{j=0}^{\infty} (1-p)^j = \frac{1}{1-1+p} = \frac{1}{p}.$$

□

Up to this point, we talked a lot about the fact that we expect to need $1/p$ tries until the first 1 occurs, that we expect one 1 among the first $1/p$ tries and that we can make the probability that no 0 occurs smaller and smaller by using even more independent tries. Of course, this does *not* mean that a 1 in the next try becomes more likely only because we have already seen a lot of 0s. On the contrary, an important property of the geometric distribution is that it is *memoryless*.

Lemma 2.18. *Let X be a geometrically distributed random variable with parameter p. It holds for all n N and k N_0 that*

$$\mathbf{Pr}(X = n + k \mid X > k) = \mathbf{Pr}(X = n).$$

Proof. By definition, we have that

$$\begin{aligned}\mathbf{Pr}(X = n + k \mid X > k) &= \frac{\mathbf{Pr}\Big((X = n + k) \cap (X > k)\Big)}{\mathbf{Pr}(X > k)} = \frac{\mathbf{Pr}(X = n + k)}{\mathbf{Pr}(X > k)} \\ &= \frac{(1 - p)^{n+k-1} \cdot p}{(1 - p)^k} = (1 - p)^{n-1} \cdot p = \mathbf{Pr}(X = n).\end{aligned}$$

For the third equality, we observe that the probability that we need more than k tries to get the first 1 is the probability that we get 0 in k consecutive tries. □

2.3 Applications

Our application section features the powerful linearity of expectation in different contexts, mixed with the knowledge we aquired about integer valued random variables. We start with *randomized QuickSort*, showing how randomization can protect an algorithm from worst case input instances. Then we do an excursion to randomized approximation algorithms, where we consider algorithms for the maximum cut problem and the vertex cover problem.

2.3.1 Randomized QuickSort

In this section, we analyze a randomized version of the popular sorting algorithm `QuickSort`. Recall that `QuickSort` is a recursive Divide&Conquer algorithm. As long as there are at least two elements, it chooses a pivot (element) x, partitions the elements except x in those smaller than x and larger than x and recursively sorts the two subsets. The results are then concatenated appropriately. The following pseudo code captures the essence of `QuickSort`. We assume that the input is a set of distinct numbers. Notice that we do not specify how the set is stored at the beginning or during the algorithm, we want to focus on the main algorithmic steps. We model the output as an ordered vector. The desired output is the (unique) vector where the input elements are sorted in increasing order.

```
QuickSort($S = \{x_1, \ldots, x_n\}$)
    1. if ($n = 0$) then return ()
    2. if ($n = 1$) then return ($x_1$)
    3. Choose a pivot element $x$   $S$
    4. Compute $S_1 = \{x_i \mid x_i < x\}$ and $S_2 = \{x_i \mid x_i > x\}$
    5. $A_1$ = QuickSort($S_1$); $A_2$ = QuickSort($S_2$);
    6. Concatenate $A_1$, ($x$) and $A_2$ to obtain the vector $A$
    7. return $A$
```

The running time of `QuickSort` depends crucially on the specification of Step 3, the choice of the pivot element. We recall that the worst case running time of `QuickSort` is $\Theta(n^2)$ if the algorithm chooses x_1 as the pivot element in every step. The bad case that can then occur (in every call with $n \geq 2$) is that one of the sets S_1 and S_2 has $n - 1$ elements and the other is empty because x_1 always happened to be the smallest/largest element in S. This happens if the array is sorted in the beginning.

Ideally, the pivot element is the median of the elements in S because that splits S into two sets of equal size (or nearly equal size, depending on whether n is odd or even). Computing the median in time $O(n)$ is possible (an algorithm to do so was published in [BFP$^+$73]), and there is also an easier randomized algorithm for this task. However, for the purpose of a lean QuickSort implementation it is even easier to randomize the choice of the pivot element in a straightforward manner. We analyze the algorithm `RQuickSort` that always chooses the pivot element uniformly at random from S in Step 3.

Theorem 2.19. *The expected number of comparisons of `RQuickSort` is $2n \ln n - \Theta(n)$ for any input S with n distinct elements.*

Proof. Let $(y_1, \ldots, y_n)$ be the sorted vector containing the elements $x_1, \ldots, x_n$, i.e., the output of `RQuickSort`. For any pair $i, j \quad \{1, \ldots, n\}$ with $i < j$ we define the (indicator) random variable X_{ij} by

$$X_{ij} = \begin{cases} 1 & \text{if } y_i \text{ and } y_j \text{ are compared during the execution of } \texttt{RQuicksort}, \\ 0 & \text{else.} \end{cases}$$

Observe that two numbers y_i and y_j are never compared twice during the execution of `RQuicksort`: Comparisons are only made between a pivot element and other numbers. Thus, when y_i and y_j are compared for the first time, one of them is the pivot element. Thus, it is no longer present in the recursive calls and no further comparisons between y_i and y_j can occur. Thus, if we model the total number of comparisons by the random variable X, we have

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

where we use linearity of expectation. To compute $\mathbf{Pr}(X_{ij} = 1)$, we observe that y_i and y_j are compared if and only if either y_i or y_j is the first element from the set $\{y_i, \ldots, y_j\}$ that is chosen as a pivot element. To see that this is true, let x be the first element from $\{y_i, \ldots, y_j\}$ that is chosen as the pivot element.

- If $x = y_i$ and $x = y_j$, then y_i and y_j are compared with x and not with each other. Since $x \quad \{y_i, \ldots, y_j\}$, we have $y_i < x < y_j$. Thus, the two elements are then separated because $y_i \quad S_1$ and $y_j \quad S_2$. Thus, they are never compared.
- If $x = y_i$ or $x = y_j$, then y_i and y_j are compared. In the recursive calls, x is no longer present and no further comparisons occur.

As long as no element from $\{y_i, \ldots, y_j\}$ is chosen as a pivot element, a call of `RQuickSort` either has all or no elements from $\{y_i, \ldots, y_j\}$ in S. Fix an arbitrary call of `RQuickSort` where $\{y_i, \ldots, y_j\}$ is still part of the current set S. Let P be the random variable whose value is the pivot element that is chosen in this call of `RQuickSort`. P is chosen uniformly at random from S, so $\mathbf{Pr}(P = x) = \frac{1}{|S|}$ for all $x \quad S$. We observe that

$$\begin{aligned}
&\mathbf{Pr}((P = y_i) \quad (P = y_j) \mid P \quad \{y_i, \ldots, y_j\})\\
=&\frac{\mathbf{Pr}(((P = y_i) \quad (P = y_j)) \cap (P = x \text{ with } x \quad \{y_i, \ldots, y_j\}))}{\mathbf{Pr}(P = x \text{ with } x \quad \{y_i, \ldots, y_j\})}\\
=&\frac{\mathbf{Pr}((P = y_i) \quad (P = y_j))}{\mathbf{Pr}(P = x \text{ with } x \quad \{y_i, \ldots, y_j\})}\\
=&\frac{2/|S|}{(j - i + 1)/|S|} = \frac{2}{j - i + 1}.
\end{aligned}$$

This is intuitive: The first element from $\{y_i, \ldots, y_j\}$ that is chosen as the pivot is chosen uniformly at random from a super set of $\{y_i, \ldots, y_j\}$, so *if* it is from $\{y_i, \ldots, y_j\}$, then it is a uniformly chosen element from $\{y_i, \ldots, y_j\}$. Since y_i and y_j are two elements, choosing one of them has a probability of $2/(j - i + 1)$. We observe that

$$\mathbf{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbf{Pr}(X_{ij} = 1) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1} = \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k}.$$

For a specific $j \quad \{2, \ldots, n\}$, the term $1/j$ occurs in the second sum if and only if $j \leq n - i + 1$. This is true for all $i \quad \{1, \ldots, n - j + 1\}$, thus the term $1/j$ occurs exactly $n - j + 1$ times. We thus get that

$$\begin{aligned}
\sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k} = \sum_{j=2}^{n} \frac{2}{j}(n - j + 1) &= (n + 1)\left(\sum_{j=2}^{n} \frac{2}{j}\right) - 2(n - 1)\\
&= (2n + 2)(H_n - 1) - 2(n - 1),
\end{aligned}$$

where $H_n = \sum_{i=1}^{n}(1/i)$ is the n*-th harmonic number*. Since $H_n \leq \ln n + 1$ for $n \geq 1$, we get that

$$\mathbf{E}[X] = (2n + 2)(H_n - 1) - 2(n - 1) \leq 2n \ln n + 2 \ln n - 2n + 2 = 2n \ln n - \Theta(n).$$

□

We could use the same proof to show the following fundamentally different statement.

Fact 2.20. *Let `SQuickSort` be the version of `QuickSort` where the pivot element is always chosen as x_1. If the permutation of the elements in the input is assumed to be chosen uniformly at random from the set of all possible permutations, then the expected number of comparisons that `SQuickSort` performs is $2n \ln n - \Theta(n)$.*

The randomness is shifted from the algorithm to the input: If we assume that the distribution with which different input permutations occur is uniform, then we can show the same guarantee for a deterministic `QuickSort` variant. This is the type of statement that we will see in the second part of the lecture. However, assuming that the input is a uniformly chosen permutation is a very strong assumption. It is preferable to randomize the algorithm to guarantee the expected performance. Probabilistic analysis is about showing running time bounds for algorithms under much weaker assumptions on the input data.

2.3.2 Randomized Approximation Algorithms

Recall that an α-approximation algorithm for a minimization problem is an algorithm that outputs solutions with a value that is at most α times the value of an optimal solution. Let $\mathfrak{I}$ be the set of all possible input instances to an optimization problem, let ALG be an algorithm that computes a feasible solution $S^{ALG}(I)$ for any $I \in \mathfrak{I}$. For any $I \in \mathfrak{I}$, let $S^*(I)$ be an optimal solution for I. For a minimization problem, suppose that $c(S)$ is the cost of a solution S. Then ALG is an α-approximation algorithm iff

$$\sup_{I \in \mathfrak{I}} \frac{c(S^{ALG}(I))}{c(S^*(I))} \leq \alpha.$$

For a maximization problem, let $v(S)$ be the value of a solution S. In this case, ALG is an α-approximation algorithm iff

$$\sup_{I \in \mathfrak{I}} \frac{v(S^*(I))}{v(S^{ALG}(I))} \leq \alpha.$$

In both cases, we know that $\alpha \geq 1$ since ALG cannot compute solutions that are better than an optimal solution. Notice that we usually assume that a deterministic approximation algorithm has polynomial worst-case running time.

For randomized algorithms, $c(S^{ALG}(I))$ or $v(S^{ALG}(I))$, respectively, is a random variable. We might either want to achieve that it is close to the optimum value with high probability, or that its expected value is close to the optimum value. We choose the second alternative and will later discuss how to obtain statements of the first type. Thus, we say that a randomized algorithm ALG is an α-approximation algorithm in expectation for a minimization problem iff

$$\sup_{I \in \mathfrak{I}} \frac{\mathbf{E}\left[c(S^{ALG}(I))\right]}{c(S^*(I))} \leq \alpha$$

and it is an α-approximation algorithm in expectation in the case of a maximization problem iff

$$\sup_{I \in \mathfrak{I}} \frac{v(S^*(I))}{\mathbf{E}\left[v(S^{ALG}(I))\right]} \leq \alpha.$$

The randomized approximation algorithms in this section have a polynomial worst-case running time.

The MaxCut problem. We consider the *weighted maximum cut* problem, abbreviated as the *MaxCut problem*. We are given an undirected graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$, $E \subseteq V \times V$, and a weight function $w : E \to \mathbb{R}^+$ that maps each edge to a positive weight. We are supposed to output a subset $S \subseteq V$ that *maximizes* the total weight of the edges that are cut. In other words, we are supposed to maximize

$$v(S) := \sum_{e \in \delta(S)} w(e).$$

For the min cut problem, we explicitly excluded the cases $S = V$ and $S = \emptyset$. These solutions both cut no edge, so the sum then is zero. Since we are now interested in maximization, we do not need to exclude these solutions which are the worst possible solutions anyway.

The MaxCut problem is NP-hard, it was among the 21 problems that Karp showed to be NP-hard in 1972 [Kar72]. We discuss a randomized approximation algorithm for the MaxCut problem with expected approximation guarantee 2. The algorithm can be derandomized, i.e., one can give a deterministic 2-approximation algorithm. The deterministic variant was published by Sahni and Gonzales [SG76] in 1976. In the subsequent years, approximation algorithms were found with a guarantee of $2 - o(1)$, but no algorithm that improved the guarantee to a *constant* below 2. Thus, for a long time, the approximation factor achieved by the simple randomized algorithm was essentially the best approximation guarantee.

The constant was finally improved in 1994, in a seminal paper by Goemans and Williamson [GW94, GW95], who give a randomized 1.38-approximation by using a technique called *semidefinite programming* which goes beyond the scope of this lecture.

The randomized algorithm we consider is very simple: Start with $S = \emptyset$. For each vertex v, add v to S with probability $1/2$. The running time of this algorithm is $\Theta(|V|)$. For any edge $e = \{v_i, v_j\} \in E$, $i < j$, we define $X_e = X_{ij}$ to be the indicator random variable for the event that e is in $\delta(S)$. This happens if and only if either $v_i \in S, v_j \notin S$ or $v_i \notin S, v_j \in S$. The decisions for v_i and v_j are independent, so $\mathbf{Pr}(v_i \in S \cap v_j \notin S) = (1/2) \cdot (1/2) = 1/4$, and also $\mathbf{Pr}(v_i \notin S \cap v_j \in S) = 1/4$. Both events are disjoint, so $\mathbf{Pr}(e \in \delta(S)) = (1/4) + (1/4) = 1/2$. By linearity of expectation, we obtain

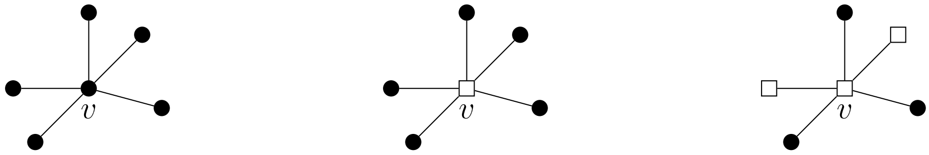
$$\mathbf{E}\left[\sum_{e \in \delta(S)} w_e\right] = \mathbf{E}\left[\sum_{e \in E} X_e w_e\right] = \sum_{e \in E} w_e \mathbf{E}[X_e] = \sum_{e \in E} w_e \cdot (1/2).$$

We can never cut more than *all* edges, so $\sum_{e \in E} w_e$ is an upper bound on the value of the best possible solution. Thus, the algorithm is a 2-approximation algorithm in expectation.

The VertexCover problem. Now we consider the *(unweighted) vertex cover problem (VertexCover)*. We are given an undirected and unweighted graph $G = (V, E)$. We want to output a set $C \subseteq V$ such that every edge in the graph is *covered* by C: For every $e \in E$ with $e = \{u, v\}$, we require that $u \in C$ or $v \in C$ (or both). The goal is to minimize the number of nodes $|C|$. The vertex cover problem is a minimization problem.

VertexCover is NP-hard, it also belongs to the problems proven to be NP-hard in [Kar72]. Different 2-approximation algorithms for the problem are known, including deterministic algorithms. It is unknown whether an α-approximation algorithm is possible for a constant $\alpha < 2$ (there are hints that this might be impossible).

From our experience with the MaxCut problem, we might try the algorithm that adds every vertex to C with probability 1/2. We observe two problems with this approach: First, the output might not actually be a vertex cover, some edges might not get covered. Second, the algorithm could pick a lot more vertices than an optimal solution. Consider a star graph S, i.e., a graph with vertex set $\{v\} \quad V$ for some set V and edge set $\{\{v, v\ \} \mid v \quad V\ \}$. If we decide for each vertex whether it is in C or not uniformly at random, then the expected number of vertices in C is $|V\ |/2$. However, an optimal vertex cover for this graph is $C\ = \{v\}$ with $|C\ | = 1$. Thus, the expected approximation ratio of this algorithm (that also does not produce feasible solutions) would be $\Omega(|V|)$.



a star graph S optimal solution C a solution with $|C| = \frac{|V\ |+1}{2}$

The following simple algorithm works. Start with $C = \quad$. For all *edges* $e \quad E$, $e = \{u, v\}$, check if e is already covered by C. If not, add either u or v to S: With probability 1/2, add u, with the remaining probability, add v. After processing all edges, it is ensured that all edges are covered. The running time of the algorithm is $\Theta(|V| + |E|)$. The algorithm is due to Pitt, who developed a version for the more general case of *weighted* vertex cover and proved that it is a 2-approximation [Pit85].

In the following, for any graph G, let $C^{ALG}(G)$ be the solution computed by the simple vertex cover approximation algorithm, and let $C\ (G)$ be an optimal vertex cover. We reconsider the above example, i.e., we let S be a star graph like defined above. We already observed that $C\ (S) = \{v\}$. We are interested in the random variable $Z := |C^{ALG}(S)|$. The algorithm starts with $C^{ALG}(S) = \quad$, so Z starts at zero. The edges of S are considered in an arbitrary order. For edge $e = \{v, v\ \}$, two events can happen: With probability 1/2, v is added to $C^{ALG}(S)$, thus Z increases by one, and only the edge $\{v, v\ \}$ is covered by the new vertex. With probability 1/2, v is added. In this case, Z also increases by one, but all edges are now covered and Z freezes. We observe that Z is a geometrically distributed random variable with parameter 1/2! Thus, we immediately conclude that $E[Z] = 2$. To be completely precise, observe that Z is actually upper bounded by a geometrically distributed random variable with parameter 1/2: While the geometrically distributed random variable can attain any positive integer, Z is bounded by $|V\ |$. So the precise statement is that $\mathbf{E}[Z] \leq 2$.

Since the optimum solution for S consists of one vertex, we have now proven that the expected approximation guarantee is bounded by 2 if the input is a star graph. The following theorem extends the analysis to general input graphs.

Theorem 2.21. *The randomized vertex cover algorithm is a 2-approximation algorithm in expectation.*

Proof. Let $G = (V, E)$ be an arbitrary input graph and assume without loss of generality that V does not contain isolated vertices. We use the abbreviations $C \ = C \ (G)$ and $C^{ALG} = C^{ALG}(G)$. Furthermore, we set $k = |C \ |$ and name the vertices in the optimum vertex cover by assuming that $C \ = \{v_1, \ldots, v_k\}$.

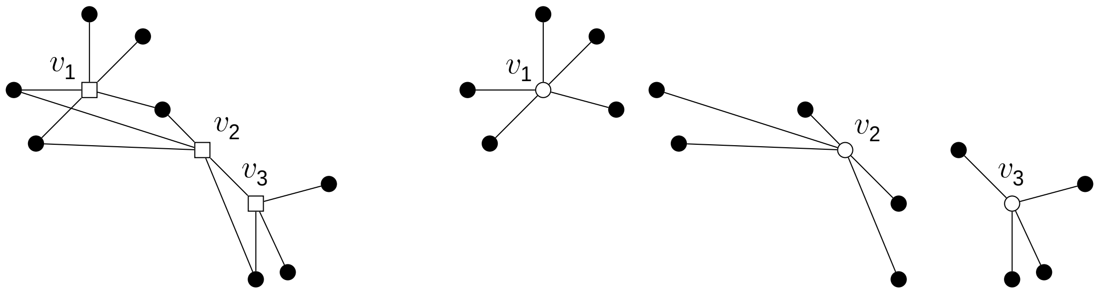
Step 1: In order to use our observations about star graphs, we want to partition G into stars. We observe the following: The set of edges covered by a vertex v_i is a star. These stars are not disjoint, so they do not form a true partitioning of G. However, they cover G completely. We formalize this idea. For any $v_i \quad C$, let $G_i = (V_i \ , E_i \ )$ be defined as

$$\begin{aligned} V_i \ &= \{v_i\} \quad \{u \quad V \mid \{u, v_i\} \quad E\}, \\ E_i \ &= \{\{u, v_i\} \mid \{u, v_i\} \quad E\}. \end{aligned}$$

As already indicated, the intersection of V_i and V_j for $i = j$ is not necessarily empty, the same holds for E_i and E_j for $i = j$. However, we have that

$$V = \bigcup_{i=1}^{k} V_i \ \text{ and } E = \bigcup_{i=1}^{k} E_i$$

because C is a feasible vertex cover. The following is an example in which the subgraphs overlap.



G and an optimal C subgraphs G_1, G_2, G_3

Step 2: Observe that the number of vertices added to C^{ALG} is exactly the number of edges that are still uncovered when considered by the algorithm. We define the random variable Z_i as the number of edges from E_i that are uncovered when considered by the algorithm. Then $|C^{ALG}| \leq \sum_{i=1}^{k} Z_i$. Now we look at the edges in E_i . For each uncovered edge, we have a probability of 1/2 to add v_i to C^{ALG}, and then all edges in E_i are covered. Thus, Z_i is upper bounded by a geometrically distributed random variable with parameter 1/2, and this implies that $E(Z_i) \leq 2$.

Step 3: By linearity of expectation, we obtain that

$$\mathbf{E}\left[|C^{ALG}|\right] = \sum_{i=1}^{k} \mathbf{E}\left[Z_i\right] \leq 2k.$$

Since the optimal vertex cover has k nodes, we have now proven that the algorithm computes a 2-approximation in expectation. $\square$

Chapter 3

Concentration Bounds

This chapter covers selected material from Chapter 3 and 4 in [MU05]. Section 3.3.2 is also covered in 4.5.1 in [MU05], but is closer to chapter 4 in [Wor15] (in German).

We see bounds for the probability that a random variable deviates from its expected value, depending on the extent of the deviation. Bounding this probability is not always possible, in fact, the expected value of a random variable might not even exist and then the question becomes irrelevant. However, if we know that the random variable has good properties, we can apply one of various concentration bounds, also called ***tail bounds***. We see three types of tail bounds in this chapter. The first one only needs that the random variable is non-negative and that its expected value exists. These are rather weak preconditions, and the bound is also the weakest that we see in this chapter. Nevertheless, it is tremendously useful.

Theorem 3.1 (Markov's inequality). *Let* $(\Omega, \mathbf{Pr})$ *be a discrete probability space, let* $X : \Omega \to \mathsf{R}_{\geq 0}$ *be a random variable that attains non-negative values. Assume that* $\mathbf{E}[X]$ *exists. Then*

$$\mathbf{Pr}(X \geq a) \leq \frac{\mathbf{E}[X]}{a} \tag{3.1}$$

holds for all $a > 0$*. Thus, it also holds*

$$\mathbf{Pr}(X \geq b \cdot \mathbf{E}[X]) \leq \frac{1}{b} \tag{3.2}$$

for all $b > 1$*.*

Proof. Let Y be the indicator random variable with

$$Y = \begin{cases} 1 & \text{if } X \geq a \\ 0 & \text{else.} \end{cases}$$

Then, $\mathbf{E}[Y] = \mathbf{Pr}(Y = 1) = \mathbf{Pr}(X \geq a)$ is the quantity that we want to bound. We observe that $Y \leq \frac{X}{a}$ is always true. We know that $X/a \geq 0$ since $X \geq 0$ and $a > 0$.

Thus, if $X < a$, implying that $Y = 0$, then $X/a \geq 0 = Y$. Furthermore, if $X \geq a$ and thus $Y = 1$, then $X/a \geq 1 = Y$, so $X/a \geq Y$ holds as well. This extends to the expected value, i.e., $\mathbf{E}[Y] \leq \mathbf{E}[X/a]$ (recall Observation 2.6). Now we can conclude by linearity that

$$\mathbf{Pr}(X \geq a) = \mathbf{E}[Y] \leq \mathbf{E}\left[\frac{X}{a}\right] = \frac{\mathbf{E}[X]}{a}.$$

Inequality 3.2 follows by setting $a = b \cdot \mathbf{E}[X]$. Notice that we could do this for values $b \quad (0,1]$ as well, but then the bound is not meaningful, so we restrict b to values greater than one. □

Markov's inequality becomes particularly easy to memorize if one looks at the interpretation for finite random variables. For example, assume that we have n non-negative numbers $a_1, \ldots, a_n$, and we consider the random variable X that chooses one of these numbers uniformly at random. Then the expected value of X is just the arithmetic mean $\hat{a}$ of the numbers:

$$\mathbf{E}[X] = \sum_{i=1}^{n} \frac{1}{n} a_i = \frac{1}{n} \sum_{i=1}^{n} a_i = \hat{a}.$$

It is easy to prove that not too many numbers can deviate very much from $\hat{a}$. It is also intuitive: In a group of people with average income $\hat{a}$, not more than half of the people can earn $2\hat{a}$ or more. Otherwise, the average would be higher. Similarly, assume that more than n/k of the a_i satisfy that $a_i \geq k\hat{a}$. Since the a_i are non-negative, the sum of all numbers is not smaller than the sum of these more than n/k numbers. Thus:

$$\sum_{i=1}^{n} a_i > \frac{n}{k} k\hat{a} = n \frac{1}{n} \sum_{i=1}^{n} a_i = \sum_{i=1}^{n} a_i.$$

Clearly, $\sum_{i=1}^{n} a_i > \sum_{i=1}^{n} a_i$ is impossible, so our assumption was wrong. There can at most be n/k numbers a_i that satisfy $a_i \geq k\hat{a}$. This is exactly the statement of Markov's inequality for X:

$$\begin{aligned} \frac{|\{a_i \mid a_i \geq k \cdot \hat{a}\}|}{n} &= \mathbf{Pr}(X \geq k \cdot \mathbf{E}[X]) \leq \frac{1}{k} \\ \Leftrightarrow \quad |\{a_i \mid a_i \geq k \cdot \hat{a}\}| &\leq \frac{n}{k}. \end{aligned}$$

Observe that we used that the numbers are non-negative to argue that $\sum_{i=1}^{n} a_i > \frac{n}{k} k\hat{a}$. Indeed, if the numbers can be negative (like in the list of numbers $2, 2, 2, 2, -8$), then a large fraction of the numbers can be larger than the expected value ($4/5$ in the example, or $(n-1)/n$ if we have $n-1$ twos and $-2(n-1)$ as the nth number). They can even be a lot larger than the expected value (just make the nth number $-k$ for a large $k \quad \mathsf{N}$).

Implications of Markov's inequality. We directly obtain the following consequences from Markov's inequality. Think of X as the running time of an algorithm or the approximation ratio of a randomized approximation algorithm. Assume that we have shown a bound for $\mathbf{E}[X]$, so, in particular, $\mathbf{E}[X]$ exists. In both cases, $X \geq 0$. By Markov's inequality,

$$\mathbf{Pr}(X \geq (1+\varepsilon)\mathbf{E}[X]) \leq \frac{1}{1+\varepsilon}$$

holds for any constant $\varepsilon > 0$. We can use independent repetitions to decrease the probability even further. If X is the running time of an algorithm, stop the algorithm after $(1+\varepsilon)\mathbf{E}[X]$ steps. If the algorithm was not finished, try again, doing at most t repetitions. If X is the approximation ratio, simply repeat the algorithm t times and output the best solution. Let X_i be the running time / approximation ratio in the ith run, and set $t = \log_{1+\varepsilon} n \quad O(\log_2 n)$. Then

$$\mathbf{Pr}(\min_{i=1,\ldots,t} X_i \geq (1+\varepsilon)\mathbf{E}[X]) \leq \frac{1}{(1+\varepsilon)^t} = \frac{1}{n},$$

so the probability that we have at least one run with $X \leq (1+\varepsilon)\mathbf{E}[X]$ in $\Theta(\log n)$ tries is at least $1 - \frac{1}{n}$, and obtaining this statement only needed Markov's Inequality.

3.1 Variance and Chebyshev's inequality

Before we prove Chebyshev's inequality, we need to define the *variance* of a random variable X. It is the expected value of the squared deviation of X from its expected value and thus a very useful quantity for obtaining a tail bound.

Definition 3.2. *Let* $(\Omega, \mathbf{Pr})$ *be a discrete probability space, let* $X : \Omega \to \mathrm{R}$ *be a discrete random variable. Assume that* $\mathbf{E}[(X - \mathbf{E}[X])^2]$ *exists. Then the* variance $\mathbf{Var}[X]$ *of* X *exists and is defined as*

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

Observe that $\mathbf{E}[X]$ is a constant. If it appears inside another expected value, we can use linearity. We use this fact to prove that the two definitions of the variance are indeed equivalent.

$$\begin{aligned}
\mathbf{E}\left[(X - \mathbf{E}[X])^2\right] &= \mathbf{E}\left[X^2 - 2X\mathbf{E}[X] + (\mathbf{E}[X])^2\right] \\
&= \mathbf{E}\left[X^2\right] - 2\mathbf{E}[X \cdot \mathbf{E}[X]] + \mathbf{E}\left[(\mathbf{E}[X])^2\right] \\
&= \mathbf{E}\left[X^2\right] - 2\mathbf{E}[X] \cdot \mathbf{E}[X] + (\mathbf{E}[X])^2 \\
&= \mathbf{E}\left[X^2\right] - 2(\mathbf{E}[X])^2 + (\mathbf{E}[X])^2 \\
&= \mathbf{E}\left[X^2\right] - (\mathbf{E}[X])^2.
\end{aligned}$$

Since the variance is inherently a squared measure, it is common to also define the *standard deviation* $\sigma(X)$ as $\sqrt{\mathbf{Var}[X]}$.

Example 3.3. *If X always attains the same value, e.g., $X = 7$, then $\mathbf{E}[X] = X$ and thus $\mathbf{E}[(X - \mathbf{E}[X])^2] = 0$, the random variable has zero variance. We have also seen an example for the other extreme case in Example 2.8 where we chose each number i N with probability c/i^4 for a constant $c > 0$ and let X be the chosen number. We saw that $\mathbf{E}[X]$ exists, but $\mathbf{E}[X^2]$ does not. Thus, $\mathbf{Var}[X]$ also does not exist.*

The rules for computing expected values in Theorem 2.7 can be used to prove the following rules for computing the variance of random variables.

Lemma 3.4. *Let $(\Omega, \mathbf{Pr})$ be a discrete probability space and let $X, Y : \Omega \to$ R be discrete random variables. Assume that $\mathbf{Var}[X]$ and $\mathbf{Var}[Y]$ exist, let a, b R be two real values. Then the following statements hold.*

1. *$\mathbf{Var}[aX + b]$ exists and $\mathbf{Var}[aX + b] = a^2 \cdot \mathbf{Var}[X]$.*
2. *If X and Y are independent, then $\mathbf{Var}[X + Y]$ exists and $\mathbf{Var}[X + Y] = \mathbf{Var}[X] + \mathbf{Var}[Y]$.*

Proof. Exercise. □

The second rule can be extended to sums of n random variables by induction.

Corollary 3.5. *Let $(\Omega, \mathbf{Pr})$ be a discrete probability space and let $X_1, X_2, \ldots, X_n$ be independent random variables, assume that $\mathbf{Var}[X_i]$ exists for all i $\{1, \ldots, n\}$. Let $X := \sum_{i=1}^{n} X_i$. Then $\mathbf{Var}[X]$ exists and it holds that $\mathbf{Var}[X] = \sum_{i=1}^{n} \mathbf{Var}[X_i]$.*

We can use Corollary 3.5 to compute the variance of binomially distributed variables.

Example 3.6. *Let X be a Bernoulli variable with parameter p. We know that $\mathbf{E}[X^2]$ exists since Ω is finite in this case. We observe that $X^2 = X$ is always true, and thus, $\mathbf{E}[X^2] = p$. We use this and compute the variance of X:*

$$\mathbf{Var}[X] = \mathbf{E}\left[X^2\right] - (\mathbf{E}[X])^2 = p - p^2 = p(1 - p).$$

By Corollary 3.5, we conclude that the variance of a binomially distributed variable $Y = Y_1 + \ldots + Y_n$ also exists and it is $\mathbf{Var}[Y] = np(1 - p)$.

Now we can state and prove Chebyshev's inequality. The variance is the expected value of $(X - \mathbf{E}[X])^2$, so it is an expected value itself. Chebychev's inequality arises from applying Markov's inequality to this expected value.

Theorem 3.7 (Chebyshev's inequality). *Let $(\Omega, \mathbf{Pr})$ be a discrete probability space, let $X : \Omega \to$ R be a discrete random variable. Assume that $\mathbf{E}[X]$ and $\mathbf{Var}[X]$ exist. Let $c > 0$ be a constant. Then it holds that*

$$\mathbf{Pr}(|X - \mathbf{E}[X]| \geq c) \leq \frac{\mathbf{Var}[X]}{c^2}.$$

This also implies that

$$\mathbf{Pr}(|X - \mathbf{E}[X]| \geq \delta \cdot |\mathbf{E}[X]|) \leq \frac{\mathbf{Var}[X]}{\delta^2 \cdot (\mathbf{E}[X])^2}$$

holds for all $\delta > 0$.

Proof. First, we observe that

$$\mathbf{Pr}(|X - \mathbf{E}[X]| \geq c) = \mathbf{Pr}((X - \mathbf{E}[X])^2 \geq c^2).$$

Now, we apply Markov's inequality to the random variable $(X - \mathbf{E}[X])^2$. Notice that the expected value of this variable is $\mathbf{Var}[X]$, so it exists by the precondition of the theorem. Furthermore, observe that $(X - \mathbf{E}[X])^2$ only attains non-negative values. Thus, we can apply Markov's inequality and obtain

$$\mathbf{Pr}((X - \mathbf{E}[X])^2 \geq c^2) \leq \frac{\mathbf{E}[(X - \mathbf{E}[X])^2]}{c^2} = \frac{\mathbf{Var}[X]}{c^2},$$

the inequality we wanted to prove. For the second inequality, set $c = \delta \cdot |\mathbf{E}[X]|$, which is always positive since $\delta > 0$. Furthermore, notice that $(|\mathbf{E}[X]|)^2 = (\mathbf{E}[X]|)^2$. □

3.2 Chernoff/Rubin bounds

Chernoff bounds are a powerful type of concentration bounds. They are named after Herman Chernoff. However, Chernoff attributes them to Herman Rubin, for example in an article by himself [Che04] and in an interview [Bat96], thus they should rather be called *Rubin bounds*. Chernoff/Rubin bounds are obtained by applying Markov's inequality to the moment generating function of X. We skip the definition of moment generating functions and the derivation of Chernoff/Rubin bounds and just cite two versions. Both are proven in Section 4.2 of [MU05], see Theorem 4.4 (2.+3.)/4.5 (2.). Notice that the strongest bounds in [MU05] are Theorem 4.4 (1.) and Theorem 4.5 (1.), but we are satisfied with the following simplified variants.

Theorem 3.8. *Let $(\Omega, \mathbf{Pr})$ be a discrete probability space and let $X_1, \ldots, X_n$ be independent Bernoulli random variables with parameters $p_1, \ldots, p_n$. Let $X = X_1 + \ldots + X_n$. Observe that $\mathbf{E}[X] = p_1 + \ldots + p_n$.*

1. *For every $0 < \delta \leq 1$, it holds that*

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

2. *For every $0 < \delta < 1$, it holds that*

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

The second version will be helpful for our second application later on. It allows us to use upper bounds instead of the expected value, and it has an easy to memorize form.

Theorem 3.9. *Let $(\Omega, \mathbf{Pr})$ be a discrete probability space and let $X_1, \ldots, X_n$ be independent Bernoulli random variables with parameters $p_1, \ldots, p_n$. Let $X = X_1 + \ldots + X_n$. Then it holds for every $a \geq 6 \cdot \mathbf{E}[X] = 6(p_1 + \ldots + p_n)$ that*

$$\mathbf{Pr}(X \geq a) \leq 2^{-a}.$$

The following example compares the three types of concentration bounds that we have now seen.

Example 3.10. *We consider n independent tosses of a fair coin. We use the random variable X to model the number of heads that we see. This variable is binomially distributed with parameters n and $1/2$, and its expected value is $n/2$. With Markov's inequality, we observe that*

$$\mathbf{Pr}\left(X \geq \frac{3}{4}n\right) \leq \frac{\mathbf{E}[X]}{\frac{3}{4}n} = \frac{n/2}{\frac{3}{4}n} = \frac{2}{3}.$$

To apply Chebyshev's inequality, we observe that $\mathbf{Var}[X]$ exists and that it holds that $\mathbf{Var}[X] = n \cdot (1/2) \cdot (1-(1/2)) = (n/4)$. Chebyshev's inequality thus gives the stronger bound

$$\mathbf{Pr}\left(X \geq \frac{3}{4}n\right) \leq \mathbf{Pr}\left(\left|X - \frac{1}{2}n\right| \geq \frac{1}{4}n\right) \leq \frac{\mathbf{Var}[X]}{(\frac{1}{4}n)^2} = \frac{n/4}{n^2/16} = \frac{4}{n}.$$

This bound is stronger. It tells us that the probability for $(3/4)n$ heads in n coin tosses decreases polynomially in n. We observe that Theorem 3.8 can also be applied to binomially distributed random variables since it is a sum of Bernoulli variables. We apply the first inequality with $\delta = 1/2$ since $(1+(1/2)) \cdot (n/2) = (3/4)n$. The result is that

$$\mathbf{Pr}\left(X \geq \frac{3}{4}n\right) \leq e^{-(\mathbf{E}[X]\delta^2)/3} = e^{-\frac{1}{2}\cdot n \cdot \frac{1}{4} \cdot \frac{1}{3}} = e^{-\frac{n}{24}}.$$

That is the strongest of the three bounds, it says that the probability to see $(3/4)n$ heads in n coin flips decreases exponentially in n.

3.3 Applications

We discuss three applications. For the first application, we apply Chebyshev's inequality. The application is about estimating the number of connected components in a graph in sublinear time. Then we see applications for Chernoff/Rubin Bounds: One in statistics, where we want to estimate the parameter of a distribution, and one from the area of parallel computing, where we discuss oblivious routing strategies (in hypercubes).

3.3.1 A sublinear algorithm

In Section 1.3.2, we developed a sampling algorithm for the data stream model. In the data stream model, we can read the data only once and have to solve our problem based on information that we stored in a memory of sublinear size. In this section, we consider a similar yet different model: We want to solve a problem (approximately) in sublinear *time*. This means that we cannot even read the complete data. We have to choose which parts of the data we want to see. However, we do have random access to the data.

Example 3.11. *Consider the following toy problem. We are given a picture with n black/white pixels and want to decide whether the picture is completely white or has a significant number of black pixels. By the latter we mean that there are at least $(1/10)n$ black pixels. Thus, we want an algorithm that can distinguish the following two cases:*

1. *all pixels are white*
2. *at least $(1/10)n$ pixels are black*

If none of the two cases applies, then we allow our algorithm to answer arbitrarily, i.e., it may reply either case 1 or case 2.

Observe that a deterministic algorithm that distinguishes the two cases needs linear time: Assume that the algorithm is presented with a picture, and the first $\leq (9/10)n$ pixels that it reads are all white. Then it can still not be sure whether the number of black pixels is at least $(1/10)n$ or not. To decide this, it has to read at least $(9/10)n+1$ pixels.

A randomized way of solving this problem is the following: Choose $t = \log_{(10/9)} n$ pixels uniformly at random (with replacement). If at least one pixel is black, output case 2, otherwise, output case 1. If the picture is completely white, then the algorithm has no error since it will only see white pixels. If there are at least $(1/10)n$ black pixels, then the probability to draw a black pixel is at least $1/10$. Thus, the probability that none of the t pixels is black is bounded by

$$\left(\frac{9}{10}\right)^t = \left(\frac{1}{(10/9)^t}\right) = \left(\frac{1}{(10/9)^{\log_{(10/9)} n}}\right) = \frac{1}{n}.$$

Similarly, we can get any constant failure probability δ by setting $t = \log_{(10/9)} 1/\delta$.

The algorithm in Example 3.11 is a typical (much simplified) example for a sublinear algorithm that approximately solves a decision problem.[1] In this section, we want to study a different type of problem, where we want to approximate a quantity in sublinear time. Let us consider an example for this as well.

Example 3.12. *Let $a_1, \ldots, a_n \geq 0$ be n positive numbers. We want to estimate the average $a = \frac{1}{n}\sum_{i=1}^n a_i$ of these numbers, i.e., we want to compute a value $\hat{a}$ which is close to a (for example, which satisfies $|a - \hat{a}| \leq \epsilon$).*

Consider the algorithm that chooses t numbers $\hat{a}_1, \ldots, \hat{a}_t$ from $\{a_1, \ldots, a_n\}$ uniformly at random with replacement and then outputs

$$\hat{a} = \frac{1}{t}\sum_{j=1}^{t} \hat{a}_j.$$

We observe that every $\hat{a}_j$ is an unbiased estimator *for a, which just means that its expectation is a:*

$$\mathbf{E}[\hat{a}_j] = \sum_{i=1}^{n} \frac{1}{n} a_i = a.$$

[1] Problems of this type are studied in the area of *property testing*.

This also implies that $\hat{a}$ is an unbiased estimator since

$$\mathbf{E}[\hat{a}] = \mathbf{E}\left[\frac{1}{t}\sum_{j=1}^{t}\hat{a}_j\right] = \frac{1}{t}\sum_{j=1}^{t} a = a.$$

However, if $t \quad o(n)$, then $\hat{a}$ is in general not close to a. To see this, think of an example with $a_i = 0$ for $i \quad \{1, \ldots, n-1\}$ and $a_n = M$ for a large number M. Then $a = M/n$, but the probability that we sample a nonzero number (i.e., that we sample a_n), is only $1/n$. By the Union bound, the probability that we sample a_n in any of the t tries is at most t/n. So for any $t \quad o(n)$, we have a high probability that we only see 0 and that the output of the algorithm is just 0, far away from M/n.

Now we consider the algorithm from Example 3.12 in an easier scenario, where in addition to the lower bound ($a_i \geq 0$), we also know an upper bound for the given numbers. It turns out that this is crucial to bound the variance of the unbiased estimator in Example 3.12.

Lemma 3.13. *Let $a_1, \ldots, a_n$ be n numbers that satisfy $0 \leq a_i \leq L$ for $i \quad \{1, \ldots, n\}$, where $L \quad \mathsf{R}^+$. Let $\hat{a}_1, \ldots \hat{a}_t$ be chosen uniformly at random from $\{a_1, \ldots, a_n\}$ (with replacement), and set $\hat{a} = \frac{1}{t}\sum_{j=1}^{t}\hat{a}_j$. For $t = L^2/(\epsilon^2 \cdot \delta)$, it holds that*

$$\mathbf{Pr}(|\hat{a} - a| > \epsilon) \leq \delta.$$

Proof. We first observe that $\mathbf{Var}[\hat{a}_j]$ is bounded by L^2 for all $j \quad \{1, \ldots, t\}$: The average a of numbers between 0 and L lies in the interval $[0, L]$ as well, and the difference between a and any number in $[0, L]$ is thus at most L. This implies that $\mathbf{Var}[\hat{a}_j] = \mathbf{E}[(\hat{a}_j - a)^2] \leq L^2$ for all $j \quad \{1, \ldots, t\}$. Now we bound the variance of $\hat{a}$. Recall Corollary 3.5: The variance of a sum of independent random variables is the sum of the variances of the random variables. We use this to obtain that

$$\mathbf{Var}[\hat{a}] = \mathbf{Var}\left[\frac{1}{t}\sum_{j=1}^{t}\hat{a}_j\right] = \frac{1}{t^2}\sum_{j=1}^{t}\mathbf{Var}[\hat{a}_j] \leq \frac{1}{t^2}\sum_{j=1}^{t}L^2 \leq \frac{1}{t^2}t \cdot L^2 = \frac{L^2}{t}$$

where we used Lemma 3.4 for the second equality.

Now we apply Chebyshev's Inequality (Theorem 3.7):

$$\mathbf{Pr}(|\hat{a} - a| > \epsilon) \leq \frac{\mathbf{Var}[\hat{a}]}{\epsilon^2} \leq \frac{L^2}{t \cdot \epsilon^2} = \frac{L^2 \cdot \epsilon^2 \cdot \delta}{L^2 \cdot \epsilon^2} = \delta.$$

□

Estimating the number of connected components. Now we want to study the following estimation problem. We are given a graph $G = (V, E)$ with c connected components $C_1, \ldots, C_c$. The graph is stored in the adjacency model. This means that we can read the first $\geq t$ neighbors of a node v in time $\Theta(t)$. We want to output a good estimation $\hat{c}$ for the number of connected components c. For us, 'good' means that $|c - \hat{c}| \leq \epsilon n$. In the following, we develop an algorithm that is due to

Czumaj and Sohler [CS09] who improved an algorithm due to Chazelle, Rubinfeld and Trevisan [CRT05].

```
CC-Estimator-Prototype($G = (V, E)$)
    1. Choose $u$   $V$ uniformly at random, let $i$   $\{1, \ldots, c\}$ be such that $u$   $C_i$
    2. Determine the size $|C_i|$ of $C_i$
    3. return $\frac{n}{|C_i|}$
```

The above algorithm is a prototype of our sublinear algorithm. It is only a prototype since it is not clear how to implement the second step.

The prototype gives us a first idea how we could estimate the number of connected components. Let Y be the random variable that takes the output of this algorithm as its value. Then the expected value of Y is

$$\mathbf{E}[Y] = \sum_{i=1}^{c} \sum_{u \quad C_i} \frac{1}{n} \cdot \frac{n}{|C_i|} = \sum_{i=1}^{c} \sum_{u \quad C_i} \frac{1}{|C_i|} = \sum_{i=1}^{c} \frac{|C_i|}{|C_i|} = c.$$

This means that we do at least get an unbiased estimator for c. We could determine the size of a connected component by using BFS.

```
CC-Estimator-Slow($G = (V, E)$)
    1. Choose $u$   $V$ uniformly at random, let $i$   $\{1, \ldots, c\}$ be such that $u$   $C_i$
    2. Run BFS starting at $u$ to determine the size $|C_i|$ of $C_i$
    3. return $\frac{n}{|C_i|}$
```

Now we have constructed a randomized algorithm that is much like a deterministic implementation: Instead of just running BFS for every not yet discovered vertex and counting the number of necessary BFS runs to explore the whole graph, we just pick a vertex uniformly at random and explore its connected component. The problem with this is that we might end up exploring a large fraction of G. Consider a graph G_b which consists of $n/2 = c - 1$ singleton connected components and one large component that is a clique with $n/2$ nodes. Then our BFS run will go through all edges of the graph and half of the nodes if u is chosen from the large component. Our running time would thus be $\Theta(n + m)$ for this example. In order to avoid this, we have to stop our BFS before it explores too many nodes.

```
CC-Estimator-Prototype-B($G = (V, E)$)
  1. Choose $u$   $V$ uniformly at random, let $i$   $\{1, \dots, c\}$ be such that $u$   $C_i$
  2. Run a BFS starting at $u$ until
  3.    - $C_i$ is explored (success) or
  4.    - more than $s$ nodes were found (failure)
  5. If failure then  goto 1.
  6. return $\frac{n}{|C_i|}$
```

But how many nodes are ***too*** many? That is not at all clear. In G_b, any $s \geq 2$ would mean that we never explore the big clique, so we get the best running time for $s = 2$. Then the algorithm would sample nodes until it finds one of the singleton clusters (on expectation, this would happen in the second try). However, we could get an input graph where all connected components are of size 3. Then the algorithm would never terminate. In fact, this can happen to us for any s that we choose, unless we set $s = n$. The way out of this problem is to randomize s as well. Instead of fixing s, we draw s according to a cleverly chosen probability distribution.

```
CC-Estimator($G = (V, E)$)
  1. Choose $u$   $V$ uniformly at random, let $i$   $\{1, \dots, c\}$ be such that $u$   $C_i$
  2. Choose $\ell$   $N^{\geq 1}$ such that $\mathbf{Pr}(\ell = i) = \frac{1}{i(i+1)}$ for all $i$   $N^{\geq 1}$
  3. Run a BFS starting at $u$ until
  4.    - $C_i$ is explored (success) or
  5.    - at least $\ell$ nodes were found (failure)
  6. If success then
  7.    return $n$
  8. Else
  9.    return 0
```

Notice that we changed the return value as well. This will prove to be crucial for proving that the output is indeed an unbiased estimator for the number of connected components. Before we do that, let us study the probability distribution that we use to choose ℓ.

Lemma 3.14. *In Algorithm* `CC-Estimator`*, the choice of* ℓ *satisfies*

$$\mathbf{Pr}(\ell \geq i) = \frac{1}{i}$$

for all i $N^{\geq 1}$.

Proof. By definition, $\mathbf{Pr}(\ell = j) = \frac{1}{j(j+1)}$. The key observation is that $\frac{1}{j(j+1)}$ is just the difference between $1/j$ and $1/(j+1)$. We get that

$$\begin{aligned}\mathbf{Pr}(\ell \geq i) &= \sum_{j=i}^{\infty} \mathbf{Pr}(\ell = j) = \sum_{j=i}^{\infty} \frac{1}{j(j+1)} = \sum_{j=i}^{\infty} \frac{j+1-j}{j(j+1)} = \sum_{j=i}^{\infty} \left(\frac{1}{j} - \frac{1}{j+1}\right) \\ &= \left(\frac{1}{i} - \frac{1}{i+1}\right) + \left(\frac{1}{i+1} - \frac{1}{i+2}\right) + \ldots = \frac{1}{i}\end{aligned} \qquad \square$$

The previous lemma shows that the distribution is valid because $\mathbf{Pr}(\ell \geq 1) = 1$.

Notice that we explore a connected component completely if and only if we draw an ℓ that is at least as large as the number of nodes in the component. Thus, Lemma 3.14 means that the probability that we explore a component of size $|C_i|$ completely is just $1/|C_i|$. Now we can analyze the algorithm `CC-Estimator(`G`)`.

Lemma 3.15. *Let R be the result of `CC-Estimator(`G`)`. Then R satsfies $\mathbf{E}[R] = c$ and $\mathbf{Var}[R] = cn - c^2$.*

Proof. Let A_u be the event that u was chosen in the first step of `CC-Estimator(`G`)`. We observe that

$$\begin{aligned}\mathbf{E}[R] = \mathbf{Pr}(R = n) \cdot n &= n \cdot \sum_{u \;\, V} \mathbf{Pr}(R = n \cap A_u) \\ &= n \cdot \sum_{u \;\, V} \mathbf{Pr}(R = n | A_u) \cdot \mathbf{Pr}(A_u) \\ &= n \cdot \sum_{i=1}^{c} \sum_{u \;\, C_i} \mathbf{Pr}(R = n | A_u) \cdot \mathbf{Pr}(A_u) \\ &= n \cdot \sum_{i=1}^{c} \sum_{u \;\, C_i} \mathbf{Pr}(\ell \geq |C_i|) \cdot \frac{1}{n} \\ &= \sum_{i=1}^{c} \sum_{u \;\, C_i} \mathbf{Pr}(\ell \geq |C_i|) \cdot \frac{1}{n} \cdot n \\ &= \sum_{i=1}^{c} \sum_{u \;\, C_i} \frac{1}{|C_i|} = \sum_{i=1}^{c} \frac{|C_i|}{|C_i|} = c.\end{aligned}$$

Notice that we have implicitly also shown that $\mathbf{Pr}(R = n) = c/n$. We thus know all about the behavior of the random variable R: It only has two possible outputs, 0 and n, and attains the output n with probability $p = c/n$ and the output 0 with probability $(n-c)/n$. We can thus observe that $(R-c)^2$ is $(n-c)^2$ with probability c/n, and c^2 with the remaining probability and get that the variance of R is

$$\begin{aligned}\mathbf{Var}[R] = \mathbf{E}\left[(R-c)^2\right] = p(n-c)^2 + (1-p)c^2 &= \frac{c}{n}(n-c)^2 + \frac{n-c}{n}c^2 \\ &= \frac{c(n^2 - 2cn + c^2) + nc^2 - c^3}{n} \\ &= \frac{cn^2 - 2c^2n + c^3 + nc^2 - c^3}{n} \\ &= cn - 2c^2 + c^2 = cn - c^2.\end{aligned} \qquad \square$$

Now we try to apply Chebyshev's inequality (Theorem 3.7) to obtain a bound on the deviation from c:

$$\mathbf{Pr}(|R - c| > \epsilon n) \leq \frac{\mathbf{Var}[R]}{(\epsilon n)^2} = \frac{cn - c^2}{(\epsilon n)^2} < \frac{cn}{\epsilon^2 n^2} \leq \frac{1}{\epsilon^2}.$$

This bound is useless: $|R - c|$ always lies in $\{0, \ldots, n\}$ by definition, so the statement only makes sense for $\epsilon < 1$. But for $\epsilon < 1$, the probability for $|R - c| \leq \epsilon n$ is now bounded by $1/\epsilon^2 > 1$, and 1 is a trivial upper bound for any probability. Usually, when we have an algorithm that fails with probability p_f, then we use independent repetitions and show that $(1 - p_f)^t$ is small. Notice that $(1 - p_f)^t$ is only decreasing in t when $p_f < 1$. Thus, this approach will not help us. However, we have a new tool at our disposal: Corollary 3.5 implies that independent repetitions decrease the variance (we already used this in the proof of Lemma 3.13). Thus, to show better bounds, we can first show that independent repetitions decrease the variance sufficiently, and then apply Chebyshev. Here is our final algorithm for estimating the number of connected components:

```
CC-Algorithm($G = (V, E), \epsilon, \delta$)
  1. Set $r = \frac{1}{\delta\epsilon^2}$
  2. for $i = 1$ to $r$ do
  3.     $R_i$ = CC-Estimator($G$)
  4. return $\hat{R} = \frac{1}{r}\sum_{i=1}^{r} R_i$
```

Theorem 3.16. *The result $\hat{R}$ of CC-Algorithm(G, ϵ, δ) satisfies $\mathbf{E}\left[\hat{R}\right] = c$, $\mathbf{Var}\left[\hat{R}\right] = \delta\epsilon^2(cn - c^2)$ and*

$$\mathbf{Pr}(|\hat{R} - c| \geq \epsilon n) \leq \delta.$$

Proof. By Corollary 3.5,

$$\mathbf{Var}\left[\hat{R}\right] = \mathbf{Var}\left[\sum_{i=1}^{r} \frac{1}{r} R_i\right] = \frac{1}{r^2}\sum_{i=1}^{r} \mathbf{Var}[R_i] = \frac{1}{r^2} \cdot r \cdot (cn - c^2) = \frac{cn - c^2}{r} = \delta\epsilon^2(cn - c^2)$$

where we inserted the definition of r for the last equation. Now we apply Chebyshev's inequality (Theorem 3.7) and get

$$\mathbf{Pr}(|\hat{R} - c| \geq \epsilon n) \leq \frac{\mathbf{Var}\left[\hat{R}\right]}{(\epsilon n)^2} = \frac{\delta\epsilon^2(cn - c^2)}{(\epsilon n)^2} = \frac{\delta \cdot (cn - c^2)}{n^2} < \frac{\delta \cdot cn}{n^2} \leq \delta$$

since $c \leq n$. □

Now we are nearly done. What remains to do? We started with the idea of a sublinear algorithm in mind. This is why we randomized the size of the subgraph that we explore by our BFS. However, this means that now the running time is a random variable, too. We conclude the discussion of CC-Algorithm(G, ϵ) by bounding its expected running time.

Theorem 3.17. *The expected running time T of CC-Algorithm(G, ϵ, δ) is in $O(D \cdot \delta^{-1} \cdot \epsilon^{-2} \cdot \log n)$, where D is the maximum degree of any vertex in G.*

Proof. Let T_i be the running time of the ith run of `CC-Estimator(`G`)`. For any graph $G = (V, E)$, the running time of BFS is bounded by $O(|V| + |E|)$. Let $G_i = (V_i, E_i)$ be the subgraph that is explored in the ith run of `CC-Estimator(`G`)`, i.e., G_i consists of all vertices that are seen by the BFS in this run, and the edges between them. Since the graph is in adjacency list representation, we can make sure that G_i has at most $\ell + 1$ nodes, and that we do not see more than one edge that points 'out' of G_i (ends in a vertex that is not in V_i). Thus, T_i is bounded by $O(|V_i| + |E_i|)$. We could bound $|E_i|$ by $|V_i|^2$. Another possibility is to bound it by $D \cdot |V_i|$, since every vertex has at most D neighbors. We choose the latter and define Δ to be a large enough constant such that T_i is bounded by $\Delta D|V_i| \leq \Delta D\ell_i$ for all i $\{1, \ldots, r\}$, where we let ℓ_i be the value of ℓ in the ith run. Notice that $\ell_i \leq n$ because we can at most explore the whole graph. Thus, although we theoretically allow arbitrarily large values of ℓ, ΔDn is always an upper bound for T_i. We obtain that

$$\begin{aligned}
\mathbf{E}[T_i] &\leq \Delta Dn \cdot \mathbf{Pr}(\ell_i \geq n) + \sum_{j=1}^{n-1} \Delta Dj \cdot \mathbf{Pr}(\ell_i = j) \\
&= \Delta Dn \cdot \frac{1}{n} + \Delta D \cdot \sum_{j=1}^{n-1} j \cdot \frac{1}{j \cdot (j+1)} \\
&= \Delta D + \Delta D \cdot \sum_{j=1}^{n-1} \frac{1}{j+1} \\
&= \Delta D(1 + H_n - 1) \quad O(D \log n).
\end{aligned}$$

We get a total running time in $O(D\delta^{-1}\epsilon^{-2} \log n)$ since `CC-Algorithm(`G, ϵ, δ`)` executes $r = \ \delta^{-1}\epsilon^{-2}\ $ runs of `CC-Estimator(`G`)`. □

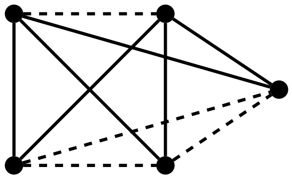
Outlook: Minimum Spanning Trees Czumaj and Sohler [CS09] and before them, Chazelle, Rubinfeld and Trevisan [CRT05], developed algorithms for the estimation of the number of connected components in order to obtain algorithms that estimate the weight of minimum spanning trees (MSTs) in graphs. While their work on MSTs goes beyond the scope of this lecture, we still want to have a glimpse at the connection between the two topics. For this, we discuss a special case.

In addition to a graph $G = (V, E)$, our input now contains weights for all edges. Our special case is that the input graph is the complete graph, and all weights are either 1 or 2, i.e., we get a weight function $w : V \times V \to \{1, 2\}$. Under this condition, we want to compute a good estimation $\hat{M}$ for the value M of a minimum spanning tree in G. Here, good means that $|\hat{M} - M| \leq \epsilon M$. For our analysis, we need the following fact about forests (an undirected graph is a forest if it has no cycles).

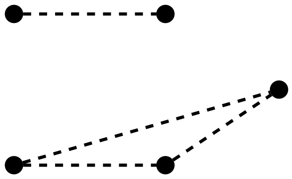
Fact 3.18. *Let F be a forest with n nodes and i connected components. Then the number of edges in F is $n - i$.*

For the special case of a tree, Fact 3.18 says that a tree has $n - 1$ edges, since a tree has 1 connected component. For n singleton nodes, we have $n - n = 0$ edges. Adding an edge will decrease the number of connected components by 1, thus the formula in Fact 3.18 makes sense.

Consider the following example, where we see a graph with five nodes. The dashed edges have weight 1, the solid edges have weight 2. The MST value M in this example is 5.



Now consider the following graph, where we leave out all edges of weight 2. We see that this graph has two connected components. For a minimum spanning tree, we will connect all nodes within a connected components by 1-edges. Observe that a solution constructed like this is a forest with 5 nodes and 2 connected components. Thus, it has 3 edges. In order to get a spanning tree, we need to connect the connected components by 2-edges. In our example, we only need one 2-edge, thus we pay a total of $3 + 2 = 5$.



In general, let $G^{(1)} = (V, E^{(1)})$ be the graph where we drop all 2-edges, i.e., $E^{(1)} = \{e \quad E \mid w(e) = 1\}$. Let $c^{(1)}$ be the number of connected components in this graph. Connecting all vertices within the components gives a forest on n vertices with $c^{(1)}$ connected components, i.e., a forest with $n - c^{(1)}$ edges. For a spanning tree, we need to connect the connected components. We can imagine this as connecting $c^{(1)}$ 'nodes' with a tree, which tells us that we need $c^{(1)} - 1$ edges. Since these are of weight 2, we pay $2(c^{(1)} - 1)$ for the second step. Together with the $n - c^{(1)}$ edges from the first step, the cost of any minimum spanning tree is $n + c^{(1)} - 2$. We have thus developed the following formula:

Lemma 3.19. *For a graph $G = (V, E)$ with weights $w : V \times V \to \{1, 2\}$ and $c^{(1)}$ as defined above, it is true that the value of a minimum spanning tree in G satisfies*

$$M = n + c^{(1)} - 2.$$

So, all we need to do is to estimate the number of connected components in $G^{(1)}$! Observe that we do not have $G^{(1)}$, and we also cannot compute $G^{(1)}$ in sublinear time. However, we can use G within the estimation algorithm and just ignore all edges of weight 2. That will simulate $G^{(1)}$, and it does not increase the running time beyond our upper bound.

```
MST-12-Estimator($G = (V, E), w : V \times V \to \{1, 2\}, \epsilon, \delta$)
    1. Compute $\hat{c}^{(1)}$ = CC-Algorithm($G^{(1)}, \epsilon, \delta$)
    2. return $\hat{M} = n + \hat{c}^{(1)} - 2$
```

Theorem 3.20. *MST-12-Estimator has an expected running time of O$(D\delta^{-1}\epsilon^{-2}\log n)$. Its result $\hat{M}$ of MST-12-Estimator satisfies*

$$\mathbf{Pr}(|\hat{M} - M| \geq \epsilon \cdot M) \leq \delta$$

Proof. The running time bound follows because the algorithm consists of one call to CC-Algorithm($G^{(1)}, \epsilon, \delta$) and doing one additional computation. For the concentration bound, we observe that

$$\begin{aligned}\mathbf{Pr}(|\hat{M} - M| \geq \epsilon \cdot M) &= \mathbf{Pr}(|n + \hat{c}^{(1)} - 2 - n - c^{(1)} + 2| \geq \epsilon \cdot (n + c^{(1)} - 2)) \\ &= \mathbf{Pr}(|\hat{c}^{(1)} - c^{(1)}| \geq \epsilon \cdot (n + c^{(1)} - 2)) \leq \delta.\end{aligned}$$

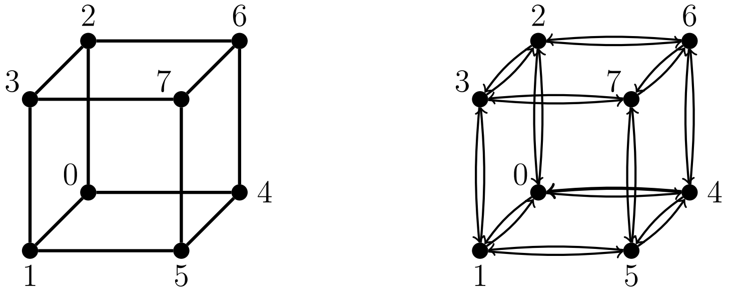
For the last line, distinguish two cases: If $c^{(1)} > 1$, then $n + c^{(1)} - 2 \geq n$, thus we get exactly the expression which we bounded in Theorem 3.16. For $c^{(1)} = 1$, recall that the variance of $\hat{c}^{(1)}$ is $\frac{1}{r} \cdot (c^{(1)}n - (c^{(1)})^2) = (n-1)/r$, so for this case we get that the probability is bounded by

$$\frac{(n-1)/r}{\epsilon^2(n-1)^2} = \delta\frac{n-1}{(n-1)^2} < \delta.$$

□

3.3.2 Routing in Hypercubes

An n-dimensional cube is called a ***hypercube***. A standard definition of a hypercube is undirected, has 2^n vertices and $n2^{n-1}$ edges since every vertex has undirected edges to n other vertices. We define a directed version, where every undirected edge is replaced by two directed edges of opposite direction, thus there are $n2^n$ edges in total.

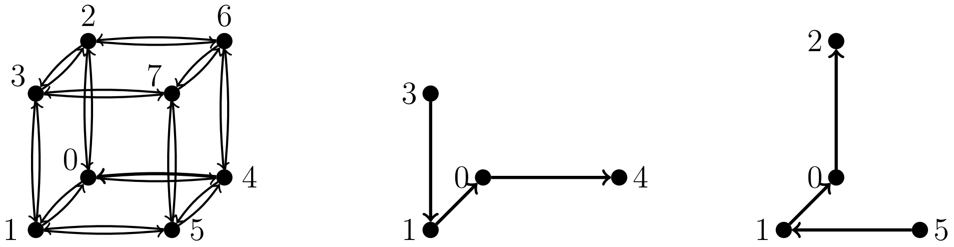


For an integer i $\{0, \dots, 2^n - 1\}$, let $\texttt{bin}(i)$ be its binary representation. For two bit strings x, y $\{0, 1\}^n$, let $h(x, y)$ be the ***Hamming distance*** of x and y, i.e., the number of bits in which x and y differ.

Definition 3.21. *Let $H_n = \{V_n, E_n\}$ be the directed graph defined by*

$$\begin{aligned} V_n &= \{0, \dots, 2^n - 1\} \\ E_n &= \{(i, j) \mid h(\texttt{bin}(i), \texttt{bin}(j)) = 1\} \end{aligned}$$

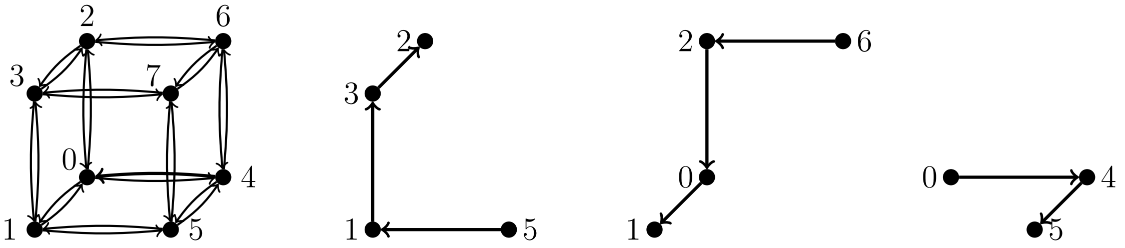
Our problem consists of simultaneously sending 2^n packets between the vertices of H_n. Each vertex i V_n is the starting point of one packet which we name a_i. The input is a permutation $\pi : V_n \to V_n$ that defines the destination $\pi(i)$ for a_i for every i V_n. The routing strategy has to choose one path $P(i)$ from i to $\pi(i)$ for every i V_n. The execution of a routing (consisting of paths $P(i)$ for all i V_n) happens in discrete time steps. At each time step, at most one packet can travel over each edge. If a packet travels over an edge, it arrives at the end of the edge at the end of the time step. Every packet a_i follows its path $P(i)$. When a packet wants to travel over an edge, then it enters a FIFO queue at the start of the edge (if two packets arrive at the same time, an arbitrary but consistent tie-breaking is used). If the new packet is the only packet in the queue, then it can use the edge immediately. Otherwise, the first packet in the queue travels over the edge, while the other packets in the queue wait. In the following example, we see conflicting paths for two packets. It takes four time steps until these two packets arrive.



Let $|P(i)|$ be the number of edges on $P(i)$. Then a packet a_i needs at least $|P(i)|$ steps to reach $\pi(i)$, but it might need considerably more steps if it is delayed. The ***execution time*** of a routing is the time that it takes until all packets have arrived at their destination. We want to minimize the execution time.

We look for an ***oblivious routing*** strategy. That means that the path $P(i)$ may depend only on i and $\pi(i)$ and is not influenced by the destinations of other packets. Thus, the problem is to avoid that many packets choose the same path, but with a strategy that computes the path only based on start and end vertex (and possibly on random decisions).

Bit fixing paths. ***Bit fixing*** is a deterministic oblivious routing strategy for the hypercube. It "fixes" the bits in $\texttt{bin}(i)$ from left to right (i.e., starting at the most significant bit) such that the number i becomes $\pi(i)$. For example, if the packet a_5 wants to travel from 5 to 2 in H_3, then the bit fixing path travels through the vertices with the binary representations 101, 001, 011 and 010 in this order. The bit fixing path from 6 to 1 visits the vertices with the binary representations 110, 010, 000, 001 in this order, and the path from 0 to 5 visits 000, 100 and 101.



In general, let $x_1 \dots x_n$ be the binary representation of i and let $y_1 \dots y_n$ be the binary representation of $\pi(i)$. The bit fixing path $P(i)$ has $h(\mathtt{bin}(i), \mathtt{bin}(\pi(i)))$ edges. Let ℓ_m for $m \quad \{0, \dots, h(\mathtt{bin}(i), \mathtt{bin}(\pi(i)))\}$ be the index of the mth bit in which $\mathtt{bin}(i)$ and $\mathtt{bin}(\pi(i))$ differ. Then the mth edge on $P(i)$ goes from the vertex with binary representation $y_1 \dots y_{\ell_m - 1} x_{\ell_m} \dots x_n$ to the vertex with binary representation $y_1 \dots y_{\ell_m - 1} y_{\ell_m} x_{\ell_m + 1} \dots x_n$. Notice that this is well-defined: Only one bit changes, so two consecutive vertices on $P(i)$ are indeed always connected in H_n. Furthermore, the path $P(i)$ indeed starts in i and ends in $\pi(i)$. Finally, observe that $|P(i)| = h(\mathtt{bin}(i), \mathtt{bin}(\pi(i))) \leq n$ and that a bit fixing path is always a shortest path between i and $\pi(i)$. The latter holds because every edge in E_n connects two vertices with Hamming distance one, so any path between i and $\pi(i)$ has at least $h(\mathtt{bin}(i), \mathtt{bin}(\pi(i))$ edges.

Collisions and large worst case execution times. In our above example, the three bit fixing paths do not intersect. However, consider the following family of instances. Let n be a positive even integer. For any $i \quad \{0, \dots, 2^n - 1\}$, we define $\pi(i)$ in the following way. Let $x_1 \dots x_n$ be the binary representation of i. Then $\pi(i)$ is in binary given by

$$x_{\frac{n}{2}+1} \dots x_n x_1 \dots x_{\frac{n}{2}},$$

i.e., the first and second part of the binary representation are swapped. The problem is that for all i with the same last $n/2$ bits, the bit fixing paths for this instance intersect in a vertex. Let $z_{\frac{n}{2}+1} \dots z_n$ be $n/2$ bits and let the binary representation of i be

$$x_1 \dots x_{\frac{n}{2}} z_{\frac{n}{2}+1} \dots z_n.$$

Then $\pi(i)$ has the binary representation

$$z_{\frac{n}{2}+1} \dots z_n x_1 \dots x_{\frac{n}{2}},$$

and the bit fixing path from i to $\pi(i)$ contains the vertex with the binary representation

$$z_{\frac{n}{2}+1} \dots z_n z_{\frac{n}{2}+1} \dots z_n.$$

There are $2^{n/2}$ numbers in $\{0, \dots, 2^n - 1\}$ with $z_{\frac{n}{2}+1} \dots z_n$ as the last $n/2$ bits of their binary representation. All these numbers have the same vertex i in their bit fixing path. One of them is the number i itself, which has i as destination, but all $2^{n/2} - 1$ other numbers want to leave i again. In every time step, at most n packets can leave i. Thus, it takes at least $(2^{n/2} - 1)/n$ steps until all packets have left i, and that is a lower bound for the execution time of the whole routing strategy. Thus, the worst case execution time is $\Omega(\sqrt{2^n}/n)$ for the routing strategy that uses bit fixing paths for all packets. Surprisingly, this is true for any oblivious deterministic routing strategy.

Theorem 3.22 (Kaklamanis, Krizanc, Tsantilas [KKT91]). *For any oblivious deterministic routing strategy in H_n, there exists a permutation $\pi : V_n \to V_n$ such that the execution time of the routing strategy is $\Omega(\sqrt{2^n}/n)$ for π.*

The paper [KKT91] also contains a deterministic routing strategy that matches this worst case execution time. In light of the routing time n of a single packet, we would like a faster strategy. Randomization will provide us with a way out.

Valiant's randomized routing strategy. The following strategy is attributed to Valiant and is published in [VB81, Val82]. While there exist permutations for which bit fixing paths have a lot of collisions, this is not the case if the permutation is chosen randomly (we do not prove this statement, but it is implicit in the proof that we will see). However, we do not want to force the input to be random, we want the algorithm to be randomized. Valiant's idea is to send each packet a_i to $\pi(i)$ in two phases: First, the packet is sent to a random intermediate location $\delta(i)$. Second, it is sent from $\delta(i)$ to $\pi(i)$. In this way, each phase has a random component: packets are either sent to or from a random location. With high probability, bit fixing routings will be fast for both phases.

The intermediate location $\delta(i)$ is chosen uniformly at random from all V_n for all i V_n. Thus, δ is not necessarily a permutation and packets might be routed to the same intermediate location. However, the number of packets routed to the same location will with high probability not be large enough to cause high congestion in the graph. To make the two phases of the algorithm independent, we assume that all packets wait until step $4n + 1$ until they start with phase 2. We will see that with high probability, phase 1 is completed in $4n$ time steps. Phase 2 then takes at most $4n$ more time steps with high probability. The remaining part of this section consists of proving the following theorem.

Theorem 3.23. *For any $\pi : V_n \to V_n$, Valiant's randomized routing strategy has an execution time of at most $8n$ steps with probability $1 - \frac{1}{2^n}$.*

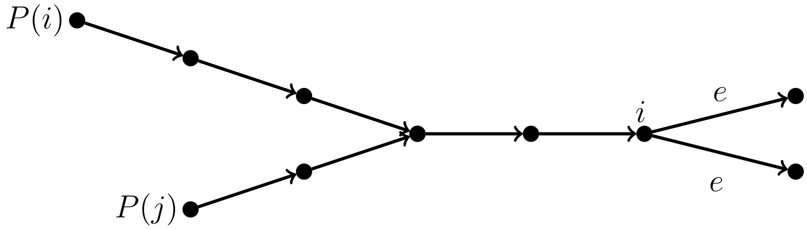
Before proving the theorem, we develop structural insights about bit fixing paths. For any i V_n let $P(i)$ be the bit fixing path from i to $\delta(i)$ during phase 1, i.e., $P(i)$ is a bit fixing path to a location chosen uniformly at random from V_n. By $|P(i)|$ we mean the number of edges on $P(i)$, and by $P(i) \cap P(j)$ for i, j V_n we mean the set of edges that are on both paths. We define the set

$$S(i) = \{j \mid j = i, P(i) \cap P(j) = \quad \}$$

which is the set of all vertices j such that the bit fixing path of a_j crosses the path of a_i. The packets $a_j, j \quad S(i)$ are all the packets that can possibly delay a_i when it travels to $\delta(i)$. We investigate how and how often a packet $a_j, j \quad S(i)$ can delay a_i. The following lemma shows that two bit fixing paths can only cross in one contiguous subpath.

Lemma 3.24. *For all $i \quad V_n$ and all $j \quad S(i)$, $P(i) \cap P(j)$ is a contiguous subpath of $P(i)$ and of $P(j)$.*

Proof. Let i be the last vertex of the first subpath of $P(i)$ that also lies on $P(j)$. We want to show that i is the last vertex on $P(i)$ and $P(j)$ that lies on both paths. Let e and e be the edges of $P(i)$ and $P(j)$, respectively, that leave i .



Then e and e flip different bits in the binary representation of i . Let

$$\texttt{bin}(i\ ) = x_1 \dots x_s \dots x_t \dots x_n$$

where x_s and x_t with $s < t$ are the bits that are inverted, i.e., e goes to the vertex with binary representation

$$x_1 \dots \bar{x}_s \dots x_t \dots x_n$$

and e goes to the vertex with binary representation

$$x_1 \dots x_s \dots \bar{x}_t \dots x_n$$

or vice versa. Then all future nodes on $P(i)$ and $P(j)$ differ in the sth bit, thus the paths cannot cross again. □

We learned that a packet $a_j, j \quad S(i)$ might delay a_i during one contiguous subpath, but once $P(i)$ and $P(j)$ diverge for the first time, a_j can no longer delay a_i. We use this fact to prove the following statement.

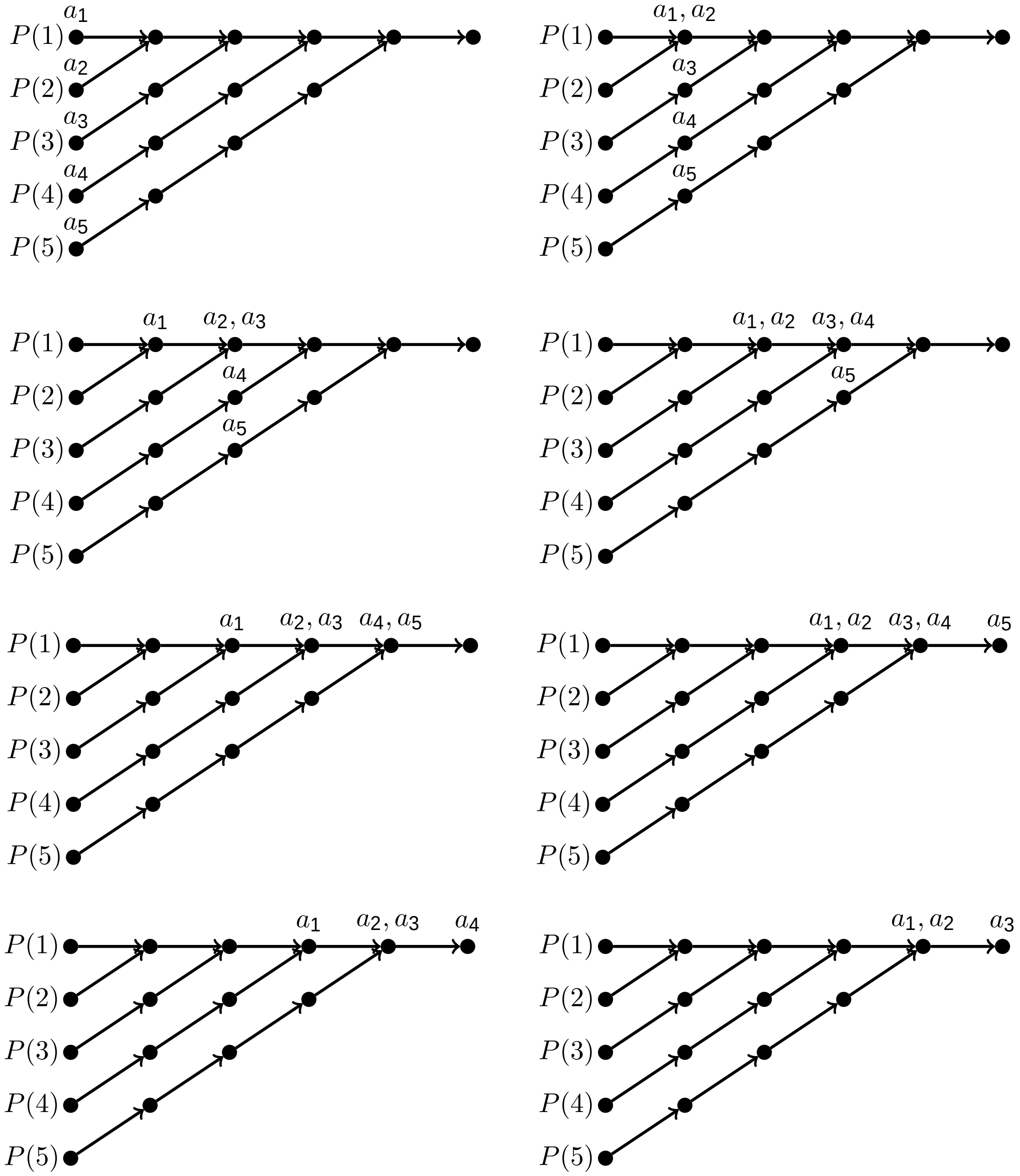
Lemma 3.25. *Let $T(i)$ be the number of steps that a_i needs to arrive at $\delta(i)$ and define the* delay *of a_i by $D(i) = T(i) - |P(i)|$. Then it holds*

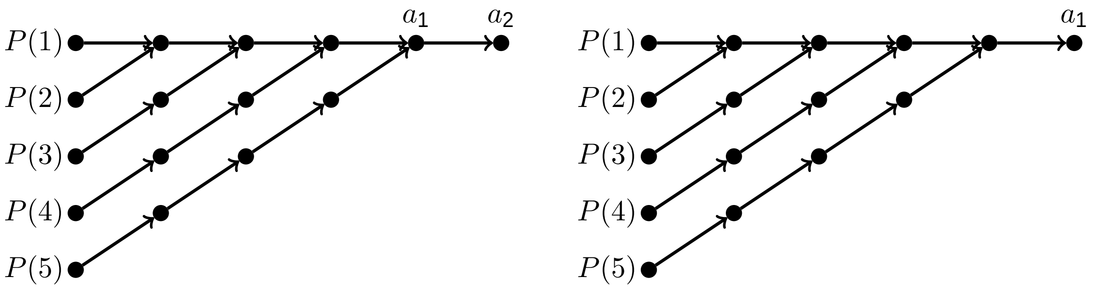
$$D(i) \leq |S(i)|$$

for all $i \quad V_n$.

Proof. The packet a_i waits $D(i)$ steps because it is delayed by other packets, i.e., its delay goes from 0 to $D(i)$ in steps of one. We would like to charge a distinct packet in $S(i)$ for the increase from L to $L+1$ for all $L \quad \{0, \ldots, D(i) - 1\}$. The packet that we charge is not necessarily the packet that caused the delay for a_i. It is only important that no packet in $S(i)$ gets charged twice.

In the following example, a_1 wants to travel to the rightmost vertex, the destinations of all other packets lie beyond this vertex. We assume that the FIFO queue breaks ties by sending the packet with the higher index first. Even though $P(1)$ has only five edges, a_1 needs nine steps to reach its destination. It is always delayed by the same packet a_2. However, the delay is still bounded by the number of crossing paths, since $P(1)$ is crossed by $P(2)$, $P(3)$, $P(4)$ and $P(5)$.





Let $P(i)$ consist of the edges $e_1, \ldots, e_{|P(i)|}$ (in this order) and of the vertices $v_1, \ldots, v_{|P(i)|+1}$ (in this order). We define the ***offset*** of a packet a_j with $j \in S(i) \setminus \{i\}$ at time t as

$$\text{offset}(t, j) = t - m$$

iff a_j is at vertex v_m at time t (if a_j is not on $P(i)$ at time t, then the offset is undefined). Thus, the offset says whether a_j is ahead or behind a_i's schedule. In particular, offset(t, i) is the delay of a_i at the beginning of step t. The packet a_i can only be delayed by packets with the same offset because only these packets are at the same vertex and potentially want to travel over the same edge as a_i.

In our example above, $i = 1$ and the upper path is $P(i) = P(1) = (v_1, \ldots, v_6)$. We see that a_1 is at v_1 at the beginning of step 1 and travels to v_2 in the first time step. Thus, offset$(1, 1) = 0$ and offset$(2, 1) = 0$. Then it waits a time step because the edge (v_2, v_3) is blocked. Thus, offset$(3, 1) = 1$. In step 3, a_1 travels to v_3, thus it is at v_3 at the beginning of step 4 and offset$(4, 1) = 1$. Now it is delayed again, it is still at v_3 at the beginning of step 5, implying that offset$(5, 1) = 2$. For a_1, the offset is the delay. Now consider a_2. It arrives on $P(1)$ after one time step and is at v_2 at the beginning of the second time step, i.e., offset$(2, 2) = 0$. Also offset$(3, 2) = 0$ because a_2 travels in step 2. However, then it waits, implying that offset$(4, 2) = 1$ because a_2 is still at v_3 at the beginning of step 4. We see that a_1 is delayed twice by a_2 when both have the same offset: At time 2, both have offset 0, and at time 4, both have offset 1.

Observe that for a given offset L, a packet j with offset$(t, j) = L$ is at vertex v_{t-L} at time t. If the increase in a_i's delay from L to $L+1$ happens at time t, then it happens because of a packet which is also at vertex v_{t-L} at time t and wants to travel over the edge e_{t-L}. Now we define

$$\text{lucky}(t, L) = \{j \mid j \in S(i), a_j \text{ travels along } e_{t-L} \text{ in step } t\}$$

for all $L \in \{0, \ldots, D(i)-1\}$ and for all $t \in \{1+L, \ldots, |P(i)|+L-1\}$ as the lucky packet with offset L that actually travels over e_{t-L} in step t. Observe that lucky(t, L) either contains no or exactly one index. In our example, lucky$(1, 0) = \emptyset$, lucky$(2, 0) = \{2\}$, lucky$(3, 0) = \{3\}$, lucky$(4, 0) = \{4\}$, lucky$(5, 0) = \{5\}$ and lucky$(6, 0)$ is not defined because there is no edge e_6. For offset 1, we observe lucky$(2, 1) = \emptyset$, lucky$(3, 1) = \emptyset$, lucky$(4, 1) = \{2\}$, lucky$(5, 1) = \{3\}$ and lucky$(6, 1) = \{4\}$ in the example.

Assume that the increase of a_i's delay from L to $L+1$ happens in step t. Thus, lucky(t, L) is nonempty because another packet caused a_i to wait during step t. Now let t^* be the largest $t^* \in \{1+L, \ldots, |P(i)|+L-1\}$ such that lucky(t^*, L) is nonempty.

We charge the increase of a_is delay from L to $L + 1$ to the a_j with $j \quad \text{lucky}(t\ , L)$. In the example, we charge the increase from 0 to 1 to a_5 because it is the last packet that travels at offset 0. The increase from 1 to 2 is charged to a_4 and so on.

Assume that a packet $a_j, j \quad S(i)$ gets charged twice by this process, once for increasing the delay from L_1 to $L_1 + 1$ and once for increasing it from L_2 to $L_2 + 1$. By definition, $j \quad \text{lucky}(t_1, L_1)$ and $j \quad \text{lucky}(t_2, L_2)$ for $t_1, t_2 \quad \{1 + L, \ldots, |P(i)| + L - 1\}$. Without loss of generality, assume that $t_1 < t_2$. Since $t_1 + 1 \leq t_2 \leq |P(i)| + L - 1$, $\text{lucky}(t_1 + 1, L_1)$ is well-defined and it is empty by definition of t_1. Observe that j has not reached the end of $P(i)$ at time t_1 because it is still on $P(i)$ at time $t_2 > t_1$. However, it did not travel on the next edge, since otherwise, $\text{lucky}(t_1 + 1, L_1)$ would not be empty. It cannot have been blocked by another packet either, because this packet then would be in $\text{lucky}(t_1 + 1, L_1)$ (notice that packets that are not in $S(i)$ will never travel along edges of $P(i)$ and can thus not block a_j from advancing on $P(i)$). Thus, a_j did not want to travel the next edge. It either arrived at its destination at time t_1 or its path deviated from $P(i)$. If it arrived at its destination, then j cannot be in $\text{lucky}(t_2, L_2)$ because a_j stopped moving at time t_1. If $P(j)$ deviated from $P(i)$, then by Lemma 3.24, $P(j)$ never joins $P(i)$ again. Thus, j cannot be in $\text{lucky}(t_2, L_2)$ in this case either. We conclude that a_j cannot be charged twice.

Since no $j \quad S(i)$ can be charged twice, we proved that $D(i) \leq |S(i)|$. □

Until now, our analysis had no probabilistic component. Now, we use the structural insights that we gained to bound the probability that $|S(i)|$ and thus $D(i)$ is very large.

Lemma 3.26. *For all* $i \quad V_n$*, it holds that*

$$\mathbf{Pr}(D(i) > 3n) \leq \mathbf{Pr}(|S(i)| > 3n) \leq 2^{-3n}.$$

Proof. For all $e \quad P(i)$, let $S(i, e) := \{j \mid j \quad S(i), e \quad P(j)\}$, i.e., $j \quad S(i, e)$ iff it is in $S(i)$ and $P(j)$ contains e. Notice that the $S(i, e)$ are not necessarily pairwise disjoint, but we know that $S(i) = \quad _{e \quad P(i)} S(i, e)$ and that implies that $|S(i)| \leq \sum_{e \quad P(i)} |S(i, e)|$. By Observation 2.6 and linearity of expectation, $\mathbf{E}[|S(i)|] \leq \sum_{e \quad P(i)} \mathbf{E}[|S(i, e)|]$. Now we bound the expected value of $|S(i, e)|$. We do this by using the indicator variables

$$X(i, j, e) = \begin{cases} 1 & \text{if } e \quad P(i) \cap P(j) \\ 0 & \text{else} \end{cases}$$

for all $i, j \quad V_n$, $i = j$, $e \quad P(i)$. Observe that

$$|S(i, e)| = \sum_{j \quad V_n} X(i, j, e), \text{ thus } \mathbf{E}[|S(i)|] \leq \sum_{e \quad P(i)} \mathbf{E}[|S(i, e)|] = \sum_{e \quad P(i)} \sum_{j \quad V_n} \mathbf{E}[X(i, j, e)]\,.$$

For an edge $e \quad P(i)$, let i be the starting node of e and let i be the end node of e. Let r be the index of the bit in which $\mathtt{bin}(i\ )$ and $\mathtt{bin}(i\ \ )$ differ. Let

$$\mathtt{bin}(i\ ) = x_1 \ldots x_{r-1} x_r x_{r+1} \ldots x_n, \qquad \text{implying that}$$

$$\texttt{bin}(i\ ) = x_1 \dots x_{r-1}\bar{x}_r x_{r+1} \dots x_n.$$

Let $j \quad S(i,e)$. Thus, the bit fixing path from j to $\delta(j)$ visits i and i in this order. By the definition of the bit fixing strategy, that means that $\texttt{bin}(j)$ is of the form

$$u_1 \dots u_{r-1} x_r x_{r+1} \dots x_n$$

for $u_1, \dots, u_{r-1} \quad \{0,1\}$. Notice that there are 2^{r-1} numbers of this form because that is the number of choices for $u_1 \dots u_{r-1}$. This means that for a given $i \quad V_n$ and a given $e \quad P(i)$, at most 2^{r-1} indicator variables $X(i,j,e)$ can be one, all other are zero because j will never be routed through i , no matter what $\delta(j)$ is. Let $J(i,e)$ be this set of the 2^{r-1} bad starting locations that can possibly be routed through e.

The destination $\delta(j)$ is chosen uniformly at random from V_n. Even if $j \quad J(i,e)$, the bit fixing path to $\delta(i)$ only visits i and i if $\delta(j)$ is of the form

$$x_1 \dots x_{r-1}\bar{x}_r z_{r+1} \dots z_n$$

for $z_{r+1}, \dots, z_n \quad \{0,1\}$. Since the first r bits are fixed, there are 2^{n-r} numbers of this form. The probability that one of them is chosen out of the 2^n possible values for $\delta(j)$ is $2^{n-r}/2^n = 1/2^r$ because $\delta(j)$ is chosen uniformly at random from V_n. Thus, for $j \quad J(i,e)$, the probability for $X(i,j,e) = 1$ is $1/2^r$. We get that

$$\begin{aligned} \mathbf{E}[|S(i)|] &\leq \sum_{e \ \ P(i)} \mathbf{E}[|S(i,e)|] = \sum_{e \ \ P(i)} \sum_{j \ \ V_n} \mathbf{E}[X(i,j,e)] = \sum_{e \ \ P(i)} \sum_{j \ \ J(i,e)} \mathbf{E}[X(i,j,e)] \\ &= \sum_{e \ \ P(i)} |J(i,e)| \cdot \frac{1}{2^r} \leq n \cdot 2^{r-1} \cdot \frac{1}{2^r} = \frac{n}{2} \end{aligned}$$

because $P(i)$ has at most n edges. We now know that $n/2$ is an upper bound on $\mathbf{E}[|S(i)|]$ and thus $3n \geq 6\mathbf{E}[|S(i)|]$. Furthermore, $|S(i)|$ is a sum of independent Bernoulli random variables which means that we can apply Theorem 3.9. It yields that

$$\mathbf{Pr}(|S(i)| > 3n) \leq 2^{-3n}.$$

Applying Lemma 3.25 concludes the proof. □

Notice that Lemma 3.24 and Lemma 3.25 have purely deterministic arguments. However, for the deterministic routing strategy that just uses bit fixing paths without intermediate locations, we saw an example where a lot of packets are routed through the same vertex, and, ultimately, over the same edges. This means that $S(i)$ is large for a lot of vertices.

Furthermore, the lemmas follow in the same way if $P(i)$ is the bit fixing path from from $\delta(i)$ to $\pi(i)$ in phase 2 and $D(i)$ is the delay of a_i in phase 2. Lemma 3.26 uses that the destination is chosen uniformly at random. However, the argument works analogously for phase 2, where the starting point is chosen uniformly at random. The following lemma can thus be proven similarly to Lemma 3.26.

Lemma 3.27. *Let $P\ (i)$ be the bit fixing path from $\delta(i)$ to $\pi(i)$. Let $T\ (i)$ be the number of steps that a_i needs to travel from $\delta(i)$ to $\pi(i)$ and define $D\ (i) = T\ (i) - |P\ (i)|$. For all $i \quad V_n$, it holds that*

$$\mathbf{Pr}(D\ (i) > 3n) \leq 2^{-3n}.$$

Now the proof of the following theorem is nearly finished.

Theorem 3.23. *For any $\pi : V_n \to V_n$, Valiant's randomized routing strategy has an execution time of at most $8n$ steps with probability $1 - \frac{1}{2^n}$.*

Proof. The probability that $D(i)$ is more than $3n$ is at most 2^{-3n}, thus the probability that a_i arrives after more than $4n$ time steps is at most 2^{-3n} as well since $P(i)$ has at most n edges. By the Union Bound, the probability that there is at least one packet that takes more than $4n$ steps is bounded by $2^n \cdot 2^{-3n} = 2^{-2n}$. Thus, phase 1 is finished after $4n$ steps with probability $1 - 2^{-2n}$, and the same holds for phase 2. The probability that the algorithm is successful is thus at least

$$1 - 2^{-2n} - 2^{-2n} \geq 1 - 2 \cdot 2^{-2n} \geq 1 - 2^{-n}.$$

□

Chapter 4

Random Walks

A random walk in a graph is a process that moves between the vertices of the graph in a random way. The graph can be described explicitly or implicitly, in the latter case, the random walk might describe a process that changes its state every turn. We start the chapter with useful facts about factorials and binomial coefficients. Then we see two examples for analyzing random walks. The applications where these random walks occur are k-SAT for $k = 2$ and $k = 3$.

Section 4.2 corresponds to 7.1 in [MU05].

4.1 Stirling's Formula and the Binomial Theorem

Stirling's formula provides an approximation for the factorial of a number. It is due to Abraham de Moivre and James Stirling and states that $n!$ is approximately the same as $(n/e)^n$ times a smaller term, and that the smaller term is $\sqrt{2\pi n}$. For a proof of the following lemma, see for example 2.27 in [Mit70] (statement (6) in 2.27 gives tighter bounds from which the lemma follows).

Lemma 4.1 (Stirling's formula)**.** *It holds*

$$\sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \leq n! \leq 2\sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n$$

for all integers $n \geq 1$*.*

Lemma 4.1 implies the following often used bounds for the binomial coefficient.

Corollary 4.2. *For all integers* $n \geq k > 0$*, it holds that*

$$\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{n \cdot e}{k}\right)^k.$$

Proof. Exercise. □

For our 3-SAT analysis, we will need a lower bound on $\binom{3i}{i}$. Corollary 4.2 says that

$$\binom{3i}{i} \geq \left(\frac{3i}{i}\right)^i = 3^i.$$

We need the following slightly stronger lower bound which can also be shown by using Lemma 4.1.

Corollary 4.3. *For any integer* $i > 0$,

$$\binom{3i}{i} \geq \frac{c}{\sqrt{i}} \cdot \left(\frac{27}{4}\right)^i$$

holds for $c = \sqrt{3/(64\pi)}$.

Proof. By Lemma 4.1, we get that $(3i)! \geq \sqrt{2\pi 3i} \cdot \left(\frac{3i}{e}\right)^{3i}$, $(2i)! \leq 2\sqrt{2\pi 2i} \cdot \left(\frac{2i}{e}\right)^{2i}$ and $i! \leq 2\sqrt{2\pi i} \cdot \left(\frac{i}{e}\right)^i$. We get that

$$\begin{aligned}
\binom{3i}{i} = \frac{(3i)!}{(2i)! \cdot i!} &\geq \frac{\sqrt{2\pi 3i} \cdot (3i)^{3i}}{e^{3i}} \cdot \frac{e^{2i}}{2\sqrt{2\pi 2i} \cdot (2i)^{2i}} \cdot \frac{e^i}{2\sqrt{2\pi i} \cdot i^i} \\
&= \frac{\sqrt{3}}{4\sqrt{4\pi}} \cdot \frac{1}{\sqrt{i}} \cdot \frac{(3i)^{3i}}{(2i)^{2i} i^i} \\
&= \sqrt{\frac{3}{64\pi}} \cdot \frac{1}{\sqrt{i}} \cdot \left(\frac{(3i)^3}{(2i)^2 \cdot i}\right)^i \\
&= \sqrt{\frac{3}{64\pi}} \cdot \frac{1}{\sqrt{i}} \cdot \left(\frac{27}{4}\right)^i = \frac{c}{\sqrt{i}} \cdot \left(\frac{27}{4}\right)^i
\end{aligned}$$

for $c = \sqrt{\frac{3}{64\pi}}$ which proves the corollary. □

Observe that $27/4 = 6.75 > 3$, so Corollary 4.3 does indeed provide a stronger bound for our term than Corollary 4.2.

The binomial theorem helps computing the sum of binomial coefficients. It can be proved by induction.

Theorem 4.4 (Binomial Theorem). *For all* $x, y \quad \mathsf{R}$ *and all* $n \quad \mathsf{N}_0$, *it holds that*

$$\sum_{k=0}^{n} \binom{n}{k} \cdot x^{n-k} y^k = (x + y)^n.$$

Observe three interesting special cases: $x = 1$ and $y = 1$, $x = 1$ and $y = 1$, $x = 1$ and $y = 1$. We will need the first case, i.e., the statement that

$$\sum_{k=0}^{n} \binom{n}{k} \cdot y^k = (1 + y)^n.$$

holds for all $y \quad \mathsf{R}$ and all $n \quad \mathsf{N}_0$.

4.2 Applications

We analyze two algorithms whose analysis reduces to analyzing random walks on a line graph. The second application is what we are really interested in, a local search based algorithm for 3-SAT. The algorithm was developed and analyzed by Schöning [Sch02] for k-SAT. It is an elegant randomized algorithm that performs surprisingly well. When it was developed, it was the fastest known algorithm for 3-SAT (of its type, i.e., a randomized exact algorithm with a one-sided failure probability). It has a running time of $O(1.334^n \cdot \text{poly}(n))$. The currently fastest algorithm is a randomized algorithm due to Hertli [Her14] with a running time of $O(1.30704^n \cdot \text{poly}(n))$. Schöning's algorithm was derandomized by Moser and Scheder [MS11]. The best known deterministic algorithm for 3-SAT is a derandomized version of an algorithm by Hofmeister, Schöning, Schuler and Watanabe [HSSW07], developed by Makino, Tamaki and Yamamoto [MTY13]. The running time of this algorithm is $O(1.3302^n \cdot \text{poly}(n))$.

To understand the basic analysis principle, we first look at a similar algorithm for 2-SAT which was previously developed by Papadimitriou [Pap91].

The input to the ***k-satisfiability (k-SAT) problem*** is a Boolean formula that is the conjunction of m clauses with at most k literals over n variables $x_1, \dots, x_n$. Recall that a ***literal*** is either x_i or $\overline{x_i}$ and that a ***clause*** is a disjunction of literals. For example,

$$(x_1 \quad \overline{x_3}) \quad (x_2 \quad x_3) \quad (x_1 \quad x_2) \quad (\overline{x_2} \quad x_3) \quad x_4$$

is a valid input to the 2-SAT problem. The k-SAT problem is to decide whether an assignment $a \quad \{0,1\}^n$ of x exists that satisfies the formula (where x is the vector of all x_i). The 2-SAT problem can be solved in polynomial time. For $k \geq 3$, the k-SAT problem is NP-hard [Kar72].

4.2.1 A local search algorithm for 2-SAT

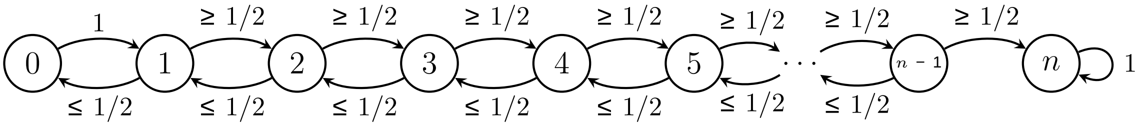
Papadimitriou's 2-SAT algorithm is given a 2-SAT formula ϕ over n variables $x_1, \dots, x_n$ and computes an assignment a. The algorithm starts with an arbitrary assignment and then adjusts this assignment. In every step, it chooses an arbitrary clause C that is not satisfied. In order to satisfy C, the assignment of at least one of the two involved variables needs to be changed. Instead of deciding for one based on the structure of ϕ, the algorithm chooses uniformly at random. Assume that ϕ is satisfiable and that a is an assignment that satisfies all clauses. Then at least one literal in C has the inverse assignment in a compared to a , since C would otherwise be satisfied. The algorithm chooses one of them uniformly at random and changes the assignment of the corresponding variable. With probability at least $1/2$, this increases the number of variables whose assignment agrees with a by one (the Hamming distance of a and a is reduced by one). With the remaining probability, the number of variables with matching assignments decreases by one (the Hamming distance increases). The goal of the algorithm is to agree with one satisfying assignment in all variables. The important step is now to prove that this happens with high probability when the algorithm does sufficiently many steps. The following pseudo code `Rand-2-SAT`$(\phi, t, a\ )$ realizes

Papadimitriou's algorithm. Here, ϕ is the input formula, t is a parameter that controls the number of steps and a is a parameter that allows us to start the algorithm with an arbitrary assignment.

```
Rand-2-SAT($\phi, t, a$ )
    initialize the assignment $a$ with $a$
    for $i = 1$ to $t$ do
        if $a$ does not satisfy $\phi$ then
            choose an arbitrary clause $C$ in $\phi$ that is not satisfied by $a$
            choose a literal in $C$ uniformly at random and change its assignment
    return "YES" if $a$ satisfies $\phi$, else return "NO"
```

The algorithm has one-sided error because it only returns YES if it finds a satisfying assignment. If ϕ is not satisfiable, then it cannot find such an assignment and outputs NO, which is correct in this case. If ϕ is satisfiable, then the algorithm is correct if it finds a satisfying assignment, otherwise, it errs.

To analyze the failure probability of Rand-2-SAT in the case that ϕ is satisfiable, we interpret the execution as a random walk W. The graph is implicitly defined. It consists of a node i for all i $\{0, \ldots, n\}$, and node i represents the state that the current assignment agrees with an (arbitrary but fixed) satisfying assignment a in exactly i variables. If n is reached, then the algorithm correctly outputs YES. If n is not reached, it might still output YES because there could be other satisfying assignments. For formulas with a unique satisfying assignment, the algorithm will err if the random walk does not reach n. We thus analyze the probability that the algorithm reaches n within t steps which is a lower bound on the success probability of Rand-2-SAT.

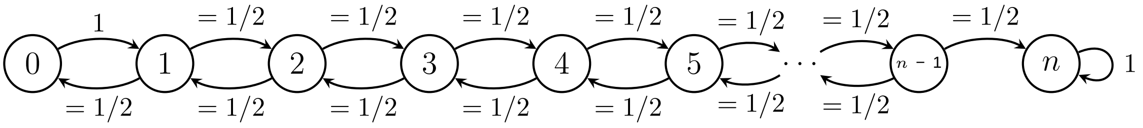


The initial assignment is arbitrary, so the algorithm starts at an arbitrary node i. If the state is 0, then flipping any variable increases the number of agreeing variables by one. Thus, going from 0 to 1 has probability 1. If the state is n, then the algorithm has found a , outputs YES and ends. Thus, once n is reached, it is never left. For all other i $\{1, \ldots, n-1\}$, the algorithm chooses a clause C that is not satisfied. At least one of the two involved variables is in disagreement with a . It is also possible that both variables are assigned differently than in a . Thus, the probability to go to $i+1$ is either 1/2 or 1 (i.e., at least 1/2), and the probability to go to $i-1$ is either 1/2 or 0 (i.e., at most 1/2).

Lemma 4.5. *Let ϕ be a 2-SAT formula that is satisfiable. Let a be an arbitrary assignment. Let Z be the number of steps until Rand-2-SAT(ϕ, t, a) (run with sufficiently large t) finds a satisfying assignment. Then $\mathbf{E}[Z] \leq n^2$.*

Proof. Let a be an arbitrary but fixed satisfying assignment for ϕ. We analyze the expected number of steps until Rand-2-SAT(ϕ, t, a) finds a . This number is an upper bound for the expected number of steps until the algorithm finds any satisfying assignment.

To facilitate the analysis, we look at a simplified version W of the random walk W discussed above: We assume that the probability of increasing the number of agreeing variables by one is exactly $1/2$ if a and a agree on at least one and at most $n - 1$ variables. Likewise, we assume that decreasing the number of agreeing variables by one is exactly $1/2$ if a and a agree on at most $n - 1$ variables.



Let Z and Z be random variables that describe the number of steps Rand-2-SAT(ϕ, t, a) needs to reach state n. The modified walk W is a pessimistic variant of W in the sense that we have $\mathbf{E}[Z] \leq \mathbf{E}[Z\ ]$ and so it is sufficient to show that $\mathbf{E}[Z\ ] \leq n^2$. In the sequel, let X_i denote the state that W is in after i steps, i.e., X_i denotes the number of agreeing variables in a and $a^\star$ after i iterations of Rand-2-SAT(ϕ,t,a). For all $j = 0, \ldots, n$ we define $h_j := \mathbf{E}[Z \mid X_0 = j]$ as the expected number of steps that W needs to reach n from if started at j. Now, the crucial insight is that because the transition probabilities of all states j in W only depend on j and not on how j was reached[1] we have for all $i = 1, \ldots, t$ and all $j = 0, \ldots, n$:

$$\mathbf{E}[Z \mid X_i = j] = \mathbf{E}[Z + i \mid X_0 = j] = \mathbf{E}[Z \mid X_0 = j] + i.$$

where the second equality holds by linearity of expectation.

Proving $\max_{j=0,\ldots,n} h_j \leq n^2$ proves the lemma. By definition, we have $h_n = 0$. If Z starts at state 0, then it will move to state 1 in the first step with probability 1 and therefore by the above insight

$$\begin{aligned} h_0 &= \mathbf{E}[Z \mid X_0 = 0] \\ &= \mathbf{E}[Z \mid X_0 = 0 \quad X_1 = 1] \\ &= \mathbf{E}[Z \mid X_1 = 1] \\ &= \mathbf{E}[Z \mid X_0 = 1] + 1 \\ &= h_1 + 1. \end{aligned}$$

If W starts in state $j \quad \{1, \ldots, n - 1\}$, we condition on the first step of W : With probability $1/2$, the walk W moves into state $j+1$ in the first step; also with probability $1/2$, it moves into state $j - 1$.

$$\mathbf{E}[Z \mid X_0 = j] = \frac{1}{2}\mathbf{E}[Z \mid X_0 = j \quad X_1 = j - 1] + \frac{1}{2}\mathbf{E}[Z \mid X_0 = j \quad X_1 = j + 1]$$

[1] This is not true in W where the actual transition probabilities depend on the current assignment.

$$
\begin{aligned}
&= \frac{1}{2}\mathbf{E}[Z \mid X_1 = j-1] + \frac{1}{2}\mathbf{E}[Z \mid X_1 = j+1] \\
&= \frac{1}{2}(\mathbf{E}[Z \mid X_0 = j-1] + 1) + \frac{1}{2}(\mathbf{E}[Z \mid X_0 = j+1] + 1) \\
&= \frac{h_{j-1}}{2} + \frac{h_{j+1}}{2} + 1
\end{aligned}
$$

By induction, we show that for all $j \quad \{0, \ldots, n-1\}$ it holds that

$$h_j = h_{j+1} + 2j + 1. \tag{4.1}$$

For $j = 0$, it is $h_0 = h_1 + 1 = h_1 + 2 \cdot 0 + 1$. For $j \quad \{1, \ldots, n-1\}$, we compute

$$
h_j = 1 + \frac{1}{2} \cdot h_{j-1} + \frac{1}{2} \cdot h_{j+1} \qquad
\begin{aligned}
h_{j+1} &= 2h_j - h_{j-1} - 2 \\
&\overset{I.H.}{=} 2h_j - (h_j + 2(j-1) + 1) - 2 \\
&= h_j - 2j - 1
\end{aligned}
$$

Hence, we have $h_j = h_{j+1} + 2j + 1$ and this proves (4.1). This implies that

$$
\begin{aligned}
h_j = h_{j+1} + 2j + 1 &= h_{j+2} + [2(j+1) + 1] + [2j + 1] \\
&= \ldots = h_n + [2(n-1) + 1] + \ldots + [2j + 1] \\
&= \sum_{k=j}^{n-1} (2k+1) \leq \sum_{k=0}^{n-1} (2k+1) = n + n(n-1) = n^2.
\end{aligned}
$$

Thus $h_j \leq n^2$ for all $j \quad \{0, \ldots, n-1\}$, which proves the lemma. □

Lemma 4.5 says that the local search needs n^2 steps in expectation to find a satisfying assignment. We can ensure that a satisfying assignment is found with probability $1 - \delta$ by running the algorithm for a sufficiently large number of steps.

Theorem 4.6. *If* `Rand-2-SAT(`ϕ, t, a`)` *is called with a satisfiable 2-SAT formula ϕ, a number $t \geq 2n^2 \ \log_2(1/\delta)$ and an arbitrary starting assignment a, then it finds a satisfying assignment with probability at least $1 - \delta$.*

Proof. We split the t steps into $t = \log_2(1/\delta)$ phases of length $2n^2$. Let A_i be the event that phase i fails, i.e., no satisfying assignment is found. The algorithm fails if all events A_i occur for $i \quad \{1, \ldots, t\}$. The probability that this happens is

$$\mathbf{Pr}(A_1 \cap A_2 \cap \ldots \cap A_t) = \prod_{i=1}^{t} \mathbf{Pr}(A_i \mid A_{i-1} \cap \ldots \cap A_1),$$

which follows from the definition of conditional probability (compare our derivation on page 13 in Section 1.2). The condition $A_{i-1} \cap \ldots \cap A_1$ only means that phase i starts in an assignment $a = a$. Observe that phase i starting at assignment a is identical to calling `Rand-2-SAT(`$\phi, 2n^2, a$`)`. Lemma 4.5 holds for ***any*** starting assignment, thus the expected number of steps until a satisfying assignment is found in phase i is n^2. By Markov's inequality, the probability that no satisfying assignment is found within the $2n^2$ steps that the phase is long is bounded by $1/2$. We get that

$$\mathbf{Pr}(A_1 \cap A_2 \cap \ldots \cap A_t) \leq \left(\frac{1}{2}\right)^t = \frac{1}{2^{\ \log_2(1/\delta)}} \leq \delta.$$

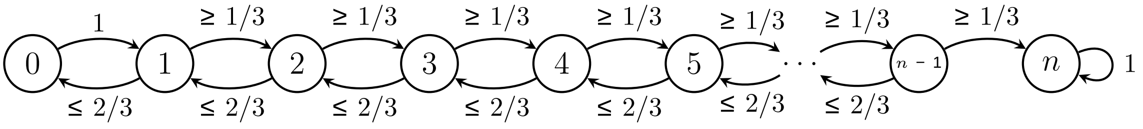
Thus, the probability that the algorithm fails after $2t\,n^2$ steps is at most δ. □

4.2.2 Local search algorithms for 3-SAT

Schöning's algorithm for k-SAT is ***not*** identical to the 2-SAT algorithm we saw above. It has an additional twist. Before discussing it (for $k = 3$), we analyze the straightforward generalization of `Rand-2-SAT` to 3-SAT. The resulting algorithm `Rand-3-SAT(`ϕ, t, a `)` again has the parameters ϕ for the input formula, t for the number of steps and a for the initial assignment.

```
Rand-3-SAT($\phi, t, a$ )
    initialize the assignment $a$ with $a$
    for $i = 1$ to $t$ do
        if $a$ does not satisfy all clauses then
            choose an arbitrary clause $C$ that is not satisfied by $a$
            choose a literal in $C$ uniformly at random and change its assignment
    return "YES" if $a$ satisfies all clauses, else return "NO"
```

The analysis of `Rand-3-SAT` for a satisfying instance ϕ differs from the analysis of the algorithm `Rand-2-SAT` in one important aspect. The clauses in ϕ now have up to three literals. For a non-satisfied clause C and some fixed satisfying assignment a , it is now possible that two of the involved variables agree with a and the clause is still not satisfied because a and a disagree on the third involved variable. By choosing one of the involved variables uniformly at random, the probability to increase the number of agreeing variables by one can be 1/3, 2/3 or 1. Thus we are only guaranteed that the probability is at least 1/3 and that the probability to decrease the number of agreeing variables is at most 2/3. As for 2-SAT, we model the process by a random walk on a line graph where vertex i represents the state that a and a agree in i variables. As before, being in vertex 0 means that the next change increases the number of agreeing variables with probability 1, and being in vertex n means that the assignment is no longer changed. For all j $\{1, \dots, n - 1\}$, we just argued that we go to $j - 1$ with probability at most 2/3 and to $j + 1$ with probability at least 1/3.



Intuitively, we expect that the random walk is likely to go into the wrong direction. However, since 3-SAT is NP-hard, we do not expect a polynomial algorithm. Even waiting an exponential number of steps for a satisfying assignment can lead to an algorithm that is good in our eyes. We apply the analytic technique we used in Lemma 4.5 to `Rand-3-SAT`.

Lemma 4.7. *Let ϕ be a 3-SAT formula that is satisfiable and let a be an arbitrary satisfying assignment. Let a be an assignment that agrees with a in i variables. Let Z be a random variable describing the number of steps until* `Rand-3-SAT(`ϕ, t, a `)` *(run* *with sufficiently large t) finds a satisfying assignment. Then* $\mathbf{E}[Z] \leq 2^{n+2} - 2^{i+2} - 3(n-i)$.

Proof. Observe that the lemma assumes that the random walk is in vertex i when we start. As before, we analyze the pessimistic walk where the probability to move from j to $j-1$ is set to exactly $2/3$ and the probability to move from j to $j+1$ is set to exactly $1/3$, for all $j \in \{1, \ldots, n-1\}$. Let X_ℓ be a random variable describing the state of the pessimistic walk at time ℓ and let Z be a random variable describing the number of steps that the pessimistic walk takes to reach state n. We define $h_j := \mathbf{E}[Z \mid X_0 = j]$ as before.

As in the proof of Lemma 4.5 we get that $h_0 = h_1 + 1$ and $h_n = 0$. For $j \in \{1, \ldots, n-1\}$ we have $\mathbf{Pr}(X_1 = j+1 | X_0 = j) = 1/3$ and $\mathbf{Pr}(X_1 = j-1 | X_0 = j) = 2/3$. Again analogously to Lemma 4.5, we thus obtain the following recursion for $j \in \{1, \ldots, n-1\}$ by conditioning on the first step of the walk:

$$\begin{aligned}
h_j = \mathbf{E}[Z \mid X_0 = j] &= \frac{2}{3}\mathbf{E}[Z \mid X_0 = j \wedge X_1 = j-1] + \frac{1}{3}\mathbf{E}[Z \mid X_0 = j \wedge X_1 = j+1] \\
&= \frac{2}{3}\mathbf{E}[Z \mid X_1 = j-1] + \frac{1}{3}\mathbf{E}[Z \mid X_1 = j+1] \\
&= \frac{2}{3}(\mathbf{E}[Z \mid X_0 = j-1] + 1) + \frac{1}{3}(\mathbf{E}[Z \mid X_0 = j+1] + 1) \\
&= \frac{2}{3}h_{j-1} + \frac{1}{3}h_{j+1} + 1.
\end{aligned}$$

This implies that $h_j \leq 2^{n+2} - 2^{j+2} - 3(n-j)$ for all $j \in \{0, \ldots, n\}$ (exercise). The lemma assumes that the algorithm starts in i, so the expected number of steps needed until n is reached is bounded by $2^{n+2} - 2^{i+2} - 3(n-i)$. □

An arbitrary starting assignment can disagree in all n variables. Then the bound from Lemma 4.7 is $2^{n+2} - 4 - 3n$. This is worse than the immediate upper bound of 2^n for (deterministically) iterating through all possible assignments to determine whether one of them is satisfying. We are interested in an algorithm with exponential running time, but the running time should be d^n for a $d < 2$.

Schöning's algorithm has two twists. First, the starting assignment is not chosen arbitrarily but uniformly at random from all possible starting assignments. Such an assignment will agree with a fixed a in $n/2$ variables in expectation. There is also a chance that it agrees in more variables. Second, Schöning's algorithm restarts the local search after a certain number of steps. This is similar to the implicitly defined phases that we used in Theorem 4.6, however, there is one important difference. After every restart, the algorithm again starts with an assignment chosen uniformly at random instead of the final assignment of the phase before. The following pseudo code `Improved-3-SAT(`ϕ, s`)` realizes Schöning's algorithm. It uses `Rand-3-SAT` as a subroutine. The parameter s controls the number of restarts. As in all of this section, n refers to the number of variables in ϕ in the pseudo code.

```
Improved-3-SAT($\phi, s$)
for $i = 1$ to $s$ do
    choose an assignment $a$ uniformly at random
    if Rand-3-SAT($\phi, 3n, a$) = "YES" then return "YES";
return "NO";
```

Let us first investigate whether starting with at least $n/2$ agreeing variables is a significant improvement over a worst case start (i.e., an assignment that disagrees in all variables). For $i = n/2$, the bound from Lemma 4.7 is $2^{n+2} - 2^{n/2+2} - 3(n/2)$ which is still $\Omega(2^{n+2})$. However, Lemma 4.7 only analyzes the *expected* number of steps that the random walk needs. In the previous sections, we often used statements on the expectation of a random variable and then argued that the random variable is close to this expected value because that was a good event for us. Here, we can also state that the number of steps will be close to $2^{n+2} - 2^{n/2+2} - 3(n/2)$ with constant probability. But we are now interested in the inverse question: Is there a good probability that the number of steps deviates significantly from its expected value, in particular, that it is significantly smaller?

We need a better understanding of the probability distribution, so we continue the analysis of the random walk. We want to find a lower bound for the probability that the walk reaches vertex n if it starts in vertex $n - i$. The probability space underlying this random process is rather complex. In particular, the walks are of different lengths. Assume we encode a walk starting in $n - i$ by a sequence of L and R that represent whether the algorithm made a step to the left or to the right. Then this sequence has at most t letters since the number of iterations is t. However, there are also walks that have fewer steps since the algorithm stops moving when it reaches n. The latter also means that not all strings of length t are feasible walks. For example, for $t > i$ the string $R^i(LR)^{(t-i)/2}$ does not represent a walk that can actually happen since n is reached after i steps and then the algorithm stops making changes. Furthermore, a string that starts with L^{n-i+1} does not represent a walk because the algorithm would go to the right after reaching 0 after $n - i$ steps to the left.

Instead of analyzing this rather complex structure, we analyze a simplified random walk. We extend the line graph sufficiently to the left and right such that t consecutive steps in the same direction are possible from all vertices 0 to n. Furthermore, we assume that the walk acts in 0 and n as for all other nodes, having a probability of $\geq 1/3$ to go to right and a probability of $\leq 2/3$ to go to the left. Now we define the probability space Ω_i for $i \quad \{0, \ldots, n\}$ as $\{L, R\}^{3i}$. This means that we consider all walks consisting of exactly $3i$ steps. (The number $3i$ will turn out to be a good choice later in our analysis). We define a *success* as the event that the walk ends in vertex n. In particular, if the walk visits n but the walk does not end in n, then this is counted as a failure. Observe that the probability for a success is a lower bound for the probability that we reach n from $n - i$ in the original random walk.

Let q_i be the probability of a success. In order to achieve a success, the walk has to end in vertex n after doing $3i$ steps. The walk reaches n by making k steps to the left and $i + k$ steps to the right, and in order to reach n in exactly $3i$ steps, k has to be i. Thus, every walk represented by a string with i L's and $2i$ R's leads to a success. There are $\binom{3i}{i}$ such strings in $\{L,R\}^{3i}$. Each of the corresponding walks has a probability of $(2/3)^i \cdot (1/3)^{2i}$. Thus, the probability for a success is

$$q_i = \binom{3i}{i} \cdot \left(\frac{2}{3}\right)^i \cdot \left(\frac{1}{3}\right)^{2i} = \binom{3i}{i} \cdot \frac{2^i}{3^{3i}} = \binom{3i}{i} \cdot \frac{2^i}{(3^3)^i} = \frac{(3i)!}{(2i)! \cdot i!} \cdot \frac{2^i}{27^i}. \tag{4.2}$$

Now we use Stirling's formula to observe that

$$q_i = \frac{(3i)!}{(2i)! \cdot i!} \cdot \frac{2^i}{27^i} \overset{\text{Cor. (4.3)}}{\geq} \frac{c}{\sqrt{i}} \cdot \left(\frac{27}{4}\right)^i \left(\frac{2}{27}\right)^i = \frac{c}{\sqrt{i}} \cdot \left(\frac{27 \cdot 2}{4 \cdot 27}\right)^i = \frac{c}{\sqrt{i}} \cdot \frac{1}{2^i}.$$

In particular, $q_i \geq \frac{c}{\sqrt{n/2}} \cdot \frac{1}{2^{n/2}}$ for $i \leq n/2$. With this, we can show that `Improved-3-SAT` for an $s \in O(1.42^n)$ finds a satisfying assignment with constant probability if there is one (exercise). This algorithm has a running time of $O(n \cdot 1.42^n)$, an improvement over the 2^n bound for the brute force solution.

However, this result only used knowledge about reaching n from a vertex that is at least $n/2$. We can do better by using the full information that we gained about the q_i. Let q be the probability that one iteration of choosing an arbitrary assignment and doing $3n$ random walk steps successfully reaches vertex n, i.e., q is the success probability of one iteration of `Improved-3-SAT`. For any $i \in \{0,\ldots,n\}$, the probability to start in vertex $n-i$ is $p_i = \binom{n}{n-i}/2^n = \binom{n}{i}/2^n$. Let A_i be the event that the iteration starts in vertex $n-i$ and reaches n. The event A_i has probability $p_i \cdot q_i$. All events $A_i, i \in \{0,\ldots,n\}$ are disjoint, so $q = \sum_{i=0}^{n} \mathbf{Pr}(A_i) = \sum_{i=0}^{n} p_i \cdot q_i$. To compute a lower bound on this term, we use the binomial theorem and get that

$$\begin{aligned} q &= \sum_{i=0}^{n} p_i \cdot q_i \\ &\geq \sum_{i=0}^{n} \frac{\binom{n}{i}}{2^n} \cdot \frac{c}{\sqrt{i}} \cdot \frac{1}{2^i} \geq \sum_{i=0}^{n} \frac{\binom{n}{i}}{2^n} \cdot \frac{c}{\sqrt{n}} \cdot \frac{1}{2^i} \\ &= \frac{c}{\sqrt{n}2^n} \sum_{i=0}^{n} \binom{n}{i} \cdot \left(\frac{1}{2}\right)^i \cdot 1^{n-i} \\ &= \frac{c}{\sqrt{n}2^n} \left(\frac{3}{2}\right)^n = \frac{c}{\sqrt{n}} \cdot \left(\frac{3}{4}\right)^n. \end{aligned} \tag{4.3}$$

We have now gathered enough information to prove the main theorem of this section.

Theorem 4.8. *Let ϕ be a satisfiable 3-SAT formula. For $s = \frac{(\ln 1/\delta)\sqrt{64\pi \cdot n}}{\sqrt{3}} \cdot (\frac{4}{3})^n$, the algorithm `Improved-3-SAT`$(\phi, s)$ finds a satisfying assignment for ϕ with probability at least $1-\delta$ and has a running time of $O((\ln 1/\delta) \cdot n^{3/2} \cdot 1.334^n \cdot \chi(n))$ where $\chi(n)$ is the time needed to evaluate a 3-SAT formula in n variables.*

Proof. Recall that $c = \sqrt{3}/\sqrt{64\pi}$. By (4.3), the success probability of one iteration of `Improved-3-SAT`(ϕ, s) is at least $q = \frac{c}{\sqrt{n}} \cdot (3/4)^n$. The probability that all s tries fail is at most $(1-q)^s \leq (1-q)^{(\ln 1/\delta)/q} \leq \delta$. The running time for one iteration is $\Theta(n \cdot \chi(n))$, so the total running time is $O(s \cdot n \cdot \chi(n))$. □

Part II

Probabilistic Analysis

Chapter 5

Introduction

The theory of algorithms has traditionally focused on worst-case analysis. This focus has led to both a deep theory and many beautiful and useful algorithms. There are, however, a number of important problems and algorithms for which worst-case analysis does not provide useful or empirically accurate results. One prominent example is the simplex method for linear programming whose worst-case running time is exponential while in fact it runs in near-linear time on almost all inputs of interest. Another example is the knapsack problem. While this problem is NP-hard, it is a very easy optimization problem in practice and even very large instances with millions of items can be solved efficiently. The reason for this discrepancy between worst-case analysis and empirical observations is that for many algorithms worst-case instances have an artificial structure and hardly ever occur in practical applications.

In recent years a paradigm shift towards a more realistic and robust algorithmic theory has been initiated. The development of a more realistic theory hinges on finding models that measure the performance of an algorithm not only by its worst-case behavior but rather by its behavior on typical inputs. However, for an optimization problem at hand it is usually impossible to rigorously define the notion of “typical input” because what such an input looks like depends on the concrete application and on other indistinct parameters. In order to cope with this problem, Spielman and Teng introduced the model of *smoothed analysis* [ST04]. This model can be considered as a less pessimistic variant of worst-case analysis in which the adversarial input is subject to a small amount of random noise.

Let us illustrate smoothed analysis by means of the knapsack problem. Worst-case analysis can be viewed as a game between an algorithm designer and an adversary. First the algorithm designer chooses an algorithm, then the adversary chooses an input on which the algorithm performs as poorly as possible. For the knapsack problem, the adversary chooses profits, weights, and the capacity of the knapsack. In order to limit the power of the adversary to construct artificial instances that do not resemble typical inputs, we add some randomness to his decisions in the following way. Instead of being able to determine the numbers in the input exactly, he can only choose for each number an interval of length $1/\phi$ from which it is chosen uniformly at random.

Here $\phi \geq 1$ is some parameter that determines the power of the adversary. The larger it is, the more precisely he can specify the input. In the limit for $\phi \to \infty$ the adversary is as powerful as in a worst-case analysis.

We will see that there are heuristics for the knapsack problem whose expected running time is polynomially bounded in the number of items and the parameter ϕ. This shows that instances on which the algorithm requires exponential running time are fragile with respect to random perturbations and even a small amount of randomness suffices to rule out such instances with high probability. In practice, random noise can stem from measurement errors, numerical imprecision, or rounding errors. It can also model arbitrary influences that we cannot quantify exactly, but for which there is also no reason to believe that they are adversarial.

After its invention in 2001, smoothed analysis has attracted a great deal of attention and it has been applied in a variety of different contexts, e.g., in multi-objective optimization, local search, clustering, and online algorithms. By now smoothed analysis is widely accepted as a realistic alternative to worst-case analysis. Part II of this lecture discusses various results from the area of smoothed analysis.

In Sections 1 and 2 of Part I, we discussed some facts about discrete probability theory. In Part II, we also need continuous probability spaces. The remainder of this chapter gives a short introduction and reviews some facts from discrete probability theory that extend to continuous probability spaces.

Readers who are familiar with probability theory can skip the sections 5.1 to 5.4. In Section 5.5 we introduce the notion of *ϕ-perturbed numbers* and discuss the probabilistic input model that we will employ throughout this lecture.

5.1 Continuous Probability Spaces

In Part II we will often deal with continuous random experiments with uncountable sample spaces. Recall the definition of discrete probability spaces in Definition 1.2. We defined probability measures to map from 2^{Ω} to $[0,1]$, but then observed that it suffices to define discrete probability measures as mappings from Ω to $[0,1]$ and extend it to arbitrary events by

$$\mathbf{Pr}(A) = \sum_{a \quad A} \mathbf{Pr}(\{a\}).$$

A simple example for a continuous probability space would be to choose a real number from the interval $[0,1]$ uniformly at random by which we mean intuitively that each outcome from $\Omega = [0,1]$ should have the same probability. We cannot model this by a probability measure $\mathbf{Pr} : \Omega \to [0,1]$ as in the case of discrete probability spaces. Indeed if one wants to assign the same probability to each outcome in $[0,1]$, then one has no choice but to set $\mathbf{Pr}(x) = 0$ for each $x \quad \Omega$. Clearly it makes no sense to extend this function to a function $\mathbf{Pr} : 2^{\Omega} \to \mathsf{R}$ in the same way as for discrete probability spaces (in particular, because the sum $\sum_{x \quad X} \mathbf{Pr}(x)$ is not even defined for uncountable X $[0,1]$). Instead if Ω is uncountable, one has to define the function $\mathbf{Pr} : 2^{\Omega} \to \mathsf{R}$ directly without recourse to a function $\mathbf{Pr} : \Omega \to [0,1]$.

Definition 5.1. *A* (continuous) probability space *is a tuple* $(\Omega, F, \mathbf{Pr})$ *with the following components.*

1. *The* sample space Ω *is an arbitrary set. It represents the set of all possible outcomes of the random experiment modeled by the probability space.*
2. *The* set of events $F \quad 2^{\Omega}$ *is a* σ-algebra *on* Ω*. This means that* F *is a family of subsets of* Ω *with the following properties.*
   (a) $\Omega \quad F$
   (b) F *is closed under complementation:* $X \quad F = \quad \Omega \setminus X \quad F$*.*
   (c) F *is closed under countable unions:* $X_1, X_2, X_3, \ldots \quad F = \quad \bigcup_{i=1}^{\infty} X_i \quad F$*.*

   The set F *represents the events for which a probability is defined.*
3. *The* probability measure $\mathbf{Pr} : F \to [0,1]$ *is a* σ*-additive function that satisfies* $\mathbf{Pr}(\Omega) = 1$*. This means that for pairwise disjoint events* $X_1, X_2, X_3, \ldots \quad F$ *one has* $\mathbf{Pr}(\bigcup_{i=1}^{\infty} X_i) = \sum_{i=1}^{\infty} \mathbf{Pr}(X_i)$*.*

We will not need to deal very deeply with continuous probability theory and we presented the definition above mainly for the sake of completeness. Still let us briefly discuss it. First of all, one might ask what the σ-algebra F is needed for. In light of Definition 1.2 it seems reasonable to always choose $F = 2^{\Omega}$ because then a probability is defined for every subset of the sample space. The problem with this is that under the assumption of the continuum hypothesis there is no σ-additive probability measure $\mathbf{Pr}$ on 2^{R} with $\mathbf{Pr}(\{x\}) = 0$ for each $x \quad \mathsf{R}$ (Banach-Kurnatowski theorem). Since we need such measures to model, for example, the random experiment in which a real number from the interval $[0,1]$ is chosen uniformly at random, it makes sense to consider other σ-algebras than 2^{R}.

We will often consider probability spaces with $\Omega = \mathsf{R}$. Then F will always be chosen as the *Borel* σ*-algebra*, which is defined as the smallest σ-algebra containing all intervals (a,b) with $a, b \quad \mathsf{R}$. All subsets of R that occur in this lecture (and in most other contexts) belong to the Borel algebra. This includes, in particular, all closed intervals $[a,b]$, all half-closed intervals $(a,b]$ and $[a,b)$, and the union of countably many such intervals. To define a probability measure $\mathbf{Pr} : F \to [0,1]$, it suffices in this case to define $\mathbf{Pr}$ only for all open intervals (a,b). Then the value $\mathbf{Pr}(X)$ is uniquely determined for every $X \quad F$ because $\mathbf{Pr}$ is σ-additive. Similarly it suffices to define $\mathbf{Pr}$ only for all closed intervals $[a,b]$.

Let us consider as an example the random experiment that a real number from the interval $[0,1]$ is chosen uniformly at random. This can be modeled by the probability space $(\Omega, F, \mathbf{Pr})$ where $\Omega = \mathsf{R}$, F is the Borel σ-algebra, and $\mathbf{Pr}([a,b]) = b' - a'$ for $[a', b'] := [0,1] \cap [a,b]$ with $b' \geq a'$. Then, in particular, $\mathbf{Pr}(\{x\}) = 0$ and $\mathbf{Pr}([0,x]) = x$ for every $x \quad [0,1]$.

Recall the Union Bound from Lemma 1.5. It extends to continuous probability spaces.

Lemma 5.2. *Let* $X_1, \ldots, X_n$ *be events in a continuous probability space* $(\Omega, F, \mathbf{Pr})$*. Then*

$$\mathbf{Pr}(X_1 \quad \cdots \quad X_n) \leq \sum_{i=1}^{n} \mathbf{Pr}(X_i).$$

Another elementary property of probabilities is the *law of total probability*. We did not use it in Part I, but will need it in Part II. It holds in both discrete and continuous probability spaces.

Lemma 5.3. *Let A be an event in some probability space $(\Omega, F, \mathbf{Pr})$ and let $B_1, \dots, B_n$ be events that form a partition of the sample space Ω, that is, $B_i \cap B_j = \quad$ for $i = j$ and $B_1 \quad \dots \quad B_n = \Omega$. Then*

$$\mathbf{Pr}\,(A) = \sum_{i=1}^{n} \mathbf{Pr}\,(A \cap B_i)\,.$$

5.2 Random Variables

As for discrete random experiments, we are often not interested in the actual outcome of the experiment, but only in a numerical parameter. Consider the example that we study the performance of some algorithm for the knapsack problem on random inputs with n items in which every profit, every weight, and the capacity are chosen at random. Then the underlying probability space is defined on the set $\Omega = \mathsf{R}^{2n+1}$ because $2n + 1$ numbers are chosen at random. However, we do not care about the actual outcome of this random experiment but we are only interested in the running time of the algorithm. For this, we introduced the concept of random variables in Definition 2.1 and now extend it to continuous probability spaces.

Definition 5.4. *Let $(\Omega, F, \mathbf{Pr})$ be a probability space. A (real-valued) continuous random variable $X : \Omega \to \mathsf{R}$ is a function from the set of elementary events to the real numbers that satisfies*

$$X^{-1}\big([a,b]\big) := \Big\{z \quad \Omega \mid X(z) \quad [a,b]\Big\} \quad F \tag{5.1}$$

for every interval $[a,b] \quad \mathsf{R}$.

In the example discussed before the definition, we can choose as random variable the function X that maps every outcome from $\Omega = \mathsf{R}^{2n+1}$ to the number of steps the algorithm requires on that outcome. Then the probability that the algorithm requires at most T steps on a randomly chosen input can be written as $\mathbf{Pr}(X^{-1}([0,T]))$ where $\mathbf{Pr}$ denotes the probability measure of the underlying probability space on $\Omega = \mathsf{R}^{2n+1}$. Observe that condition (5.1) in Definition 5.4 is only necessary to ensure that for any interval $[a,b]$ the probability of X taking a value in $[a,b]$ is defined by the underlying probability space.

As before, we will denote the probability that a random variable X takes some value $x \quad \mathsf{R}$ by $\mathbf{Pr}\,(X = x)$. Furthermore, we will denote the probability that it takes a value in some interval $[a,b]$ by $\mathbf{Pr}\,(X \quad [a,b])$. Formally

$$\mathbf{Pr}\,(X = x) := P(X^{-1}(\{x\})) \qquad \text{and} \qquad \mathbf{Pr}\,(X \quad [a,b]) := P(X^{-1}([a,b])).$$

It is very convenient to describe the behavior of a continuous random variable by a *cumulative distribution function (CDF)* or a *probability density function (PDF)*.

Definition 5.5. *The* cumulative distribution function (CDF) *of a continuous random variable X is the function $F_X : \mathsf{R} \to \mathsf{R}_{\geq 0}$ given by $F_X(t) = \mathbf{Pr}(X \leq t)$ for all $t \quad \mathsf{R}$. A function $f_X : \mathsf{R} \to \mathsf{R}_{\geq 0}$ is called* probability density function (PDF) *of X if*

$$\mathbf{Pr}(X \quad [a, b]) = \int_a^b f_X(t)\, dt$$

for every interval $[a, b] \quad \mathsf{R}$.

Usually we do not mention the underlying probability space of a random variable explicitly, but we are only interested in the distribution or density of the random variable. This is, in particular, true in the following examples.

- Discrete random variables are a special case of continuous random variables. We described the behavior of a discrete random variable X by simply specifying the probability that $X = t$ for all occurring values $t \quad \mathsf{R}$. This can be interpreted as describing X by its cumulative distribution function or by its *probability mass function*.

  Reconsider the example that we have a coin that comes up heads with probability $p \quad (0, 1]$ and let us assume that we toss this coin until it comes up heads for the first time. Let X denote the total number of coin tosses in this experiment. Then X is a geometrically distributed random variable with parameter p and $\mathbf{Pr}(X = t) = p(1 - p)^{t-1}$ for $t \quad \mathsf{N}$. One can now argue that the distribution F_X of X satisfies $F_X(t) = \mathbf{Pr}(X \leq t) = 1 - (1 - p)^t$ because X is larger than t if and only if the first t tosses all come up tails, which happens with probability $(1 - p)^t$.

- Let X be uniformly distributed on $[a, b]$ for some $a < b$. Then the distribution F_X of X is defined by

  $$F_X(t) = \mathbf{Pr}(X \leq t) = \begin{cases} 0 & \text{if } t < a, \\ \frac{t-a}{b-a} & \text{if } a \leq t \leq b, \\ 1 & \text{if } t > b, \end{cases}$$

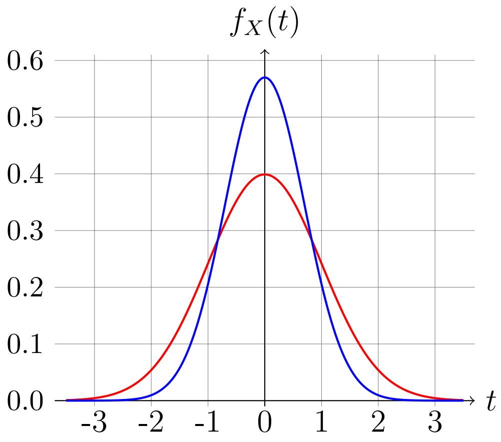
  and a density f_X of X is defined by

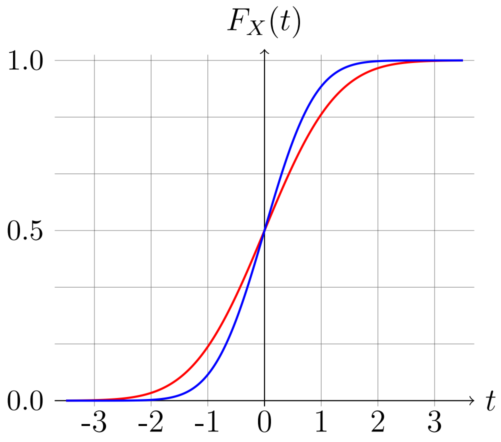
  $$f_X(t) = \begin{cases} 0 & \text{if } t < a \text{ or } t > b, \\ \frac{1}{b-a} & \text{if } a \leq t \leq b. \end{cases}$$

- A distribution that occurs frequently is the *normal distribution* (also known as *Gaussian distribution*). It is used, for example, to model measurement errors in physical experiments. We say that a random variable X is *normally distributed with mean μ and standard deviation σ* if it can be described by the density function f_X with

  $$f_X(t) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(t-\mu)^2}{2\sigma^2}}.$$

  There does not exist a formula for the distribution of a normal distribution in terms of elementary functions but numerical approximations are known. The following figures show the densities and distributions of two Gaussian random variables with mean 0 and standard deviation 0.7 (blue) and 1 (red).





The following easy property of continuous random variables with bounded density will be used extensively throughout this lecture.

Lemma 5.6. *Let X be a continuous random variable with density function $f_X : \mathsf{R} \to \mathsf{R}_{\geq 0}$ with $f_X(t) \leq \phi$ for all t R. Then for every a R and every $\varepsilon \geq 0$, $\mathbf{Pr}(X \quad [a, a+\varepsilon]) \leq \varepsilon\phi$.*

Proof. According to Definition 5.5,

$$\mathbf{Pr}(X \quad [a, a+\varepsilon]) = \int_a^{a+\varepsilon} f_X(t)\,dt \leq \int_a^{a+\varepsilon} \phi\,dt = \varepsilon\phi. \qquad \square$$

5.3 Expected Values

As for discrete random variables, we are often interested in the expected value of continuous random variables. Similar to the frequentist interpretation of probability, the expected value can be interpreted as the average value of the random variable if the random experiment is repeated independently a large number of times. The following definition extends Definition 2.4.

Definition 5.7. *The expected value of a continuous random variable X with density f_X is defined as*

$$\mathbf{E}[X] = \int_{-\infty}^{\infty} t \cdot f_X(t)\,dt$$

if the integral exists.

Let us consider an example. Let X be a Gaussian random variable with mean 0 and standard deviation σ. Then the symmetry of the density function around 0 implies that the expected value of X is 0 if it exists. One only needs to argue that the integral exists. This follows from the following calculation:

$$\mathbf{E}[X] = \int_{-\infty}^{\infty} \frac{t}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{t^2}{2\sigma^2}}\,dt$$

$$
\begin{aligned}
&= \int_{-\infty}^{0} \frac{t}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{t^2}{2\sigma^2}}\,dt + \int_{0}^{\infty} \frac{t}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{t^2}{2\sigma^2}}\,dt \\
&= \frac{1}{\sigma\sqrt{2\pi}} \cdot \left[-\sigma^2 e^{-\frac{t^2}{2\sigma^2}} \right]_{-\infty}^{0} + \frac{1}{\sigma\sqrt{2\pi}} \cdot \left[-\sigma^2 e^{-\frac{t^2}{2\sigma^2}} \right]_{0}^{\infty} \\
&= \frac{-\sigma^2}{\sigma\sqrt{2\pi}} + \frac{\sigma^2}{\sigma\sqrt{2\pi}} = 0.
\end{aligned}
$$

We leave the proof of the following extension of Lemma 2.10 as an exercise for the reader.

Lemma 5.8. *For a continuous random variable X that takes only values in $\mathbb{R}_{\geq 0}$, we can write the expected value as*

$$\mathbf{E}[X] = \int_{t=0}^{\infty} \mathbf{Pr}(X \geq t)\,dt.$$

We have seen multiple examples for the usefulness of the fact that the expected value is linear. *Linearity of expectation* also holds for continuous random variables, as stated in the following lemma which extends 1. and 2. in Theorem 2.7.

Lemma 5.9. *Let X and Y be continuous random variables and let $a, b \quad \mathbb{R}$. Then*

$$\mathbf{E}[X+Y] = \mathbf{E}[X] + \mathbf{E}[Y] \quad \textit{and} \quad \mathbf{E}[aX+b] = a\mathbf{E}[X] + b.$$

As an example, let X be a Gaussian random variable with mean μ and standard deviation σ and let Y be a Gaussian random variable with mean 0 and standard deviation σ. Then it is easy to see that X has the same density as $Y + \mu$. Hence,

$$\mathbf{E}[X] = \mathbf{E}[Y+\mu] = \mathbf{E}[Y] + \mu = 0 + \mu = \mu.$$

Another helpful tool that extends to continuous random variables is Markov's inequality, which we introduced for discrete random variables in Theorem 3.1.

Lemma 5.10. *Let X be a continuous random variable that takes only values in $\mathbb{R}_{\geq 0}$ and let $a \geq 1$. Then*

$$\mathbf{Pr}(X \geq a \cdot \mathbf{E}[X]) \leq \frac{1}{a}.$$

5.4 Conditional Probability and Independence

Given two events A and B in a continuous probability space, the conditional probability of A under the condition that event B occurs is defined analogously to Definition 1.9. We also extend Definitions 1.7 and 2.3 to define independent events and random variables in continuous probability spaces.

Definition 5.11. *Let* $(\Omega, \mathcal{F}, P)$ *be a continuous probability space and let* $A, B \quad \mathcal{F}$ *be two events with* $P(B) > 0$. *The* conditional probability of A given B *is defined as*

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}.$$

The events A *and* B *are called* independent *if* $P(A \cap B) = P(A) \cdot P(B)$. *Otherwise they are called* dependent. *Two random variables* X *and* Y *are called* independent *if the events* $\{X \leq a\}$ *and* $\{Y \leq b\}$ *are independent for any* $a, b \quad \mathbb{R}$. *Otherwise they are called* dependent.

Recall that $P(A \mid B) = P(A)$ if and only if A and B are independent. Furthermore recall that one can also prove that $\mathbf{E}[X \cdot Y] = \mathbf{E}[X] \cdot \mathbf{E}[Y]$ for independent random variables X and Y. Again, the independence is important for this statement to hold. In contrast, Lemma 5.9 (linearity of expectation) is also true for dependent random variables.

Recall the definition of conditional expectation for discrete random variables in Definition 2.11. Combining it with the law of total probability, one can easily see that

$$\mathbf{E}[X] = \mathbf{Pr}(A) \cdot \mathbf{E}[X \mid A] + \mathbf{Pr}(\neg A) \cdot \mathbf{E}[X \mid \neg A],$$

where $\neg A := \Omega \setminus A$.

5.5 Probabilistic Input Model

When we study an optimization problem or an algorithm in the model of smoothed analysis, we first need to specify the input model. In the introduction we mentioned as one example the case where the adversary can choose for the knapsack problem for each number in the input an interval of length $1/\phi$ from which it is chosen uniformly at random. This is a special case of one of the input models that we will employ in this lecture. To explain the general model, consider an arbitrary optimization problem whose instances are described completely or in part by some real numbers. This could, for example, be a problem defined on weighted graphs. Then an instance consists of the graph structure and the numbers that describe the weights. It could also be a problem, like the knapsack problem, where there is no combinatorial structure and every instance is completely described by a set of numbers.

In a worst-case analysis the adversary is allowed to choose both the combinatorial structure (if present) and the numbers exactly. In the model we consider, he can still choose the combinatorial structure arbitrarily (e.g., the graph structure) but there is some randomness in the choice of (some of) the numbers. We say that a number is ϕ-*perturbed* if it is a random variable that can be described by a density that is bounded from above by $\phi \geq 1$. A random variable that is chosen from an interval of length $1/\phi$ is one example of a ϕ-perturbed random variable. A Gaussian random variable with standard deviation σ is another example for $\phi = 1/(\sigma\sqrt{2\pi})$. As discussed in the introduction, the parameter ϕ determines the power of the adversary. The larger it is, the more precisely he can specify the input. In the limit for $\phi \to \infty$ the adversary is as powerful as in a worst-case analysis. Usually we assume that ϕ-perturbed numbers take only values between $[-1, 1]$. This normalization is necessary to ensure that the effect of the randomness cannot be ruled out by scaling all numbers in the input.

We will study the (expected) running time of algorithms on inputs with ϕ-perturbed numbers. It is not always necessary to assume that all numbers in the input are ϕ-perturbed. For the knapsack problem, we will consider, for example, instances in which the adversary can determine the profits and the capacity exactly and only the weights are assumed to be ϕ-perturbed numbers. We will call the part of the input that is not perturbed the ***deterministic part of the input***. Smoothed analysis can be considered as a worst-case analysis in which the adversary can choose the deterministic part of the input exactly and a density function that is bounded from above by ϕ for every ϕ-perturbed number in the input. The ***smoothed running time*** of an algorithm is defined to be the worst expected running time the adversary can achieve by the choice of the deterministic part of the input and the density functions, assuming that each ϕ-perturbed number is drawn independently according to the corresponding density.

The smoothed running time is expressed in terms of the input length (where each ϕ-perturbed number is assumed to contribute only one bit to the input length) and the parameter ϕ. In the next chapter, we will present, for example, an algorithm for the knapsack problem whose smoothed running time is $O(n^3\phi)$, where n denotes the number of items and the weights are ϕ-perturbed numbers. It is clear that the smoothed running time has to increase with ϕ because for very large ϕ, the smoothed running time approaches the exponential worst-case running time. Usually, we aim to prove that the smoothed running time of an algorithm is polynomially bounded in the input size and ϕ. This means that already a small amount of randomness suffices to obtain with high probability instances on which the algorithm runs in polynomial time.

Chapter 6

Knapsack Problem and Multiobjective Optimization

The ***knapsack problem*** is a well-known NP-hard optimization problem. An instance of this problem consists of a set of items, each with a profit and a weight, and a capacity. The goal is to find a subset of the items that maximizes the total profit among all subsets whose total weight does not exceed the capacity. Let $p = (p_1, \dots, p_n)^\mathsf{T}$ $\mathsf{R}^n_{\geq 0}$ and $w = (w_1, \dots, w_n)^\mathsf{T}$ $\mathsf{R}^n_{\geq 0}$ denote the profits and weights, respectively, and let W $\mathsf{R}_{\geq 0}$ denote the capacity. Formally the knapsack problem can be stated as follows:

$$\begin{aligned} \text{maximize} \quad & p^\mathsf{T} x = p_1 x_1 + \cdots + p_n x_n \\ \text{subject to} \quad & w^\mathsf{T} x = w_1 x_1 + \cdots + w_n x_n \leq W, \\ & \text{and } x = (x_1, \dots, x_n)^\mathsf{T} \quad \{0, 1\}^n. \end{aligned}$$

The knapsack problem has been shown to be NP-hard by Karp in 1972 [Kar72]. Since then it has attracted a great deal of attention, both in theory and in practice. Theoreticians are interested in the knapsack problem because of its simple structure; it can be expressed as a binary program with one linear objective function and one linear constraint. On the other hand, knapsack-like problems often occur in practical applications, and hence practitioners have developed numerous heuristics for solving them. These heuristics work very well on random and real-world instances of the knapsack problem and they usually find optimal solutions quickly even for very large instances.

In Section 6.1, we will present the Nemhauser-Ullmann algorithm for the knapsack problem. This algorithm has an exponential worst-case running time but we will prove in Section 6.2 that its smoothed running time is polynomial. We will discuss generalizations of this analysis to multiobjective optimization problems in Section 6.3. In Section 6.4 we will present the concept of core algorithms for the knapsack problem. These are the fastest known algorithms to solve the knapsack problem in practice and the most successful core algorithm uses the Nemhauser-Ullmann algorithm as a subroutine.

6.1 Nemhauser-Ullmann Algorithm

In the following, we assume that an arbitrary instance *I* of the knapsack problem is given. We use the term *solution* to refer to a vector x $\{0,1\}^n$, and we say that a solution is *feasible* if $w^\mathsf{T} x \leq W$. We say that a solution x *contains item* i if $x_i = 1$ and that it *does not contain item* i if $x_i = 0$. One naive approach for solving the knapsack problem is to enumerate all feasible solutions and to select the one with maximum profit. This approach is not efficient as there are typically exponentially many feasible solutions. In order to decrease the number of solutions that have to be considered, we observe that a solution x cannot be optimal if there exists another solution y with larger profit and smaller weight than x. We say that such a solution y *dominates* the solution x and observe that it suffices to consider only solutions that are not dominated by other solutions.

Definition 6.1. *A solution* y dominates *a solution* x *if* $p^\mathsf{T} y \geq p^\mathsf{T} x$ *and* $w^\mathsf{T} y \leq w^\mathsf{T} x$ *and if at least one of these inequalities is strict. A solution* x *is called* Pareto-optimal *if it is not dominated by any other solution* y*. The* Pareto set *is the set of all Pareto-optimal solutions.*

Let us remark that the capacity of the knapsack is irrelevant for Definition 6.1 because we do not require the solutions x and y to be feasible. Nemhauser and Ullmann [NU69] proposed an algorithm for computing the Pareto set of a given knapsack instance. Once the Pareto set is known, the instance can be solved optimally in time linear in the size of this set due to the following observation.

Lemma 6.2. *There always exists an optimal solution that is also Pareto-optimal.*

Proof. Let x be an arbitrary optimal solution for the given instance of the knapsack problem. If x is not Pareto-optimal, then there exists a solution y that dominates x. This means $p^\mathsf{T} y \geq p^\mathsf{T} x$ and $w^\mathsf{T} y \leq w^\mathsf{T} x$ and at least one of these inequalities is strict. The solution y is feasible because $w^\mathsf{T} y \leq w^\mathsf{T} x \leq W$, where the second inequality follows from the feasibility of x. Hence, solution y cannot have a larger profit than solution x because x is an optimal solution. Since $p^\mathsf{T} y \geq p^\mathsf{T} x$, this implies that $p^\mathsf{T} y = p^\mathsf{T} x$. This in turn implies $w^\mathsf{T} y < w^\mathsf{T} x$ because y dominates x. In summary solution y is also an optimal solution with smaller weight than x. This situation is shown in Figure 6.1.

Now either y is a Pareto-optimal solution or by the same reasoning as above there exists another optimal solution z with a smaller weight than y. If we repeat this construction as long as the current solution is not Pareto-optimal, we obtain a sequence of optimal solutions with strictly decreasing weights. As there is only a finite number of solutions, this sequence must be finite as well. The last solution in this sequence is an optimal solution that is also Pareto-optimal. □

We denote the Pareto set by P $\{0,1\}^n$. If this set is known, the solution

$$x^\star = \arg\max_{x \quad P} \{p^\mathsf{T} x \mid w^\mathsf{T} x \leq W\},$$

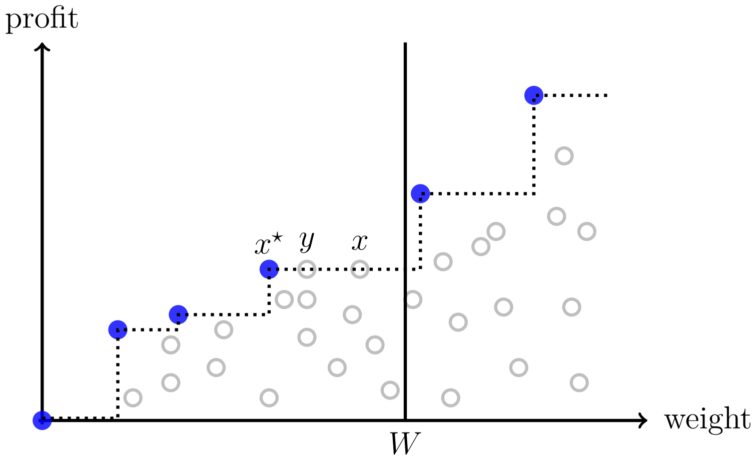


Figure 6.1: In this figure, every solution $z \quad \{0,1\}^n$ is depicted as a circle at coordinates $(w^\mathsf{T} z, p^\mathsf{T} z)$. Only the filled circles correspond to Pareto-optimal solutions and only solutions to the left of W are feasible. The solutions x, y, and $x^\star$ are optimal.

which can be found in time linear in the size of $\mathcal{P}$, is an optimal solution of the given instance of the knapsack problem due to the previous lemma. If the arg max is not uniquely determined but a set of multiple solutions, then it follows from the definition of $\mathcal{P}$ that all these solutions must have the same weight and the same profit. In this case $x^\star$ can be chosen as an arbitrary solution from the arg max-set.

The Nemhauser-Ullmann algorithm for computing the Pareto set $\mathcal{P}$ is based on dynamic programming. For each $i \quad \{0,1,\ldots,n\}$ it computes the Pareto set $\mathcal{P}_i$ of the modified instance $\mathcal{I}_i$ of the knapsack problem that contains only the first i items of the given instance $\mathcal{I}$. Then $\mathcal{P}_n = \mathcal{P}$ is the set we are looking for. Let

$$\mathcal{S}_i = \{x \quad \{0,1\}^n \mid x_{i+1} = \ldots = x_n = 0\}$$

denote the set of solutions that do not contain the items $i+1,\ldots,n$. Formally, solutions of the instance $\mathcal{I}_i$ are binary vectors of length i. We will, however, represent them as binary vectors of length n from $\mathcal{S}_i$. For a solution $x \quad \{0,1\}^n$ and an item $i \quad \{1,\ldots,n\}$ we denote by x^{+i} the solution that is obtained by adding item i to solution x:

$$x_j^{+i} = \begin{cases} x_j & \text{if } j = i, \\ 1 & \text{if } j = i. \end{cases}$$

Furthermore, for a set $\mathcal{S} \quad \{0,1\}^n$ of solutions let

$$\mathcal{S}^{+i} = \{y \quad \{0,1\}^n \mid \quad x \quad \mathcal{S} : y = x^{+i}\}.$$

If for some $i \quad \{1,\ldots,n\}$, the set $\mathcal{P}_{i-1}$ is known, the set $\mathcal{P}_i$ can be computed with the help of the following lemma.

Lemma 6.3. *For every $i \quad \{1,\ldots,n\}$, the set $\mathcal{P}_i$ is a subset of $\mathcal{P}_{i-1} \quad \mathcal{P}_{i-1}^{+i}$.*

Proof. Let $x \quad \mathcal{P}_i$. Based on the value of x_i we distinguish the following two cases.

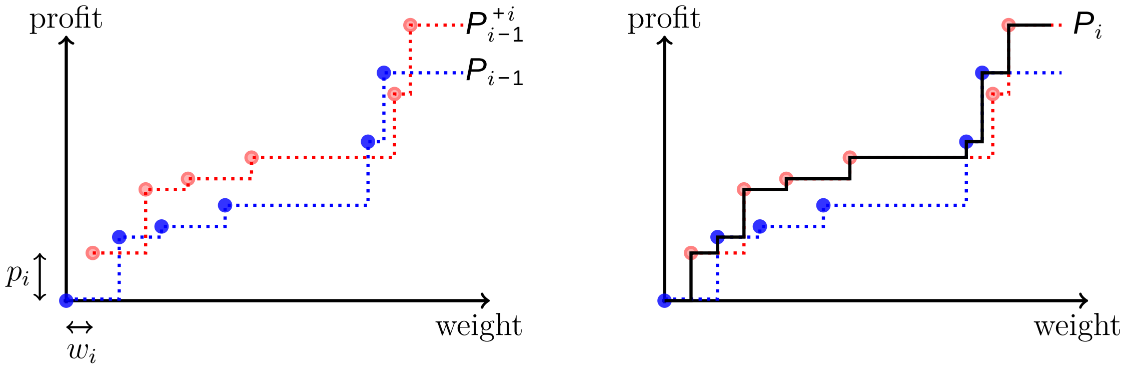


Figure 6.2: Illustration of one iteration of the for loop of the Nemhauser-Ullmann algorithm: The set P^{+i}_{i-1} is a copy of the set P_{i-1} that is shifted by (w_i, p_i). The set P_i is obtained by removing dominated solutions.

First we consider the case $x_i = 0$. We claim that in this case $x \quad P_{i-1}$. Assume for contradiction that x / P_{i-1}. Then there exists a solution $y \quad P_{i-1} \quad S_{i-1} \quad S_i$ that dominates x. Since $y \quad S_i$, solution x cannot be Pareto-optimal among the solutions in S_i. Hence, x / P_i, contradicting the choice of x.

Now we consider the case $x_i = 1$. We claim that in this case $x \quad P^{+i}_{i-1}$. Since $x \quad S_i$ and $x_i = 1$, there exists a solution $y \quad S_{i-1}$ such that $x = y^{+i}$. We need to show that $y \quad P_{i-1}$. Assume for contradiction that there exists a solution $z \quad P_{i-1}$ that dominates y. Then $p^\mathsf{T} z \geq p^\mathsf{T} y$ and $w^\mathsf{T} z \leq w^\mathsf{T} y$ and one of these inequalities is strict. By adding item i to the solutions y and z, we obtain $p^\mathsf{T} z^{+i} \geq p^\mathsf{T} y^{+i}$ and $w^\mathsf{T} z^{+i} \leq w^\mathsf{T} y^{+i}$ and one of these inequalities is strict. Hence, the solution z^{+i} dominates the solution $x = y^{+i}$. Since $z^{+i} \quad S_i$, this implies x / P_i, contradicting the choice of x. □

Due to the previous lemma, the Pareto set P_i can be computed easily if the Pareto set P_{i-1} is already known. For this one only needs to compute the set $P_{i-1} \quad P^{+i}_{i-1}$ and remove solutions from this set that are dominated by other solutions from this set. Using additionally that $P_0 = S_0 = \{0^n\}$, we obtain the following algorithm to solve the knapsack problem.

Nemhauser-Ullmann algorithm

```
$P_0 := \{0^n\}$;
**for** $i = 1, \dots, n$ **do**
    $Q_i := P_{i-1} \quad P^{+i}_{i-1}$;
    $P_i := \{x \quad Q_i \mid \quad y \quad Q_i : y$ dominates $x\}$;
**return** $x^\star := \arg\max_{x \quad P_n} \{p^\mathsf{T} x \mid w^\mathsf{T} x \leq W\}$;
```

The Nemhauser-Ullmann algorithm is illustrated in Figure 6.2.

We analyze the running time of the Nemhauser-Ullmann algorithm using the model of a unit-cost RAM. In this model, arithmetic operations like adding and comparing two numbers can be performed in constant time regardless of their bit-lengths. We use this model for the sake of simplicity and in order to keep the focus on the important details of the running time analysis.

Theorem 6.4. *The Nemhauser-Ullmann algorithm solves the knapsack problem optimally. There exists an implementation with running time $\Theta(\sum_{i=0}^{n-1} |P_i|)$.*

Proof. The correctness of the algorithm follows immediately from the previous discussion. In order to achieve the claimed running time, we do not compute the sets P_i explicitly, but only the values of the solutions in these sets. That is, instead of P_i only the set $\mathrm{val}(P_i) := \{(p^\top x, w^\top x) \mid x \quad P_i\}$ is computed. Analogously to the computation of P_i, one can compute $\mathrm{val}(P_i)$ easily if $\mathrm{val}(P_{i-1})$ is known. If we store for each element of $\mathrm{val}(P_i)$ a pointer to the element of $\mathrm{val}(P_{i-1})$ from which it originates, then in Step 5 the solution $x^\star$ can be efficiently reconstructed from the sets $\mathrm{val}(P_i)$ and these pointers. We leave the details of this reconstruction as an exercise to the reader.

The running times of Steps 1 and 5 are $O(1)$ and $O(n + |P|)$, respectively, where the term n accounts for the running time of reconstructing the solution $x^\star$ once its value $(p^\top x^\star, w^\top x^\star)$ is determined. In every iteration i of the for loop, the running time of Step 3 to compute $\mathrm{val}(Q_i)$ is $O(|P_{i-1}|)$ because on a unit-cost RAM the set $\mathrm{val}(P_{i-1}^{+i})$ can be computed in time $O(|P_{i-1}|)$ from the set $\mathrm{val}(P_{i-1})$. In a straightforward implementation, the running time of Step 4 is $\Theta(|Q_i|^2) = \Theta(|P_{i-1}|^2)$ because we need to compare every pair of values from $\mathrm{val}(Q_i)$ and each comparison takes time $O(1)$.

Step 4 can be implemented more efficiently. For this, we store the values in each set $\mathrm{val}(P_i)$ sorted in non-decreasing order of weights. If $\mathrm{val}(P_{i-1})$ is sorted in this way, then, without any additional computational effort, the computation of the set $\mathrm{val}(Q_i)$ in Step 3 can be implemented such that $\mathrm{val}(Q_i)$ is also sorted: The sorted set $\mathrm{val}(P_{i-1}^{+i})$ can be computed in time $\Theta(|P_{i-1}|)$. Then, in order to compute the set $\mathrm{val}(Q_i)$, only the two sorted sets $\mathrm{val}(P_{i-1})$ and $\mathrm{val}(P_{i-1}^{+i})$ need to be merged in time $\Theta(|P_{i-1}|)$. If the set $\mathrm{val}(Q_i)$ is sorted, Step 4 can be implemented to run in time $\Theta(|Q_i|)$ as a sweep algorithm going once through $\mathrm{val}(Q_i)$ in non-decreasing order of weights. □

We advise the reader to think about the details of the previously discussed implementation and to actually implement the Nemhauser-Ullmann algorithm so that it achieves the running time claimed in Theorem 6.4.

6.2 Number of Pareto-optimal Solutions

The running time of the Nemhauser-Ullmann algorithm is polynomially bounded in the size of the instance and the size of the Pareto sets. Hence, the algorithm runs in polynomial time on instances with a polynomial number of Pareto-optimal solutions. It is, however, easy to construct instances of the knapsack problem with exponentially many Pareto-optimal solutions. Thus, not surprisingly, the Nemhauser-Ullmann algorithm does not run in polynomial time in the worst case. On the other hand, it works well in practice and it can solve instances with thousands of items efficiently.

In this section, we study the number of Pareto-optimal solutions in the model of smoothed analysis. In light of the discussion in Section 5.5, it would be reasonable to consider instances of the knapsack problem in which the profits, the weights, and the capacity are ϕ-perturbed numbers. It turns out, however, that we do not even need that much randomness for our analysis. In fact, we assume that only the weights are ϕ-perturbed numbers, while the profits and the capacity are chosen adversarially. This makes the adversary stronger but it suffices to prove a polynomial bound on the smoothed number of Pareto-optimal solutions. Let us remark that all results that we prove in the following would also be true if the weights and the capacity are adversarial, while the profits are ϕ-perturbed numbers.

6.2.1 Upper Bound

In this section, we will prove an upper bound on the smoothed number of Pareto-optimal solutions. We assume in the following that the profits $p_1, \dots, p_n$ are arbitrary and that the weights $w_1, \dots, w_n$ are arbitrary ϕ-perturbed numbers from the interval $[0, 1]$. That is, the adversary can choose for each weight w_i a density function $f_i : [0, 1] \to [0, \phi]$ according to which w_i is chosen independently of the other weights. The restriction of the weights to the interval $[0, 1]$ is only a scaling issue and without loss of generality.

The proof of the following theorem is due to Beier et al. [BRV07]. Their result is much more general, but we present only the following simplified version in this lecture. We discuss some extensions of it in Section 6.3.

Theorem 6.5 ([BRV07]). *Let the profits $p_1, \dots, p_n$ be arbitrary and let the weights $w_1, \dots, w_n$ be arbitrary ϕ-perturbed numbers from the interval $[0, 1]$. Then the expected number of solutions $x \quad \{0, 1\}^n$ that are Pareto-optimal with respect to the objective functions $p^\mathsf{T} x$ and $w^\mathsf{T} x$ is bounded from above by $n^2\phi + 1$.*

Proof. We denote the set of Pareto-optimal solutions by $\mathcal{P}$. Every solution $x \quad \{0, 1\}^n$ has a weight $w^\mathsf{T} x$ in the interval $[0, n]$ because each weight w_i lies in the interval $[0, 1]$. We partition the interval $(0, n]$ uniformly into $k \quad \mathbb{N}$ intervals $I_0^k, \dots, I_{k-1}^k$ for some large number k to be chosen later. Formally, let $I_i^k = (ni/k, n(i + 1)/k]$.

We denote by X^k one plus the number of intervals I_i^k for which there exists at least one Pareto-optimal solution $x \quad \mathcal{P}$ with $w^\mathsf{T} x \quad I_i^k$. The term $+1$ comes from the solution 0^n, which is always Pareto-optimal and does not belong to any interval I_i^k. Nevertheless, the variable X^k can be much smaller than $|\mathcal{P}|$ because many Pareto-optimal solutions can lie in the same interval I_i^k. The main idea of the proof is that if the subintervals I_i^k are sufficiently small (that is, if k is sufficiently large), then it is very unlikely that there exists a subinterval that contains more than one Pareto-optimal solution. If every subinterval contains at most one Pareto-optimal solution, then $|\mathcal{P}| = X^k$.

In the following, we make this argument more formal. For $k \quad \mathbb{N}$, let $\mathcal{F}_k$ denote the event that there exist two different solutions $x, y \quad \{0, 1\}^n$ with $|w^\mathsf{T} x - w^\mathsf{T} y| \le n/k$.

Lemma 6.6. *For every* $k \quad \mathsf{N}$, $\mathbf{Pr}\left(F_k\right) \leq \frac{2^{2n+1}n\phi}{k}$.

Proof. There are at most 2^{2n} choices for x and y. We prove the lemma by a union bound over all these choices. Let $x, y \quad \{0,1\}^n$ with $x = y$ be fixed. Then there exists an index i with $x_i = y_i$. Assume without loss of generality that $x_i = 0$ and $y_i = 1$. We use the principle of deferred decisions and assume that all weights w_j except for w_i are already revealed. Then $w^\mathsf{T}x - w^\mathsf{T}y = \kappa - w_i$ for some constant κ that depends on x and y and the revealed weights w_j. It holds

$$\mathbf{Pr}\left(|w^\mathsf{T}x - w^\mathsf{T}y| \leq \frac{n}{k}\right) \leq \mathbf{Pr}\left(|\kappa - w_i| \leq \frac{n}{k}\right) = \mathbf{Pr}\left(w_i \quad \left[\kappa - \frac{n}{k}, \kappa + \frac{n}{k}\right]\right) \leq \frac{2n\phi}{k},$$

where the last inequality follows from Lemma 5.6 because the density of w_i is bounded from above by ϕ. Now the union bound over all choices for x and y concludes the proof. □

The following lemma is the main building block in the proof of the theorem.

Lemma 6.7. *For every* $k \quad \mathsf{N}$, $\mathbf{E}\left[X^k\right] \leq n^2\phi + 1$.

Proof. Let X_i^k denote a random variable that is 1 if the interval I_i^k contains at least one Pareto-optimal solution and 0 otherwise. Then

$$X^k = 1 + \sum_{i=0}^{k-1} X_i^k$$

and by linearity of expectation

$$\mathbf{E}\left[X^k\right] = \mathbf{E}\left[1 + \sum_{i=0}^{k-1} X_i^k\right] = 1 + \sum_{i=0}^{k-1} \mathbf{E}\left[X_i^k\right]. \tag{6.1}$$

Since X_i^k is a random variable that takes only the values 0 or 1, its expected value can be written as follows:

$$\mathbf{E}\left[X_i^k\right] = 0 \cdot \mathbf{Pr}\left(X_i^k = 0\right) + 1 \cdot \mathbf{Pr}\left(X_i^k = 1\right) = \mathbf{Pr}\left(X_i^k = 1\right) = \mathbf{Pr}\left(\quad x \quad \mathcal{P} \mid w^\mathsf{T}x \quad I_i^k\right). \tag{6.2}$$

With the help of the following lemma, whose proof is given further below, we can conclude the proof of the lemma.

Lemma 6.8. *For every* $t \geq 0$ *and every* $\varepsilon > 0$,

$$\mathbf{Pr}\left(\quad x \quad \mathcal{P} \mid w^\mathsf{T}x \quad (t, t+\varepsilon]\right) \leq n\phi\varepsilon.$$

Lemma 6.8 and (6.2) imply

$$\mathbf{E}\left[X_i^k\right] \leq \frac{n^2\phi}{k}.$$

Together with (6.1) this implies

$$\mathbf{E}\left[X^k\right] = 1 + \sum_{i=0}^{k-1} \mathbf{E}\left[X_i^k\right] \leq 1 + k \cdot \frac{n^2\phi}{k} = n^2\phi + 1.$$ □

With the help of the previous lemmas, we can finish the proof of the theorem as follows:

$$\begin{aligned}
\mathbf{E}\left[|P|\right] &\overset{(1)}{=} \sum_{i=0}^{2^n} \Big(i \cdot \mathbf{Pr}\left(|P| = i\right) \Big) \\
&\overset{(2)}{=} \sum_{i=0}^{2^n} \Big(i \cdot \mathbf{Pr}\left(|P| = i \quad F_k\right) + i \cdot \mathbf{Pr}\left(|P| = i \quad \neg F_k\right) \Big) \\
&\overset{(3)}{=} \sum_{i=0}^{2^n} \Big(i \cdot \mathbf{Pr}\left(F_k\right) \cdot \mathbf{Pr}\left(|P| = i \mid F_k\right) \Big) + \sum_{i=0}^{2^n} \Big(i \cdot \mathbf{Pr}\left(X^k = i \quad \neg F_k\right) \Big) \\
&\overset{(4)}{\leq} \mathbf{Pr}\left(F_k\right) \cdot \sum_{i=0}^{2^n} \Big(i \cdot \mathbf{Pr}\left(|P| = i \mid F_k\right) \Big) + \sum_{i=0}^{2^n} \Big(i \cdot \mathbf{Pr}\left(X^k = i\right) \Big) \\
&\overset{(5)}{\leq} \frac{2^{2n+1} n\phi}{k} \cdot \sum_{i=0}^{2^n} \Big(i \cdot \mathbf{Pr}\left(|P| = i \mid F_k\right) \Big) + \mathbf{E}\left[X^k\right] \\
&\overset{(6)}{\leq} \frac{2^{3n+1} n\phi}{k} + n^2\phi + 1. \qquad (6.3)
\end{aligned}$$

Let us comment on the steps in the previous calculation.

- (1) follows from the definition of the expected value.
- (2) follows from the law of total probability.
- The rewriting of the first term in (3) follows from the definition of the conditional probability and the rewriting of the second term follows because $X^k = |P|$ when the event $\neg F_k$ occurs.
- (4) follows because $\mathbf{Pr}\left(A \cap B\right) \leq \mathbf{Pr}\left(A\right)$ for all events A and B.
- (5) follows from Lemma 6.6 and the definition of the expected value.
- (6) follows from the identity $\sum_{i=0}^{2^n} \mathbf{Pr}\left(|P| = i \mid F_k\right) = 1$ and Lemma 6.7.

Since (6.3) holds for every $k \quad \mathbb{N}$, it must be $\mathbf{E}\left[|P|\right] \leq n^2\phi + 1$, which proves the theorem. □

It only remains to prove Lemma 6.8.

Proof of Lemma 6.8. First of all we define a random variable $\Lambda(t)$ such that

$$\Lambda(t) \leq \varepsilon \qquad\qquad x \quad P : w^{\mathsf{T}}x \quad (t, t+\varepsilon]. \qquad (6.4)$$

In order to define $\Lambda(t)$, we define the ***winner*** $x^\star$ to be the most valuable solution satisfying $w^{\mathsf{T}}x \leq t$, i.e.,

$$x^\star = \arg\max\{p^{\mathsf{T}}x \mid x \quad \{0,1\}^n \text{ and } w^{\mathsf{T}}x \leq t\}.$$

For $t \geq 0$, such a solution $x^\star$ must always exist. We say that a solution x is a ***loser*** if it has a higher profit than $x^\star$. By the choice of $x^\star$, losers do not satisfy the constraint $w^{\mathsf{T}}x \leq t$ (hence their name). We denote by $\hat{x}$ the loser with the smallest weight (see Figure 6.3), i.e.,

$$\hat{x} = \arg\min\{w^{\mathsf{T}}x \mid x \quad \{0,1\}^n \text{ and } p^{\mathsf{T}}x > p^{\mathsf{T}}x^\star\}.$$

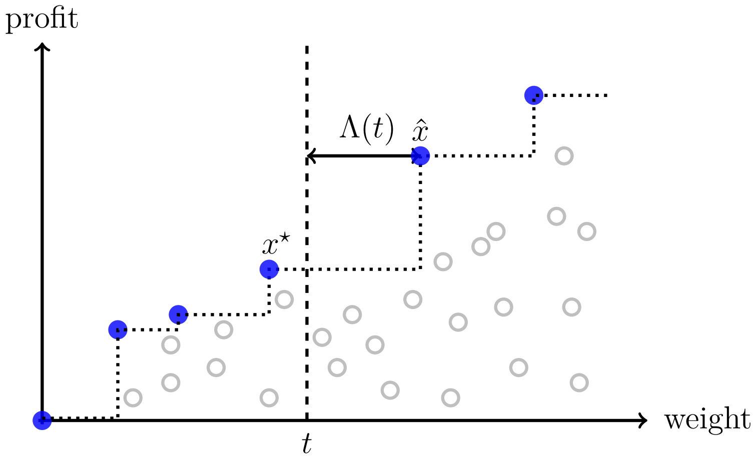


Figure 6.3: Definitions of the winner $x^\star$, the loser $\hat{x}$, and the random variable $\Lambda(t)$.

If there does not exist a solution x with $p^\mathsf{T}x > p^\mathsf{T}x^\star$, then $\hat{x}$ is undefined, which we denote by $\hat{x} = \ $. Based on $\hat{x}$, we define the random variable $\Lambda(t)$ as

$$\Lambda(t) = \begin{cases} w^\mathsf{T}\hat{x} - t & \text{if } \hat{x} = \quad , \\ & \text{if } \hat{x} = \quad . \end{cases}$$

Assume that there exists a Pareto-optimal solution whose weight lies in $(t, t+\varepsilon]$, and let y denote the Pareto-optimal solution with the smallest weight in $(t, t+\varepsilon]$. Then $y = \hat{x}$ and hence $\Lambda(t) = w^\mathsf{T}\hat{x} - t \quad (0, \varepsilon]$. Conversely, if $\Lambda(t) \leq \varepsilon$, then $\hat{x}$ must be a Pareto-optimal solution whose weight lies in the interval $(t, t+\varepsilon]$. Together this yields Equivalence (6.4). Hence,

$$\mathbf{Pr}\left(\quad x \quad \mathsf{P} \mid w^\mathsf{T}x \quad (t, t+\varepsilon]\right) = \mathbf{Pr}\left(\Lambda(t) \leq \varepsilon\right). \tag{6.5}$$

It only remains to bound the probability that $\Lambda(t)$ does not exceed ε. In order to analyze this probability, we define a set of auxiliary random variables such that $\Lambda(t)$ is guaranteed to always take a value also taken by at least one of the auxiliary random variables. Then we analyze the auxiliary random variables and use a union bound to conclude the desired bound for $\Lambda(t)$. Let $i \quad [n]$ be fixed. For $j \quad \{0, 1\}$, we define

$$\mathsf{S}^{x_i=j} = \{x \quad \{0,1\}^n \mid x_i = j\},$$

and we define $x^{\star,i}$ to be

$$x^{\star,i} = \arg\max\{p^\mathsf{T}x \mid x \quad \mathsf{S}^{x_i=0} \text{ and } w^\mathsf{T}x \leq t\}.$$

That is, $x^{\star,i}$ is the winner among the solutions that do not contain item i. We restrict our attention to losers that contain item i and define

$$\hat{x}^i = \arg\min\{w^\mathsf{T}x \mid x \quad \mathsf{S}^{x_i=1} \text{ and } p^\mathsf{T}x > p^\mathsf{T}x^{\star,i}\}.$$

If there does not exist a solution $x \quad \mathsf{S}^{x_i=1}$ with $p^\mathsf{T}x > p^\mathsf{T}x^{\star,i}$, then $\hat{x}^i$ is undefined, i.e., $\hat{x}^i = \ $. Based on $\hat{x}^i$, we define the random variable $\Lambda^i(t)$ as

$$\Lambda^i(t) = \begin{cases} w^\mathsf{T}\hat{x}^i - t & \text{if } \hat{x}^i = \quad , \\ & \text{if } \hat{x}^i = \quad . \end{cases}$$

Summarizing, $\Lambda^i(t)$ is defined similarly to $\Lambda(t)$, but only solutions that do not contain item i are eligible as winners and only solutions that contain item i are eligible as losers.

Lemma 6.9. *For every choice of profits and weights, either* $\Lambda(t) =$ *or there exists an index* i $[n]$ *such that* $\Lambda(t) = \Lambda^i(t)$.

Proof. Assume that $\Lambda(t) =$. Then there exist a winner $x^\star$ and a loser $\hat{x}$. Since $x^\star = \hat{x}$, there must be an index i $[n]$ with $x^\star_i = \hat{x}_i$. Since all weights are positive and $w^\mathsf{T} x^\star < w^\mathsf{T}\hat{x}$, there must even be an index i $[n]$ with $x^\star_i = 0$ and $\hat{x}_i = 1$. We claim that for this index i, $\Lambda(t) = \Lambda^i(t)$. In order to see this, we first observe that $x^\star = x^{\star,i}$. This follows because $x^\star$ is the solution with the highest profit among all solutions with weight at most t. Since it belongs to $\mathcal{S}^{x_i=0}$ it is in particular the solution with the highest profit among all solutions that do not contain item i and have weight at most t. Since $x^\star = x^{\star,i}$, by similar arguments it follows that $\hat{x} = \hat{x}^i$. This directly implies that $\Lambda(t) = \Lambda^i(t)$. □

Lemma 6.10. *For every* i $[n]$ *and every* $\varepsilon \geq 0$,

$$\mathbf{Pr}\left(\Lambda^i(t) \quad (0,\varepsilon]\right) \leq \phi\varepsilon.$$

Proof. In order to prove the lemma, it suffices to exploit the randomness of the weight w_i. We apply the principle of deferred decisions and assume that all other weights are fixed arbitrarily. Then the weights of all solutions from $\mathcal{S}^{x_i=0}$ and hence also the solution $x^{\star,i}$ are fixed. If the solution $x^{\star,i}$ is fixed, then also the set of losers $\mathcal{L} = \{x \quad \mathcal{S}^{x_i=1} \mid p^\mathsf{T} x > p^\mathsf{T} x^{\star,i}\}$ is fixed. Since, by definition, all solutions from $\mathcal{L}$ contain item i the identity of the solution $\hat{x}^i$ does not depend on w_i. (Of course, the weight $w^\mathsf{T}\hat{x}^i$ depends on w_i. Which solution will become $\hat{x}^i$ is, however, independent of w_i.) This implies that, given the fixed values of the weights w_j with $j = i$, we can rewrite the event $\Lambda^i(t) \quad (0,\varepsilon]$ as $w^\mathsf{T}\hat{x}^i - t \quad (0,\varepsilon]$ for a fixed solution $\hat{x}^i$. For a constant $\kappa \quad \mathbb{R}$ depending on the fixed values of the weights w_j with $j = i$, we can rewrite this event as $w_i \quad (\kappa, \kappa+\varepsilon]$. By Lemma 5.6, the probability of this event is bounded from above by $\phi\varepsilon$. □

Combining Lemmas 6.9 and 6.10 yields

$$\mathbf{Pr}\left(\Lambda(t) \leq \varepsilon\right) \leq \mathbf{Pr}\left(\quad i \quad [n] \colon \Lambda^i(t) \quad (0,\varepsilon]\right) \leq \sum_{i=1}^{n} \mathbf{Pr}\left(\Lambda^i(t) \quad (0,\varepsilon]\right) \leq n\phi\varepsilon.$$

Together with (6.5) this proves the lemma. □

Theorem 6.5 implies the following result on the running time of the Nemhauser-Ullmann algorithm.

Corollary 6.11. *For an instance of the knapsack problem with* n *items with arbitrary profits and* ϕ*-perturbed weights from* $[0,1]$*, the expected running time of the Nemhauser-Ullmann algorithm is* $O(n^3\phi)$.

Proof. It follows from Theorem 6.4 that the expected running time of the Nemhauser-Ullmann algorithm is bounded from above by

$$O\left(\mathbf{E}\left[\sum_{i=0}^{n-1} |P_i|\right]\right),$$

where P_i denotes the Pareto set of the restricted instance that consists only of the first i items. Using linearity of expectation and Theorem 6.5, we obtain that this term is bounded from above by

$$O\left(\sum_{i=0}^{n-1} \mathbf{E}\left[|P_i|\right]\right) = O\left(\sum_{i=0}^{n-1} (i^2\phi + 1)\right) = O(n^3\phi). \qquad \Box$$

6.2.2 Lower Bound

In this section we prove that the upper bound for the smoothed number of Pareto-optimal solutions in Theorem 6.5 is asymptotically tight in terms of n. In the lower bound we assume that there are n items and that the adversary has chosen profits $p_i = 2^i$. We only consider the case $\phi = 1$, in which every weight is chosen uniformly at random from $[0,1]$. The following theorem is due to Beier and Vöcking [BV04b].

Theorem 6.12 ([BV04b])**.** *Consider an instance of the knapsack problem with n items with profits $p_i = 2^i$ for $i \in [n]$. Let the weights $w_1, \dots, w_n$ be chosen independently and uniformly at random from $[0,1]$. Then the expected number of solutions $x \in \{0,1\}^n$ that are Pareto-optimal with respect to the objective functions $p^\mathsf{T} x$ and $w^\mathsf{T} x$ is $(n^2 + 3n)/4 + 1 = \Theta(n^2)$.*

Proof. In the following let P_j denote the Pareto set of the instance that consists only of the first j items. We have shown in Lemma 6.3 that P_j is a subset of $P_{j-1} \cup P_{j-1}^{+j}$. The choice of profits guarantees that every solution from P_{j-1}^{+j} has a strictly larger profit than any solution from P_{j-1} because $p_j > \sum_{i=1}^{j-1} p_i$. This implies that solutions from P_{j-1}^{+j} cannot be dominated by solutions from P_{j-1}. On the other hand, the solution 0^n is always Pareto-optimal and hence $0^n \in P_{j-1}$, which implies that P_{j-1}^{+j} contains the solution that consists only of item j. This solution has a higher profit than any solution from P_{j-1} and hence it dominates all solutions from P_{j-1} that have weight at least w_j. Since all solutions in P_{j-1}^{+j} have weight at least w_j, no solution from P_{j-1} with weight less than w_j is dominated. This is illustrated in Figure 6.4. Hence we have

$$P_j = P_{j-1}^{+j} \cup \{x \in P_{j-1} \mid w^\mathsf{T} x < w_j\}. \tag{6.6}$$

It follows from (6.6) that $|P_j| \geq |P_{j-1}^{+j}| = |P_{j-1}|$ for every j. We define $Y^j = |P_j| - |P_{j-1}|$ to be the increase in the number of Pareto-optimal solutions when item j is added to the instance. Then

$$|P| = 1 + \sum_{j=1}^{n} Y^j, \tag{6.7}$$

where the term $+1$ comes from the solution 0^n, which is already contained in P_0.

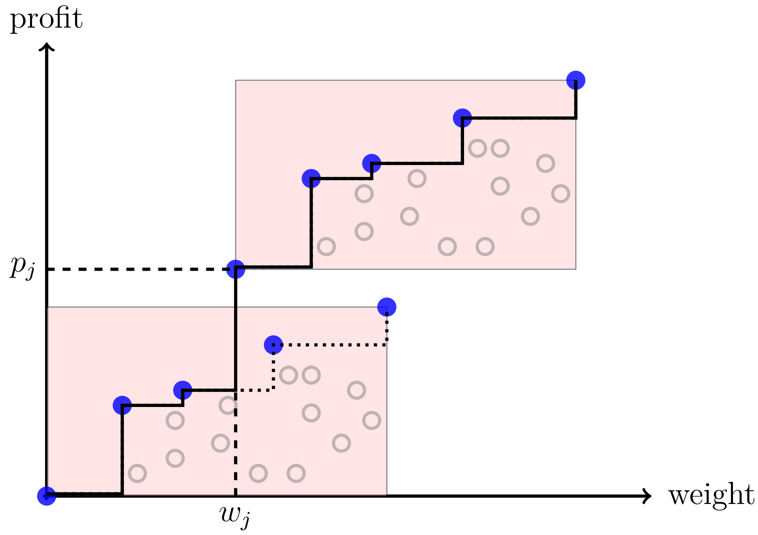


Figure 6.4: Illustration of Equation (6.6): The Pareto set $\mathcal{P}_j$ consists of all solutions from $\mathcal{P}_{j-1}^{+j}$ and all solutions from $\mathcal{P}_{j-1}$ except for the last two because their weights exceed w_j.

In order to analyze the random variables Y^j, we introduce an auxiliary random variable X_α^j for every $j \quad [n]$ and every $\alpha \quad [0,1]$. It is defined as the number of Pareto-optimal solutions from $\mathcal{P}_j$ with weights in the interval $(0,\alpha)$, i.e.,

$$X_\alpha^j = |\{x \quad \mathcal{P}_j \mid w^\top x \quad (0,\alpha)\}|.$$

Since $\mathcal{P}_1 = \{0^n, 10^{n-1}\}$, it follows immediately from the previous definition that

$$X_\alpha^1 = \begin{cases} 0 & \text{if } w_1 \geq \alpha, \\ 1 & \text{if } w_1 < \alpha. \end{cases}$$

This implies

$$\mathbf{E}\left[X_\alpha^1\right] = \mathbf{Pr}\left(w_1 < \alpha\right) = \alpha, \tag{6.8}$$

where we used that $\alpha \quad [0,1]$ and that w_1 is chosen uniformly at random from $[0,1]$.

For $j \geq 2$, the random variables X_α^j satisfy the following equation, which is illustrated in Figure 6.5:

$$X_\alpha^j = \begin{cases} X_\alpha^{j-1} & \text{if } w_j \geq \alpha, \\ 1 + X_{w_j}^{j-1} + X_{\alpha-w_j}^{j-1} & \text{if } w_j < \alpha. \end{cases} \tag{6.9}$$

In the following, let f_{w_j} denote the density of w_j. Using the law of total expectation, (6.9) implies

$$\begin{aligned} \mathbf{E}\left[X_\alpha^j\right] &= \int_0^1 f_{w_j}(t) \cdot \mathbf{E}\left[X_\alpha^j \mid w_j = t\right]\, dt \\ &= \int_0^1 \mathbf{E}\left[X_\alpha^j \mid w_j = t\right]\, dt \end{aligned}$$

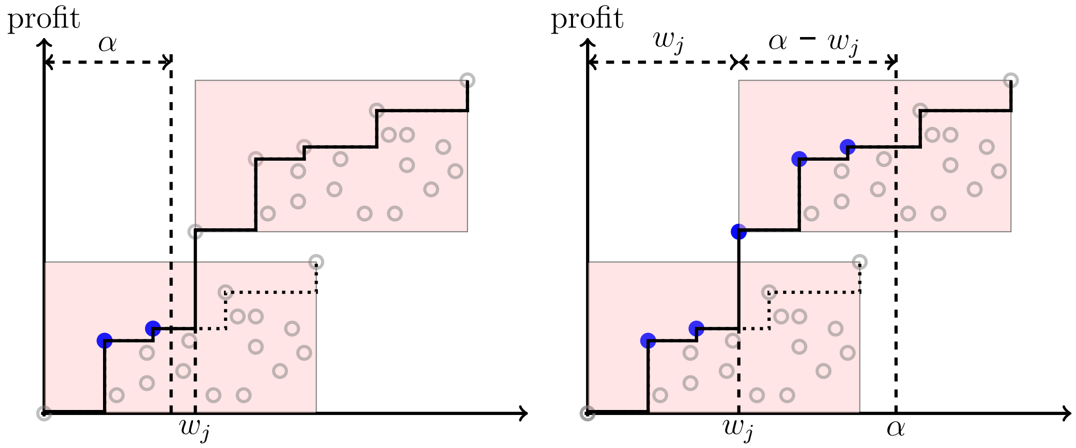


Figure 6.5: Illustration of the two cases in (6.9). In both cases only the blue solutions contribute to X_α^j. Observe that the solution 0^n does not contribute to X_α^{j-1} and X_α^j. The solution that consists only of item j contributes to X_α^j in the case $w_j < \alpha$ and is responsible for the term $+1$.

$$\begin{aligned}
&= \int_0^\alpha \mathbf{E}\left[X_\alpha^j \mid w_j = t\right]\, dt + \int_\alpha^1 \mathbf{E}\left[X_\alpha^j \mid w_j = t\right]\, dt \\
&= \int_0^\alpha \mathbf{E}\left[1 + X_{w_j}^{j-1} + X_{\alpha - w_j}^{j-1} \mid w_j = t\right]\, dt + \int_\alpha^1 \mathbf{E}\left[X_\alpha^{j-1} \mid w_j = t\right]\, dt \\
&= \int_0^\alpha \mathbf{E}\left[1 + X_t^{j-1} + X_{\alpha-t}^{j-1}\right]\, dt + \int_\alpha^1 \mathbf{E}\left[X_\alpha^{j-1}\right]\, dt \\
&= \int_0^\alpha 1 + \mathbf{E}\left[X_t^{j-1}\right] + \mathbf{E}\left[X_{\alpha-t}^{j-1}\right]\, dt + (1-\alpha)\cdot \mathbf{E}\left[X_\alpha^{j-1}\right]. \qquad (6.10)
\end{aligned}$$

In the last line of the previous calculation we integrate over all values t that w_j can take under the condition that $w_j \le \alpha$. We used that the density of w_j is 1 on $[0,\alpha]$.

Using (6.8) and (6.10) we can prove by induction that $\mathbf{E}[X_\alpha^j] = \alpha j$ for every α $\quad [0,1]$ and every j $\quad [n]$. Observe that (6.8) proves the base case $j = 1$. For $j \ge 2$ we obtain with (6.10)

$$\begin{aligned}
\mathbf{E}\left[X_\alpha^j\right] &= \int_0^\alpha 1 + \mathbf{E}\left[X_t^{j-1}\right] + \mathbf{E}\left[X_{\alpha-t}^{j-1}\right]\, dt + (1-\alpha)\cdot \mathbf{E}\left[X_\alpha^{j-1}\right] \\
&= \int_0^\alpha 1 + t(j-1) + (\alpha - t)(j-1)\, dt + (1-\alpha)\cdot \alpha(j-1) \\
&= \int_0^\alpha 1 + \alpha(j-1)\, dt + (1-\alpha)\cdot \alpha(j-1) \\
&= \alpha + \alpha\cdot\alpha(j-1) + (1-\alpha)\cdot\alpha(j-1) \\
&= \alpha(j-1) + \alpha = \alpha j.
\end{aligned}$$

Since $|\mathcal{P}_{j-1}| = |\mathcal{P}_{j-1}^{+j}|$, Equation (6.6) implies

$$Y^j = |\{x \quad \mathcal{P}_{j-1} \mid w^\mathsf{T} x < w_j\}| = X_{w_j}^{j-1} + 1.$$

The term $+1$ is due to the solution $0^n \quad \mathcal{P}_{j-1}$, which is not counted in $X_{w_j}^{j-1}$. Since the Pareto set $\mathcal{P}_{j-1}$ is independent of the random variable w_j, we can use the law of total expectation and calculate the expected value of Y^j as follows:

$$\begin{aligned}
\mathbf{E}\left[Y^j\right] &= \int_0^1 f_{w_j}(t) \cdot \mathbf{E}\left[Y^j \mid w_j = t\right] \, dt \\
&= \int_0^1 \mathbf{E}\left[Y^j \mid w_j = t\right] \, dt \\
&= \int_0^1 \mathbf{E}\left[X_{w_j}^{j-1} + 1 \mid w_j = t\right] \, dt \\
&= \int_0^1 \mathbf{E}\left[X_t^{j-1} + 1\right] \, dt \\
&= \int_0^1 (j-1)t + 1 \, dt = 1 + (j-1) \int_0^1 t \, dt = 1 + \frac{j-1}{2}.
\end{aligned}$$

Using (6.7) we obtain

$$\begin{aligned}
\mathbf{E}\left[|P|\right] = \mathbf{E}\left[1 + \sum_{j=1}^n Y^j\right] &= 1 + \sum_{j=1}^n \mathbf{E}\left[Y^j\right] = 1 + \sum_{j=1}^n \left(1 + \frac{j-1}{2}\right) \\
&= 1 + n + \frac{n(n-1)}{4} = \frac{n^2}{4} + \frac{3n}{4} + 1.
\end{aligned}$$

This concludes the proof. □

6.3 Multiobjective Optimization

The upper bound on the expected number of Pareto-optimal solutions that Beier et al. [BRV07] proved is much more general than the version that we have presented in Theorem 6.5.

Theorem 6.13 ([BRV07])**.** *Let $S \quad \{0,1\}^n$ and $p : S \to \mathsf{R}$ be arbitrarily chosen. Let $w_1, \ldots, w_n$ be arbitrary ϕ-perturbed numbers from the interval $[-1,1]$. Then the expected number of solutions $x \quad S$ that are Pareto-optimal with respect to the objective functions $p(x)$ and $w^{\mathsf{T}}x$ is $O(n^2\phi)$.*

The first generalization compared to Theorem 6.5 is that an arbitrary set $S \quad \{0,1\}^n$ of solutions is given. In the case of the knapsack problem, every vector from $\{0,1\}^n$ is a solution, i.e., $S = \{0,1\}^n$. The second generalization is that the adversarial objective function p does not have to be linear. In fact, it can be an arbitrary function that maps every solution to some real value. The third generalization is that the coefficients w_i are not restricted to positive values anymore. The restriction of the profits to the interval $[-1,1]$ is only a scaling issue and (almost) without loss of generality. The result holds regardless of whether the objective functions should be maximized or minimized.

We will not prove Theorem 6.13 in this lecture, but let us remark that its proof is very similar to the proof of Theorem 6.5. In fact we never used in that proof that $S = \{0,1\}^n$ and that p is linear. The fact that all weights w_i are positive was only used to argue that there must be an index i with $x_i^\star = 0$ and $\hat{x}_i = 1$. For general w_i, it could also be the other way round. Handling this issue is the only modification of the proof that is not completely straightforward.

To illustrate the power of Theorem 6.13, let us discuss its implications on graph problems. For a given graph with m edges $e_1, \ldots, e_m$, one can, for example, identify every vector $x \quad \{0,1\}^m$ with a subset of edges $E(x) = \{e_i \mid x_i = 1\}$. Then x is the so-called incidence vector of the edge set $E(x)$. If, for example, there is a source node s and a target node t given, one could choose the set S of feasible solutions as the set of all incidence vectors of paths from s to t in the given graph. This way, Theorem 6.13 implies that the smoothed number of Pareto-optimal paths in the bicriteria shortest-path problem is $O(m^2\phi)$. In this problem, every edge of the graph is assigned a weight and a cost, and one wants to find Pareto-optimal paths from s to t with respect to total weight and total cost. Similarly, one could choose S as the set of incidence vectors of all spanning trees of a given graph. Then the result implies that there are only $O(m^2\phi)$ Pareto-optimal spanning trees in expectation in the bicriteria spanning tree problem.

These results can even be generalized to optimization problems with one arbitrary objective function and d linear objective functions with ϕ-perturbed coefficients [BR15]. For these problems the bound becomes $O(n^{2d}\phi^{d(d+1)})$. It can even be improved to $O(n^{2d}\phi^d)$ if all densities are quasiconcave, where a function $f : \mathsf{R} \to \mathsf{R}$ is called quasiconcave if there exists some $x \quad \mathsf{R}$ such that f is monotonically increasing on $(-\infty, x]$ and monotonically decreasing on $[x, \infty)$.

6.4 Core Algorithms

Corollary 6.11 is an explanation why the Nemhauser-Ullmann algorithm performs better in practice than predicted by its worst-case running time. However, a running time in the order of n^3 is still not good enough for instances with millions of items. In this section, we will present *core algorithms* for the knapsack problem that have a much better (almost linear) running time. This section is based on an article by Beier and Vöcking [BV04a]

For a given instance of the knapsack problem with n items with profits $p_1, \ldots, p_n$, weights $w_1, \ldots, w_n$, and capacity W, core algorithms first compute an optimal solution of the *fractional knapsack problem*, in which items are divisible and can be packed fractionally into the knapsack:

$$\begin{aligned} \text{maximize} \quad & p^\mathsf{T} x = p_1 x_1 + \cdots + p_n x_n \\ \text{subject to} \quad & w^\mathsf{T} x = w_1 x_1 + \cdots + w_n x_n \leq W, \\ & \text{and } x = (x_1, \ldots, x_n)^\mathsf{T} \quad [0,1]^n. \end{aligned}$$

The following greedy algorithm computes an optimal solution of the fractional knapsack problem in time $O(n \log n)$.

Greedy algorithm for fractional knapsack problem

```
1: Sort the items in descending order by their profit-to-weight ratio $p_i/w_i$.
   Assume that after this step $p_1/w_1 \geq p_2/w_2 \geq \ldots \geq p_n/w_n$.
2: $t := W$;
3: for $i = 1, \ldots, n$ do
4:     if $w_i \leq t$ then
5:         $x_i := 1$;
6:         $t := t - w_i$;
7:     else
8:         $x_i := t/w_i$;
9:         $t := 0$;
```

It is left as an exercise for the reader to prove that the previous algorithm computes an optimal solution of the fractional knapsack problem. The running time of the algorithm is $O(n \log n)$ due to the sorting in Step 1. It is another exercise for the reader to find an algorithm that solves the fractional knapsack problem in time $O(n)$.

From the pseudocode of the greedy algorithm it is clear that it computes a solution in which all variables x_i except for at most one take values from $\{0, 1\}$: If the remaining capacity t is at least w_i, the variable x_i is set to 1. Otherwise, if $t > 0$ but $t < w_i$ the variable x_i is set to some fractional value from $(0, 1)$. After that $t = 0$ and all remaining variables are set to 0.

It can also happen that the algorithm computes a solution in which all variables have a value from $\{0, 1\}$. Then this solution is also an optimal solution of the non-fractional knapsack problem. Hence, this case is not interesting and we consider in the following only instances in which the greedy algorithm computes a solution $\bar{x}$ $[0, 1]^n$ in which exactly one variable takes a fractional value from $(0, 1)$. We call the item corresponding to this variable ***break item*** and we denote it by $i^\star$. We assume in the following that the items are sorted in descending order by their profit-to-weight ratio p_i/w_i. Then $\bar{x}_i = 1$ for all $i < i^\star$ and $\bar{x}_i = 0$ for all $i > i^\star$.

We call the ray that starts at the origin $(0, 0)$ and goes through $(w_{i^\star}, p_{i^\star})$ ***Dantzig ray***. For the sake of simplicity we assume in the following that $i^\star$ is the only item that lies on the Dantzig ray (this assumption is not necessary but it simplifies the presentation a bit). In the following, let r denote the slope of the Dantzig ray. The optimal fractional solution $\bar{x}$ has a simple geometric structure. It contains all items above the Dantzig ray while it does not contain any item below this ray. This is illustrated in Figure 6.6.

Now let $x^\star$ $\{0, 1\}^n$ denote an optimal solution of the non-fractional knapsack problem. Core algorithms are based on the observation that typically the solutions $x^\star$ and $\bar{x}$ differ only in a few items. Typically these items are close to the Dantzig ray while $x^\star$ and $\bar{x}$ agree on all items that have a significant distance from the Dantzig ray, i.e., all items significantly above this ray belong to $x^\star$ and all items significantly below this ray do not belong to $x^\star$. We call the set of all items that do not have a significant distance from the Dantzig ray the ***core***.

To make this more formal, we first define the ***loss*** ℓ_i of item i to be its vertical distance from the Dantzig ray, i.e., $\ell_i := |p_i - rw_i|$ (see Figure 6.6). For a solution x $\{0, 1\}^n$

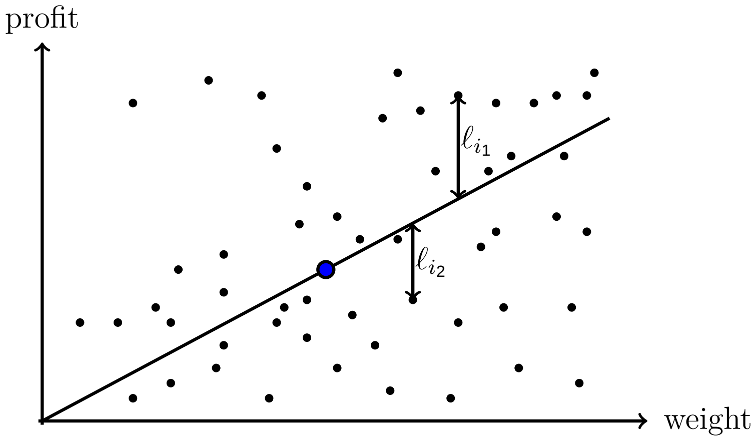


Figure 6.6: In this figure each item i corresponds to a dot at position (w_i, p_i). The break item lies on the Dantzig ray and it is shown larger than the other items. The loss of two items i_1 and i_2 is shown exemplary.

we define $\Delta(x) \quad [n]$ to be the set of items on which x and $\bar{x}$ disagree, i.e., $\Delta(x) = \{i \quad [n] \mid x_i = \bar{x}_i\}$. With the help of these definitions we can express the gap between the profits of the optimal fractional solution $\bar{x}$ and some binary solution $x \quad \{0,1\}^n$ as follows.

Lemma 6.14. *Let* $x \quad \{0,1\}^n$ *be an arbitrary solution. Then*

$$p^{\mathsf{T}}\bar{x} - p^{\mathsf{T}}x = r(W - w^{\mathsf{T}}x) + \sum_{i \quad \Delta(x)} \ell_i.$$

Intuitively the profit of the solution x can be smaller than the profit of $\bar{x}$ for two reasons. First it might be the case that the solution x does not use the complete capacity $W = w^{\mathsf{T}}\bar{x}$ of the knapsack. Second all items above the Dantzig ray that x does not contain and all items below the Dantzig ray that x contains also contribute to the gap between the profits of $\bar{x}$ and x. This is captured by the two terms in the sum.

Proof. Let $\Delta_0(x) = \{i \quad [n] \mid i < i^\star \text{ and } x_i = 0\}$ and $\Delta_1(x) = \{i \quad [n] \mid i > i^\star \text{ and } x_i = 1\}$. Then $\Delta(x) = \Delta_0(x) \quad \Delta_1(x) \quad \{i^\star\}$. For every item $i \quad \Delta_0(x)$, we have

$$p_i\bar{x}_i - p_i x_i = p_i = rw_i + \ell_i$$

and for every item $i \quad \Delta_1(x)$, we have

$$p_i\bar{x}_i - p_i x_i = -p_i = -(rw_i - \ell_i) = -rw_i + \ell_i.$$

Furthermore,

$$p_{i^\star}\bar{x}_{i^\star} - p_{i^\star}x_{i^\star} = rw_{i^\star}(\bar{x}_{i^\star} - x_{i^\star}).$$

Now taking the sum yields

$$p^{\mathsf{T}}\bar{x} - p^{\mathsf{T}}x = \sum_{i=1}^{n}(p_i\bar{x}_i - p_i x_i)$$

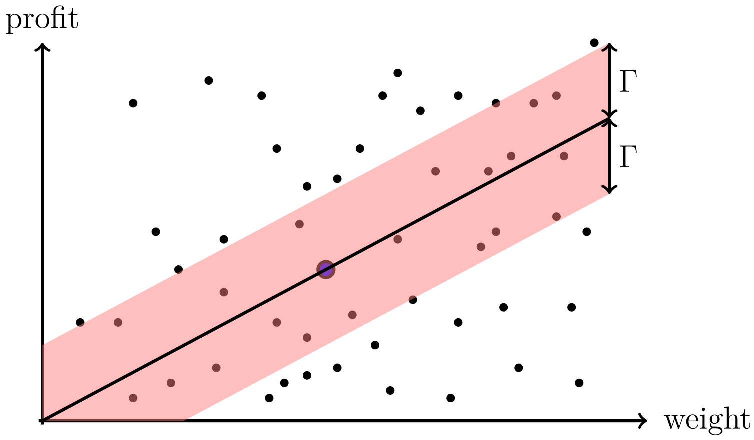


Figure 6.7: The core consists of all items whose loss is at most Γ.

$$
\begin{aligned}
&= \Bigg(\sum_{i \quad \Delta_0(x)} rw_i + \ell_i \Bigg) + \Bigg(\sum_{i \quad \Delta_1(x)} -rw_i + \ell_i \Bigg) + rw_{i^\star}(\bar{x}_{i^\star} - x_{i^\star}) \\
&= r\Bigg(\Bigg(\sum_{i \quad \Delta_0(x)} w_i \Bigg) - \Bigg(\sum_{i \quad \Delta_1(x)} w_i \Bigg) + w_{i^\star}(\bar{x}_{i^\star} - x_{i^\star}) \Bigg) + \sum_{i \quad \Delta_0(x) \quad \Delta_1(x)} \ell_i \\
&= r(w^\intercal \bar{x} - w^\intercal x) + \sum_{i \quad \Delta(x)} \ell_i \\
&= r(W - w^\intercal x) + \sum_{i \quad \Delta(x)} \ell_i .
\end{aligned}
$$

In the second to last line we used that $\Delta(x) = \Delta_0(x) \quad \Delta_1(x) \quad \{i^\star\}$ and that $\ell_{i^\star} = 0$. □

Assume that we know the ***additive integrality gap*** Γ of the given instance of the knapsack problem, i.e., $\Gamma = p^\intercal \bar{x} - p^\intercal x^\star$, where $\bar{x}$ and $x^\star$ denote the optimal fractional and non-fractional solutions, respectively. Then we could use the previous lemma to define the core C.

Corollary 6.15. *It holds*

$$\Delta(x^\star) \quad C := \{i \quad [n] \mid \ell_i \le \Gamma\}.$$

That is, all items with a loss greater than Γ above the Dantzig ray are contained in $x^\star$ while all items with a loss greater than Γ below the Dantzig ray are not contained in $x^\star$.

Proof. Assume that there exists an item i with loss $\ell_i > \Gamma$ and $i \quad \Delta(x^\star)$. Then with Lemma 6.14 we obtain the following contradiction

$$\Gamma = p^\intercal \bar{x} - p^\intercal x^\star \ge \ell_i > \Gamma. \qquad \square$$

The previous corollary is illustrated in Figure 6.7. If Γ is known, then one only needs to solve the following knapsack problem on the items in the core:

$$\text{maximize} \quad \sum_{i \quad C} p_i x_i$$

$$\text{subject to}\quad \sum_{i\quad C} w_i x_i \le W\ ,$$
$$\text{and}\quad i\quad C : x_i\quad \{0,1\},$$

where $W\ $ is defined as the remaining capacity if one takes into account that all items above the Dantzig ray that do not belong to the core are definitely contained in the optimal non-fractional solution $x^\star$, i.e.,

$$W\ = W - \sum_{i:\ell_i>\Gamma, i<i^\star} w_i.$$

This way, we have reduced the given instance of the knapsack problem to an instance with potentially fewer items. The hope is that typically there are only a few items in the core such that the remaining instance consists only of a few items. This is indeed observed in practice and proved by Beier and Vöcking [BV04a] for instances in which every weight and every profit is chosen uniformly at random from $[0,1]$.

If the remaining instance consists only of a logarithmic number of items then it can even be solved by a brute-force search in polynomial time. However, the remaining instance can also be solved by the Nemhauser-Ullmann algorithm, which yields an algorithm that runs in almost linear time on random inputs. The following theorem has been proved by Beier and Vöcking [BV04a]. As its proof is rather involved, we will not discuss it in the lecture. We will also not discuss their algorithm in detail and leave it at the main idea as discussed above.

Theorem 6.16 ([BV04a])**.** *There exists a core algorithm, which solves the remaining instance by the Nemhauser-Ullmann algorithm, with expected running time $O(n \log^c n)$ for some constant c on instances with n items in which every profit and every weight is chosen independently and uniformly at random from $[0,1]$ and the capacity is chosen as $W = \beta n$ for some $\beta\quad (0,1/2)$.*

Let us now briefly discuss the issue that our definition of the core requires Γ to be known in advance. This is of course not a practical assumption because Γ is unknown and there is also no easy way to compute it. Most implementations of core algorithms try to estimate Γ. One variant, the *expanding core algorithm*, first chooses the smallest value γ such that the core

$$C(\gamma) := \{i\quad [n] \mid \ell_i \le \gamma\}$$

with respect to γ contains at least one item apart from the break item. Then it computes the optimal solution x with respect to the core $C(\gamma)$, i.e., x is the solution that contains all items above $C(\gamma)$, no item below $C(\gamma)$, and the optimal subset of items from $C(\gamma)$. For $\gamma \ge \Gamma$, the solution x coincides with $x^\star$ according to Corollary 6.15. For smaller γ, the solution x is in general not optimal.

If $p^\mathsf{T}\bar{x} - p^\mathsf{T}x \le \gamma$, we can be sure that $\gamma \ge \Gamma$, which implies that x is an optimal solution, because

$$\Gamma = p^\mathsf{T}\bar{x} - p^\mathsf{T}x^\star \le p^\mathsf{T}\bar{x} - p^\mathsf{T}x \le \gamma.$$

If $p^\mathsf{T}\bar{x} - p^\mathsf{T}x > \gamma$, we increase γ and compute the optimal solution with respect to the enlarged core $C(\gamma)$. This algorithm is shown in the following pseudocode.

Expanding Core Algorithm

```
1: Compute the optimal fractional solution $\bar{x}$.
2: $\gamma := 0$
3: **repeat**
4:     Increase $\gamma$ by the smallest amount that is necessary for $|C(\gamma)|$ to increase.
5:     Compute the optimal solution $x$ with respect to the core $C(\gamma)$.
6: **until** $p^{\mathsf{T}}\bar{x} - p^{\mathsf{T}}x \leq \gamma$
7: **return** $x$;
```

Beier and Vöcking [BV06a] have implemented this algorithm. They used the Nemhauser-Ullmann algorithm to compute the optimal solution x with respect to the current core in Line 5. With their implementation random instances with millions of items can be solved optimally within a few seconds.

Chapter 7

Smoothed Complexity of Binary Optimization Problems

We have seen in the previous chapter that the knapsack problem can be solved efficiently on inputs with ϕ-perturbed weights or ϕ-perturbed profits. Instead of studying each problem separately, we will now give a general characterization which combinatorial optimization problems can be solved efficiently on instances with ϕ-perturbed numbers.

We will study ***linear binary optimization problems***. In an instance of such a problem Π, a linear objective function $c^{\mathsf{T}}x = c_1x_1 + \cdots + c_nx_n$ is to be minimized or maximized over an arbitrary set $S \quad \{0, 1\}^n$ of feasible solutions. The problem Π could, for example, be the traveling salesman problem and the coefficients c_i could be the edge lengths. (See also the discussion at the end of Section 6.3 on how graph problems can be encoded as binary optimization problems.) One could also encode the knapsack problem as a linear binary optimization problem. Then S contains all subsets of items whose total weight does not exceed the capacity.

We will study the ***smoothed complexity*** of linear binary optimization problems, by which we mean the complexity of instances in which the coefficients $c_1, \ldots, c_n$ are ϕ-perturbed numbers from the interval $[-1, 1]$. We will assume without loss of generality that the objective function $c^{\mathsf{T}}x$ is to be minimized. Since ϕ-perturbed coefficients are real numbers and do not have a finite encoding with probability 1, Turing machines or random-access machines do not seem to be appropriate machine models to study the smoothed complexity of linear binary optimization problems. One could change the input model and assume that the ϕ-perturbed coefficients are discretized by rounding them after a polynomial number, say n^2, of bits. The effect of this rounding is so small that it does not influence our results. We will, however, not make this assumption explicit and use, for the sake of simplicity, the continuous random variables in our probabilistic analysis. In particular, when defining the input size we will not take the encoding length of the coefficients c_i into account. Instead we assume that the coefficients $c_1, \ldots, c_n$ contribute in total only n to the input length.

To state the main results of this chapter, let us recall two definitions from computational complexity. We call a linear binary optimization problem ***strongly*** NP-***hard*** if it is already NP-hard when restricted to instances with integer coefficients c_i in which the largest absolute value $C := \max_i |c_i|$ of any of the coefficients is bounded by a polynomial in the input length. The TSP is, for example, strongly NP-hard because it is already NP-hard when all edges have length either 1 or 2. The knapsack problem, on the other hand, is not strongly NP-hard because instances in which all profits are integers and polynomially bounded in the input size can be solved by dynamic programming in polynomial time.

A language L belongs to the complexity class ZPP (zero-error probabilistic polynomial time) if there exists a randomized algorithm A that decides for each input x in expected polynomial time whether or not x belongs to L. That is, A always produces the correct answer but its running time is a random variable whose expected value is bounded polynomially for every input x. Let us point out that the expectation is only with respect to the random decisions of the algorithm and not with respect to a randomly chosen input (as usually the case in this part of the lecture). It is yet unclear whether or not P = ZPP. In any case, languages that belong to ZPP are generally considered to be easy to decide.

Theorem 7.1. *Let* Π *be a linear binary optimization problem that is strongly* NP*-hard. Then there does not exist an algorithm for* Π *whose expected running time is polynomially bounded in* N *and* ϕ *for instances with* ϕ*-perturbed coefficients from* $[-1, 1]$*, where* N *denotes the input length, unless* NP   ZPP*.*

Proof. Let A denote an algorithm that solves ϕ-perturbed instances of Π in expected time $p(N, \phi)$ for some polynomial p, where N denotes the input length. Since Π is strongly NP-hard, there exists a polynomial q such that the problem Π is already NP-hard when restricted to instances with integer coefficients whose absolute values are bounded from above by $q(N)$. We claim that we can solve these instances with the help of A in expected polynomial time. Hence we have found an NP-hard optimization problem that belongs to ZPP. This implies NP   ZPP (analogously to the fact that the existence of an NP-hard optimization problem that belongs to P implies NP   P).

Let *I* be an instance of Π with input length N and integer coefficients $c_1, \ldots, c_n$ whose absolute values are bounded from above by $q(N)$. First we scale the coefficients such that they lie in the interval $[-1, 1]$. This does not change the optimal solution. In the following we denote by $\tilde{c}_1, \ldots, \tilde{c}_n$ the scaled coefficients, i.e., $\tilde{c}_i = c_i/C$, where $C := \max_i |c_i| \leq q(N)$. Let $\phi := 2nC$. We choose for each coefficient $\tilde{c}_i$ an interval I_i $[-1, 1]$ of length $1/\phi$ with $\tilde{c}_i$ I_i and choose c_i uniformly at random from I_i. In this way, we obtain an instance of Π with ϕ-perturbed coefficients. Let us call this instance *I* . We can use algorithm A to solve this instance in expected time $p(N, \phi)$. Due to our choice of ϕ and due to $N \geq n$, this expected running time is polynomially bounded in the input length N:

$$p(N, \phi) = p(N, 2nC) \leq p(N, 2nq(N)) \leq p(N, 2Nq(N)) = \text{poly}(N).$$

We will argue that any optimal solution of the ϕ-perturbed instance *I* is also an optimal solution of the instance *I* . Hence, algorithm A solves not only the ϕ-perturbed instance *I* but also the instance *I* in expected polynomial time. Let x denote the optimal solution of instance *I* that algorithm A computes. We will argue that x is also an optimal solution of the instance *I*. Assume for contradiction that there exists a better solution $x^\star$ for instance *I*. Since all coefficients in *I* are integers, it must be $c^{\mathsf{T}}x - c^{\mathsf{T}}x^\star \geq 1$ and hence

$$\tilde{c}^{\mathsf{T}}x - \tilde{c}^{\mathsf{T}}x^\star \geq \frac{1}{C}.$$

Since $|c_i - \tilde{c}_i| \leq 1/\phi$ for all i, this implies

$$\begin{aligned}(c\ )^{\mathsf{T}}x - (c\ )^{\mathsf{T}}x^\star &\geq \tilde{c}^{\mathsf{T}}x - \tilde{c}^{\mathsf{T}}x^\star - \frac{n}{\phi}\\ &= \tilde{c}^{\mathsf{T}}x - \tilde{c}^{\mathsf{T}}x^\star - \frac{1}{2C} \geq \frac{1}{2C} > 0.\end{aligned}$$

Hence, contradicting our choice of x as optimal solution of *I* , the solution $x^\star$ is better than x also with respect to the coefficients $c_1, \ldots, c_n$. □

The previous theorem shows that ϕ-perturbed instances of strongly NP-hard optimization problems are not easier to solve than worst-case instances. Hence, these problems stay hard also in the model of smoothed analysis. One consequence of this result is that there is no hope that the TSP can be solved efficiently when the edge lengths are randomly perturbed. This is a clear contrast to the knapsack problem, which is easy to solve on randomly perturbed inputs, as shown in the previous chapter. We will now prove a more general positive result. We say that a linear binary optimization problem Π can be solved in ***pseudo-linear time*** if there exists an algorithm whose running time on instances with integer coefficients is bounded from above by $p(N) \cdot C$, where p denotes a polynomial, N denotes the input length, and C denotes the largest absolute value of any of the coefficients.

Theorem 7.2. *A linear binary optimization problem Π that can be solved in pseudo-linear time in the worst case can be solved in expected polynomial time (with respect to the input length and ϕ) on instances with ϕ-perturbed numbers.*

Proof. There exists a constant k and an algorithm A_{pseudo} that solves integer instances of Π in time $O(N^k \cdot C)$, where N denotes the input length and C denotes the largest absolute value of any of the coefficients. Instances in which all coefficients are rational numbers from $[-1, 1]$ that can each be encoded with at most b bits after the binary point can be solved by the algorithm A_{pseudo} in time $O(N^k \cdot 2^b)$ because if we scale each coefficient in such an instance by 2^b then all coefficients are integers with absolute value at most 2^b.

We will use algorithm A_{pseudo} as a subroutine to obtain an algorithm that solves ϕ-perturbed instances of Π efficiently. The idea of this algorithm is rather simple: round all coefficients of the ϕ-perturbed instance *I* after a certain number b of bits after the binary point and solve the rounded instance I_b in time $O(N^k \cdot 2^b)$ with algorithm A_{pseudo}. If $b = \Theta(\log n)$ then A_{pseudo} runs in polynomial time and the optimal solutions of the instances *I* and I_b coincide with high probability. In order to make this more precise, we will first of all introduce and analyze the ***winner gap*** Δ.

Given an instance I of Π with a set $S \quad \{0,1\}^n$ of feasible solutions, the winner gap is defined as

$$\Delta = c^\mathsf{T} x^{\star\star} - c^\mathsf{T} x^\star,$$

where

$$x^\star = \arg\min\{c^\mathsf{T} x \mid x \quad S\} \quad \text{and} \quad x^{\star\star} = \arg\min\{c^\mathsf{T} x \mid x \quad S \setminus \{x^\star\}\}$$

denote the best and second best solution of I, respectively. If I contains multiple optimal solutions, then $\Delta = 0$. We will prove that in ϕ-perturbed instances the winner gap is typically not too small.

Lemma 7.3 (Isolation Lemma)**.** *Let I be an instance of Π with ϕ-perturbed coefficients $c_1, \dots, c_n$. Then for every $\varepsilon > 0$,*

$$\mathbf{Pr}\left(\Delta \le \varepsilon\right) \le 2n\phi\varepsilon.$$

Proof. The proof of this lemma is similar to the proof of Lemma 6.8. Hence, we present it in a rather condensed version and refer the reader to the proof of Lemma 6.8 for more verbose explanations.

We first define a set of auxiliary random variables $\Delta_1, \dots, \Delta_n$ with the property that at least one of these auxiliary random variables coincides with Δ. We define

$$x^{i=0} = \arg\min\{c^\mathsf{T} x \mid x \quad S \text{ and } x_i = 0\},$$
$$x^{i=1} = \arg\min\{c^\mathsf{T} x \mid x \quad S \text{ and } x_i = 1\},$$

and $\Delta_i = |c^\mathsf{T} x^{i=0} - c^\mathsf{T} x^{i=1}|$.

Let $i \quad [n]$ be an index in which the best and the second best solution differ, i.e., $x^\star_i = x^{\star\star}_i$. For this index i it holds either $x^{i=0} = x^\star$ and $x^{i=1} = x^{\star\star}$ or $x^{i=1} = x^\star$ and $x^{i=0} = x^{\star\star}$. In any case $\Delta = \Delta_i$.

For any i, we would like to give an upper bound on the probability that $\Delta_i \le \varepsilon$. We use again the principle of deferred decisions and assume that all c_j with $j = i$ are already fixed arbitrarily. Then also the solutions $x^{i=0}$ and $x^{i=1}$ are already fixed as their identity does not depend on the yet unknown coefficient c_i. This implies that $\Delta_i = |\kappa - c_i|$ for some constant κ that depends on the coefficients c_j with $j = i$. Hence,

$$\mathbf{Pr}\left(\Delta_i \le \varepsilon\right) \le \mathbf{Pr}\left(|\kappa - c_i| \le \varepsilon\right) \le 2\phi\varepsilon,$$

where we used that c_i is a random variable whose density is bounded from above by ϕ.

Altogether we obtain

$$\mathbf{Pr}\left(\Delta \le \varepsilon\right) \le \mathbf{Pr}\left(\quad i \quad [n] : \Delta_i \le \varepsilon\right) \le \sum_{i=1}^{n} \mathbf{Pr}\left(\Delta_i \le \varepsilon\right) \le 2n\phi\varepsilon. \qquad \square$$

For a coefficient c_i we denote by $\;c_i\;_b$ and $\;c_i\;_b$ the numbers that are obtained when c_i is rounded down or up after b bits after the binary point, respectively. That is, $\;c_i\;_b \le c_i$

and $\ c_{i\ b} \geq c_i$. It holds $|\ c_{i\ b} - c_i| \leq 2^{-b}$ and $|\ c_{i\ b} - c_i| \leq 2^{-b}$. Let $\ c\ _b$ and $\ c\ _b$ denote the vectors that consist of the rounded coefficients.

For a vector $a \quad \mathsf{R}^n$, let $\mathrm{opt}(a)$ denote the optimal solution of the given instance of Π with objective function $a^{\mathsf{T}}x$, i.e., $\mathrm{opt}(a) = \arg\min\{a^{\mathsf{T}}x \mid x \quad S\}$. The isolation lemma implies the following corollary.

Corollary 7.4. *For any $b \quad \mathsf{N}$, and any vector c with $c_i \quad \{\ c_{i\ b},\ c_{i\ b}\}$ for any $i \quad [n]$,*

$$\mathbf{Pr}\left(\mathrm{opt}(c\ ) = \mathrm{opt}(c)\right) \leq 2^{-b+2}n^2\phi.$$

Proof. For every $i \quad [n]$, $|c_i - c_i| \leq 2^{-b}$. Hence, for any $x \quad \{0,1\}^n$

$$|c^{\mathsf{T}}x - (c\ )^{\mathsf{T}}x| \leq n2^{-b}.$$

Hence, if $\Delta > 2n2^{-b}$, then the optimal solution $x^\star = \mathrm{opt}(c)$ is also optimal with respect to $c\ $. This implies

$$\mathbf{Pr}\left(\mathrm{opt}(c\ ) = \mathrm{opt}(c)\right) \leq \mathbf{Pr}\left(\Delta \leq 2n2^{-b}\right) \leq 2^{-b+2}n^2\phi,$$

where the last inequality follows from Lemma 7.3. □

Corollary 7.4 immediately yields an algorithm that always runs in polynomial time and solves ϕ-perturbed instances of Π with high probability correctly: use the pseudo-linear algorithm A_{pseudo} to compute $\mathrm{opt}(\ c\ _b)$ in time $O(N^k \cdot 2^b)$. For $b = \log(n^3\phi) + 2$ the running time of A_{pseudo} is polynomial and $\mathrm{opt}(c) = \mathrm{opt}(\ c\ _b)$ with probability at least $1 - 1/n$. However, in the following we aim for an algorithm that always computes the optimal solution and whose expected running time is polynomial.

For this, we add another step to the algorithm: After we have computed $\mathrm{opt}(\ c\ _b)$, we test whether or not it is also an optimal solution with respect to the objective function $c^{\mathsf{T}}x$. If this is not the case, then we increase the precision b by one and compute $\mathrm{opt}(\ c\ _b)$ for the increased precision. This is repeated until the solution $\mathrm{opt}(\ c\ _b)$ is also optimal with respect to the objective function $c^{\mathsf{T}}x$. Now the question is of course, how one can efficiently test if the solution $\mathrm{opt}(\ c\ _b)$ is also optimal with respect to the objective function $c^{\mathsf{T}}x$. For this, we use the following method Certify(c,x,b).

```
Certify($c$,$x$,$b$)

For $i \quad [n]$, let $\tilde{c}_i = \begin{cases} \ c_{i\ b} & \text{if } x_i = 0, \\ \ c_{i\ b} & \text{if } x_i = 1. \end{cases}$
Compute $y = \mathrm{opt}(\tilde{c})$ with algorithm $A_{\mathsf{pseudo}}$.
if $x = y$ then
    return true;
else
    return false;
```

Lemma 7.5. *If the method Certify(c,x,b) returns true, then x is an optimal solution with respect to the objective function $c^{\mathsf{T}}x$.*

Proof. For contradiction, assume that Certify(c,x,b) returns true and that there exists a solution z S with $c^\mathsf{T} z < c^\mathsf{T} x$. Let i $[n]$ be an index with $z_i = x_i$. If $z_i = 1$ and $x_i = 0$, then

$$\tilde{c}_i x_i - \tilde{c}_i z_i = c_i x_i - \tilde{c}_i z_i = c_i x_i - \quad c_i \ {}_b z_i \geq c_i x_i - c_i z_i .$$

If $z_i = 0$ and $x_i = 1$, then

$$\tilde{c}_i x_i - \tilde{c}_i z_i = \tilde{c}_i x_i - c_i z_i = \quad c_i \ {}_b x_i - c_i z_i \geq c_i x_i - c_i z_i .$$

Hence, the definition of $\tilde{c}$ guarantees that for every i $[n]$, $\tilde{c}_i x_i - \tilde{c}_i z_i \geq c_i x_i - c_i z_i$. This implies

$$\tilde{c}^\mathsf{T} x - \tilde{c}^\mathsf{T} z \geq c^\mathsf{T} x - c^\mathsf{T} z > 0.$$

Hence, x is not an optimal solution with respect to $\tilde{c}$ and Certify(c,x,b) does not return true. □

Lemma 7.6. *Certify(c,x,b) returns false with probability at most $2^{-b+2} n^2 \phi$.*

Proof. The argument is similar to the argument that we used in the proof of Corollary 7.4. If $\Delta > 2^{-b+1} n$, then opt(c) $=$ opt(c) for any vector c with $c_i = \quad c_i \ {}_b$ or $c_i = \quad c_i \ {}_b$ for any i $[n]$, which implies opt(c $_b$) $=$ opt(c) $=$ opt($\tilde{c}$). According to Lemma 7.3, $\mathbf{Pr}\left(\Delta \leq 2^{-b+1} n\right) \leq 2^{-b+2} n^2 \phi$. □

Based on the method Certify(c,x,b), we can devise the following algorithm Adaptive-Rounding that solves every instance I of Π optimally.

```
AdaptiveRounding( I )
for $b = 1, \dots, n$ do
    Compute $x$ = opt(  $c$ $_b$) with algorithm $A_{\text{pseudo}}$.
    if Certify($c$,$x$,$b$) then return $x$;
Compute $x$ = opt($c$) by brute force.
return $x$;
```

The correctness of this algorithm follows immediately from Lemma 7.5. It remains to analyze its expected running time. Let X denote the random variable that corresponds to the final value of b in the algorithm, i.e., X denotes the number of iterations of the for-loop. Lemma 7.6 implies that

$$\mathbf{Pr}\left(X \geq j\right) \leq 2^{-j+3} n^2 \phi$$

because X can only be at least j if Certify returns false in iteration $j-1$. Furthermore, let t_j denote the total running time of iteration j of this loop. Since in this iteration two calls to algorithm A_{pseudo} are made with coefficients that have j bits each, $t_j = O(N^k \cdot 2^j)$. We assume in the following that the time to solve I by brute force is $O(2^n)$. Depending on the problem, it might actually be $O(2^n)$ times a polynomial in N, but we ignore this additional polynomial for the sake of simplicity because it has no significant effect on the analysis. Let T denote the random variable that describes the total running time of the algorithm. Then

$$\begin{aligned}\mathbf{E}[T] &= \sum_{j=1}^{n} t_j \cdot \mathbf{Pr}(X \geq j) + O(2^n) \cdot \mathbf{Pr}(X \geq n) \\ &\leq \sum_{j=1}^{n} O(N^k \cdot 2^j) \cdot 2^{-j+3} n^2 \phi + O(2^n) \cdot 2^{-n+3} n^2 \phi \\ &= O(N^k n^3 \phi) = O(N^{k+3}\phi).\end{aligned}$$

In summary, the algorithm AdaptiveRounding always computes an optimal solution of the given instance I and its expected running time on ϕ-perturbed instances is polynomially bounded in N and ϕ. □

The results that we presented in this section can be found in [BV06b]. Theorems 7.1 and 7.2 do not give a complete characterization of the smoothed complexity of linear binary optimization problems. They leave open the smoothed complexity of problems that can be solved in pseudo-polynomial but not pseudo-linear time in the worst case. One can show that also these problems can be solved in expected polynomial time on ϕ-perturbed instances [RT09], which completes the characterization.

Chapter 8

Successive Shortest Path Algorithm

We study the ***Successive Shortest Path (SSP) algorithm*** for the ***minimum-cost flow problem***. This chapter is based on an article by Brunsch et al. [BCMR13].

Let us first define the minimum-cost flow problem. A ***flow network*** is a simple directed graph $G = (V, E)$ together with a ***capacity function*** $u\colon E \to \mathsf{R}_{\geq 0}$, a source $s \quad V$, and a sink $t \quad V$. For convenience, we assume that there are no directed cycles of length two. In the minimum-cost flow problem there is an additional ***cost function*** $c\colon E \to [0, 1]$ given. A ***flow*** for such an instance is a function $f\colon E \to \mathsf{R}_{\geq 0}$ that obeys the capacity constraints $0 \leq f(e) \leq u(e)$ for all edges $e \quad E$ and Kirchhoff's law, i.e., $\sum_{e=(u,v)\ \ E} f(e) = \sum_{e\ =(v,w)\ \ E} f(e\ )$ for all nodes $v \quad V \setminus \{s, t\}$. A ***maximum flow*** f is a flow with maximum ***value***

$$|f| = \sum_{e=(s,v)\ \ E} f(e) - \sum_{e=(v,s)\ \ E} f(e).$$

The cost of a flow is defined as $c(f) = \sum_{e\ \ E} f(e) \cdot c(e)$. In the ***minimum-cost flow problem*** the goal is to find a cheapest maximum flow, a so-called ***minimum-cost flow***.

The SSP algorithm is a simple greedy algorithm to solve the minimum-cost flow problem, which is known to have an exponential worst-case running time [Zad73]. An experimental study of Kovács [Kov15] shows, however, that the SSP algorithm performs well in practice and is much faster than most polynomial-time algorithms for the minimum-cost flow problem. We will explain why the SSP algorithm comes off so well by applying the framework of smoothed analysis.

8.1 The SSP Algorithm

Let $G = (V, E)$ be a flow network. For a pair $e = (u, v)$, we denote by e^{-1} the pair (v, u). Observe that $e \quad E$ implies $e^{-1}\ / \ E$ because we assume that there are no cycles of length two. For a flow f, the ***residual network*** G_f is the directed graph with vertex set V, arc set $E\ = E_{\mathsf{f}} \quad E_{\mathsf{b}}$, where

$$E_{\mathsf{f}} = \Big\{ e \,|\, e \quad E \text{ and } f(e) < u(e) \Big\}$$

is the set of so-called ***forward arcs*** and

$$E_{\mathsf{b}} = \left\{ e^{-1} \,|\, e \quad E \text{ and } f(e) > 0 \right\}$$

is the set of so-called ***backward arcs***, a capacity function $u : E \;\; \to \mathsf{R}$, defined by

$$u\ (e) = \begin{cases} u(e) - f(e) & \text{if } e \quad E\,, \\ f(e^{-1}) & \text{if } e^{-1} \quad E\,, \end{cases}$$

and a cost function $c : E \;\; \to \mathsf{R}$, defined by

$$c\ (e) = \begin{cases} c(e) & \text{if } e \quad E\,, \\ -c(e^{-1}) & \text{if } e^{-1} \quad E\,. \end{cases}$$

Let f be a flow, let G_f denote the corresponding residual network, and let P denote a path from s to t in G_f. We say that the flow f is obtained from the flow f ***by augmenting*** $\delta \geq 0$ ***units of flow along path*** P if for all $e \quad E$

$$f\ (e) = \begin{cases} f(e) + \delta & \text{if } e \quad P \cap E_f\,, \\ f(e) - \delta & \text{if } e^{-1} \quad P \cap E_b\,, \\ f(e) & \text{otherwise}\,. \end{cases}$$

In the SSP algorithm, which is given in the following pseudocode, we choose δ as large as possible subject to the constraint that f is a flow (i.e., $0 \leq f\ (e) \leq u(e)$ for all $e \quad E$).

```
Successive Shortest Path Algorithm
start with the empty flow $f_0 = 0$
while $G_{f_i}$ contains an $s$-$t$ path do
    find a shortest $s$-$t$ path $P_i$ in $G_{f_i}$ with respect to the arc costs
    augment the flow as much as possible along path $P_i$ to obtain a new flow $f_{i+1}$
```

8.1.1 Elementary Properties

In this section, we summarize a few elementary properties of the SSP algorithm that we will need in our analysis.

Theorem 8.1 ([KV18]). *In any round i, flow f_i is a flow with minimum cost among all flows with value $|f_i|$. In particular, the last flow obtained by the SSP algorithm is a minimum-cost flow.*

As a consequence, no residual network G_{f_i} contains a directed cycle with negative total costs. Otherwise, we could augment along such a cycle to obtain a flow with the same value as f_i but smaller costs. In particular, this implies that the shortest paths in G_{f_i} from s to nodes $v \quad V$ form a shortest path tree rooted at s.

Next, we observe that for any node v the distances in the residual network from s to v and from v to t are monotonically increasing.

Lemma 8.2. *Let $d_i(v)$ denote the distance from s to node v in the residual network G_{f_i} and let $d_i(v)$ denote the distance from node v to t in the residual network G_{f_i}. Then the sequences $d_0(v)$, $d_1(v)$, $d_2(v)$, ... and $d_0(v)$, $d_1(v)$, $d_2(v)$, ... are monotonically increasing for every $v \quad V$.*

Proof. We only prove the statement for the sequence $d_0(v), d_1(v), d_2(v), \ldots$. The statement for the sequence $d_0(v), d_1(v), d_2(v), \ldots$ can be shown analogously. Let $i \geq 0$ be an arbitrary integer. We show $d_i(v) \leq d_{i+1}(v)$ by induction on the depth of node v in the shortest path tree T_{i+1} of the residual network $G_{f_{i+1}}$ rooted at s. For the root s, the claim holds since $d_i(s) = d_{i+1}(s) = 0$. Now assume that the claim holds for all nodes up to a certain depth k, consider a node v with depth $k+1$, and let u denote its parent. Consequently, $d_{i+1}(v) = d_{i+1}(u) + c(e)$ for $e = (u, v)$. If arc e has been available in G_{f_i}, then $d_i(v) \leq d_i(u) + c_e$. If not, then the SSP algorithm must have augmented along e^{-1} in step $i+1$ to obtain flow f_{i+1} and, hence, $d_i(u) = d_i(v) + c(e^{-1}) = d_i(v) - c(e)$ (because e^{-1} is part of the shortest-path-tree in G_{f_i}). In both cases the inequality $d_i(v) \leq d_i(u) + c(e)$ holds. By the induction hypothesis for node u, we obtain $d_i(v) \leq d_i(u) + c(e) \leq d_{i+1}(u) + c(e) = d_{i+1}(v)$. □

Lemma 8.2 implies in particular that the distance from the source s to the sink t is monotonically increasing, which yields the following corollary.

Corollary 8.3. *Let $i < j$, let P_i denote the shortest path in G_{f_i}, and let P_j denote the shortest path in G_{f_j}. Then $c(P_i) \leq c(P_j)$.*

8.2 Smoothed Analysis

In the following, we consider instances of the minimum-cost flow problem in which the costs are ϕ-perturbed numbers from $[0, 1]$. That is, the adversary specifies for each edge e a probability density function $f_e \colon [0, 1] \to [0, \phi]$ according to which the cost $c(e)$ is randomly drawn independently of the other edge costs. Furthermore, the adversary chooses the network graph $G = (V, E)$ and the edge capacities. The edge capacities are even allowed to be real values. Let $n = |V|$ and $m = |E|$. We define the smoothed running time of the SSP algorithm as the worst expected running time the adversary can achieve and we prove the following theorem.

Theorem 8.4. *The SSP algorithm requires $O(mn\phi)$ augmentation steps in expectation and its smoothed running time is $O(mn\phi(m + n \log n))$.*

This theorem can be seen as an explanation for why it is unlikely to encounter instances on which the SSP algorithm requires exponentially many steps. The following theorem, which we will not prove in this lecture, shows that the upper bound given in the previous theorem is asymptotically tight for $\phi = \Omega(n)$.

Theorem 8.5 ([BCMR13]). *For given positive integers n, $m \quad \{n, \ldots, n^2\}$, and $\phi \leq 2^n$ there exists a minimum-cost flow network with $O(n)$ nodes, $O(m)$ edges, and ϕ-perturbed edge costs on which the SSP algorithm requires $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$ augmentation steps with probability 1.*

8.2.1 Terminology and Notation

Consider the execution of the SSP algorithm on the flow network G. We denote the set $\{f_0, f_1, \dots, f_N\}$ of all flows encountered by the SSP algorithm by $\mathsf{F}(G)$. Furthermore, we denote by $\mathsf{P}(G) = \{P_0, P_1, \dots, P_{N-1}\}$ the paths that the SSP algorithm augments along, i.e., P_i is a shortest path in the residual network G_{f_i}. We omit the parameter G if it is clear from the context and we say that f_i *induces* P_i. In the following, we will determine the expected size of $\mathsf{P}(G)$ because this equals the expected number of augmentation steps of the SSP algorithm.

To distinguish between the original network G and some residual network G_f, we refer to the edges in the residual network as *arcs*, whereas we refer to the edges in the original network as *edges*. For a given arc e in a residual network G_f, we denote by e_0 the corresponding edge in the original network G, i.e., $e_0 = e$ if $e \quad E$ (i.e., e is a forward arc) and $e_0 = e^{-1}$ if $e\ /\ E$ (i.e., e is a backward arc). An arc e is called *empty* in a flow f if e belongs to G_f, but e^{-1} does not. Empty arcs e are either forward arcs that do not carry flow or backward arcs whose corresponding edge e_0 carries as much flow as possible. We say that an arc *becomes saturated* (during an augmentation) when it is contained in the current augmenting path, but it does not belong to the residual network that we obtain after this augmentation.

In the remainder, a *path* is always a simple directed path. Let P be a path, and let u and v be contained in P in this order. By $u \overset{P}{\quad} v$, we refer to the sub-path of P starting from node u going to node v, by $\overleftarrow{P}$ we refer to the path we obtain by reversing the direction of each arc of P. We denote by

$$c(P) = \sum_{e \quad P \cap E_f} c(e) - \sum_{e \quad P \cap E_b} c(e^{-1})$$

the cost of P. We will also refer to $c(P)$ as the *length* of the path P. We call any flow network G a *possible residual network* of G if there is a flow f for G such that $G = G_f$. Paths in possible residual networks are called *possible paths*.

8.2.2 Proof of the Upper Bound

The idea and structure of the proof of Theorem 8.4 are very similar to the proof of Theorem 6.5. Since all edges have costs from $[0, 1]$, the cost $c(P)$ of any path $P \quad \mathsf{P}$ lies in the interval $[0, n]$. Similarly in the proof of Theorem 6.5, every Pareto-optimal solution from P is mapped to some number from $[0, n]$ by the weight function $w^\mathsf{T} x$. The idea to count the number of Pareto-optimal solutions was to divide the interval $[0, n]$ into small subintervals of length n/k for some k. We argued that if k is large enough, then with high probability there are no two Pareto-optimal solutions that are mapped to the same subinterval. Hence, in order to count the number of Pareto-optimal solutions, it suffices to count the number of non-empty subintervals. The approach to prove Theorem 8.4 is the same, except that we do not count Pareto-optimal solutions but paths encountered by the SSP algorithm. The main difference is the analysis of the probability that a given subinterval contains a path from P, which is more challenging than its counterpart in the proof of Theorem 6.5.

For $k \quad \mathsf{N}$, let F_k denote the event that there exist two nodes $u \quad V$ and $v \quad V$ and two distinct possible u-v paths whose lengths differ by at most n/k.

Lemma 8.6. *For every* $k \quad \mathsf{N}$, $\mathbf{Pr}(\mathsf{F}_k) \leq \frac{2n^{2n+1}\phi}{k}$.

Proof. Fix two nodes u and v and two distinct possible u-v paths P_1 and P_2. Then there is an edge e such that one of the paths – without loss of generality path P_1 – contains arc e or e^{-1}, but the other one contains neither of them. If we use the principle of deferred decisions and fix all edge costs except for the cost of edge e, then the length of P_2 is already determined whereas the length of P_1 depends linearly on the cost $c(e)$. Hence, $c(e)$ must fall into a fixed interval of length $2n/k$ in order for the path lengths of P_1 and P_2 to differ by at most n/k. The probability for this is bounded by $2n\phi/k$ because $c(e)$ is chosen according to a density function that is bounded from above by ϕ. There are at most n^n different simple possible paths. A union bound over all pairs of such paths concludes the proof. □

The following lemma, which we prove further below, is the crucial ingredient in the proof of Theorem 8.4.

Lemma 8.7. *For every* $d \geq 0$ *and every* $\varepsilon > 0$,

$$\mathbf{Pr}(\quad P \quad \mathsf{P} \mid c(P) \quad (d, d+\varepsilon]) \leq 2m\varepsilon\phi.$$

Now the proof of Theorem 8.4 follows along the same lines as the proof of Theorem 6.5. We partition the interval $(0, n]$ uniformly into $k \quad \mathsf{N}$ intervals $I_0^k, \ldots, I_{k-1}^k$ for some large number k to be chosen later. Formally, let $I_i^k = (ni/k, n(i+1)/k]$.

We denote by X^k the number of intervals I_i^k for which there exists at least one path P P with $c(P) \quad I_i^k$. If the event F_k does not occur, then $|\mathsf{P}| = X^k$. (Observe that with probability 1 there does not exist a possible s-t path of length 0).

Lemma 8.8. *For every* $k \quad \mathsf{N}$, $\mathbf{E}\left[X^k\right] \leq 2mn\phi$.

Proof. Let X_i^k denote a random variable that is 1 if there exists a path $P \quad \mathsf{P}$ with $c(P) \quad I_i^k$ and 0 otherwise. Then

$$X^k = \sum_{i=0}^{k-1} X_i^k$$

and by linearity of expectation

$$\mathbf{E}\left[X^k\right] = \mathbf{E}\left[\sum_{i=0}^{k-1} X_i^k\right] = \sum_{i=0}^{k-1} \mathbf{E}\left[X_i^k\right] = \sum_{i=0}^{k-1} \mathbf{Pr}\left(\quad P \quad \mathsf{P} \mid c(P) \quad I_i^k\right). \tag{8.1}$$

Lemma 8.7 and (8.1) imply

$$\mathbf{E}\left[X^k\right] = \sum_{i=0}^{k-1} \mathbf{Pr}\left(\quad P \quad \mathsf{P} \mid c(P) \quad I_i^k\right) \leq \sum_{i=0}^{k-1} \frac{2mn\phi}{k} = 2mn\phi.$$ □

The last missing ingredient in the proof of Theorem 8.4 is a bound on the number of augmentation steps of the SSP algorithm in the worst case. While in Chapter 6 it was clear that there can be at most 2^n Pareto-optimal solutions, the situation is slightly more complicated here. Lemma 8.6 shows in particular (in the limit for $k \to \infty$) that any two different possible paths have different lengths with probability 1. Hence, we will assume throughout this chapter that the following property holds.

Property 8.9. *For all nodes u and v the lengths of all possible u-v paths are pairwise distinct.*

Assuming this property, we can bound the number of augmentation steps of the SSP algorithm in the worst case.

Lemma 8.10. *The number $|P|$ of augmentation steps of the SSP algorithm is bounded from above by 3^m.*

Proof. Consider two paths $P_i \quad P$ and $P_{i+1} \quad P$ encountered by the SSP algorithm. Since at least one arc of P_i is not contained in the residual network $G_{f_{i+1}}$, the paths P_i and P_{i+1} must be different. Now Corollary 8.3 and Property 8.9 imply $c(P_i) < c(P_{i+1})$. Hence the lengths of the augmenting paths that the SSP algorithm encounters are strictly increasing. In particular, no path is encountered more than once by the SSP algorithm.

We say that two residual networks are equivalent if they contain the same arcs (possibly with different capacities). Since the shortest path in two equivalent residual networks is the same, at most one residual network from each equivalence class is encountered by the SSP algorithm. The number of equivalence classes is bounded by 3^m because for every edge $e \quad E$ there are three possibilities: either e and e^{-1} are both contained in the residual network or exactly one of them. This completes the proof. □

Proof of Theorem 8.4. The theorem is implied by the following calculation:

$$\begin{aligned}
\mathbf{E}[|P|] &= \sum_{i=0}^{3^m} \Big(i \cdot \mathbf{Pr}(|P| = i)\Big) \\
&= \sum_{i=0}^{3^m} \Big(i \cdot \mathbf{Pr}(|P| = i \quad F_k) + i \cdot \mathbf{Pr}(|P| = i \quad \neg F_k)\Big) \\
&= \sum_{i=0}^{3^m} \Big(i \cdot \mathbf{Pr}(F_k) \cdot \mathbf{Pr}(|P| = i \mid F_k)\Big) + \sum_{i=0}^{3^m} \Big(i \cdot \mathbf{Pr}\Big(X^k = i \quad \neg F_k\Big)\Big) \\
&\leq \mathbf{Pr}(F_k) \cdot \sum_{i=0}^{3^m} \Big(i \cdot \mathbf{Pr}(|P| = i \mid F_k)\Big) + \sum_{i=0}^{3^m} \Big(i \cdot \mathbf{Pr}\Big(X^k = i\Big)\Big) \\
&\leq \frac{2n^{2n+1}\phi}{k} \cdot \sum_{i=0}^{3^m} \Big(i \cdot \mathbf{Pr}(|P| = i \mid F_k)\Big) + \mathbf{E}\Big[X^k\Big] \\
&\leq \frac{2n^{2n+1}3^m\phi}{k} + 2mn\phi. \qquad (8.2)
\end{aligned}$$

For a justification of the steps in this calculation, see the proof of Theorem 6.5.

Since (8.2) holds for every k N, it must be $\mathbf{E}\,[|P|] \leq 2mn\phi$, which proves the first part of the theorem. Since each step of the SSP algorithm runs in time $O(m+n\log n)$ using Dijkstra's algorithm (see, e.g., Korte [KV18] for details), also the statement about the running time of the SSP algorithm follows. □

8.2.3 Further Properties of the SSP Algorithm

We will now introduce further properties of the SSP algorithm that we will need in the proof of Lemma 8.7.

By C we denote the function $C : [0, |f_N|] \to \mathsf{R}_{\geq 0}$ that maps any x $[0, |f_N|]$ to the cost of the cheapest flow f with value x, i.e., $C(x) = \min\{c(f) \mid |f| = x\}$, where the minimum is taken over all flows f with value x.

Lemma 8.11. *The function C is continuous, monotonically increasing, and piecewise linear. The break points of C are the values of the flows f_i $\mathsf{F} \setminus \{f_0, f_N\}$ (assuming Property 8.9).*

Proof. The proof follows from Theorem 8.1 and the observation that the cost of the flow is linearly increasing when gradually increasing the flow along the shortest path in the residual network until at least one arc becomes saturated. The slope of the cost function is given by the length of that path. □

Example 8.12. *Consider the flow network depicted in Figure 8.1. The cost $c(e)$ and the capacity $u(e)$ of an edge e are given by the notation $c(e), u(e)$. For each step of the SSP algorithm, Figure 8.3 lists the relevant part of the augmenting path (excluding s, s , t , and t), its length, the amount of flow that is sent along that path, and the arcs that become saturated. As can be seen in the table, the values $|f|$ of the encountered flows f F are 0, 2, 3, 5, 7, 10, and 12. Except for 0 and 12, these are the breakpoints of the cost function C, and the lengths of the augmenting paths equal the slopes of C (see Figure 8.2).*

Another important property that we will use in the proof of Lemma 8.7 is the following.

Lemma 8.13. *In any step of the SSP algorithm, any s-t path in the residual network contains at least one empty arc.*

Proof. The claim is true for the empty flow f_0. Now consider a flow f_{i+1} $\mathsf{F} \setminus \{f_0\}$, its predecessor flow f_i, the path P_i, which is a shortest path in the residual network G_{f_i}, and an arbitrary s-t path P in the current residual network $G_{f_{i+1}}$. The paths P_i and P are different because at least one arc of P_i is not contained in $G_{f_{i+1}}$ anymore. We show that at least one arc in P is empty.

For this, fix one arc $e = (x, y)$ from P_i that is not contained in the current residual network $G_{f_{i+1}}$ since it became saturated by the augmentation along P_i. Let v be the first node of P that occurs in the sub-path $y \overset{P_i}{\quad} t$ of P_i, and let u be the last node in the sub-path $s \overset{P}{\quad} v$ of P that belongs to the sub-path $s \overset{P_i}{\quad} x$ of P_i (see Figure 8.4).

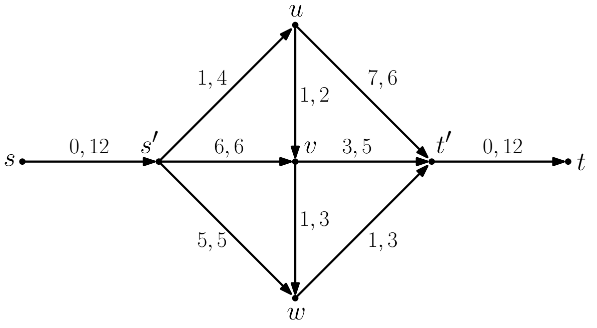


Figure 8.1: Minimum-cost flow network with source s and sink t.

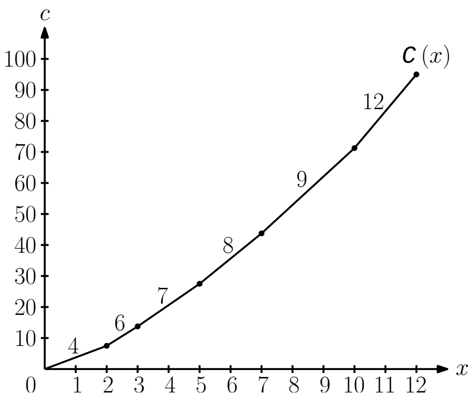


Figure 8.2: Cost function **C** .

step	1	2	3	4	5	6
path	u, v, w	w	w, v	u	v	v, u
path length	4	6	7	8	9	12
amount of flow	2	1	2	2	3	2
saturated arcs	(u, v)	$(w, t\ )$	(w, v)	$(s\ , u)$	$(v, t\ )$	(v, u)

Figure 8.3: The augmenting paths for Example 8.12.

By the choice of u and v, all nodes on the sub-path $P\ = u\ ^{P}\ v$ of P except for u and v do not belong to P_i. Hence, the arcs of P are also available in the residual network G_{f_i} and have the same capacity in both residual networks G_{f_i} and $G_{f_{i+1}}$.

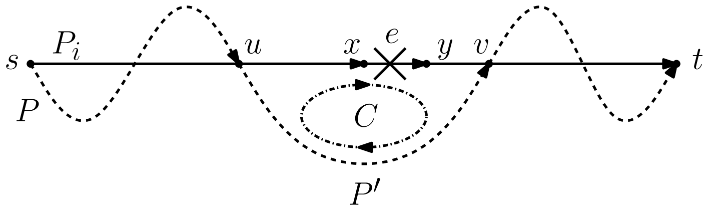


Figure 8.4: Paths P and P_i in the residual network G_{f_i}.

In the remainder of this proof, we show that at least one arc of P is empty. Assume to the contrary that none of the arcs is empty in $G_{f_{i+1}}$ and, hence, in G_{f_i}. This implies that, for each arc $a \quad P$, the residual network G_{f_i} also contains the arc a^{-1}. Since P_i is the shortest s-t path in G_{f_i} and since the lengths of all possible s-t paths are pairwise distinct, the path $s\ ^{P_i}\ u\ ^{P}\ v\ ^{P_i}\ t$ is longer than P_i. Consequently, the path $P\ = u\ ^{P}\ v$ is longer than the path $u\ ^{P_i}\ v$. This contradicts the fact that flow f_{i-1} is optimal since the arcs of path $u\ ^{P_i}\ v$ and the reverse arcs a^{-1} of the arcs a of path P form a directed cycle C in G_{f_i} of negative costs. □

We also need to show that for each flow value the minimum-cost flow is unique with probability 1.

Lemma 8.14. *For any $\varepsilon > 0$ the probability that there exists a possible residual network with a cycle whose cost lies in $[0, \varepsilon]$ is bounded from above by $2n^{2n}\varepsilon\phi$.*

Proof. Assume that in some possible residual network G_f there exists a cycle K whose cost lies in $[0, \varepsilon]$. Then K contains two nodes u and v and consists of a u-v path P_1 and a v-u path P_2. Then P_1 and $\overleftarrow{P}_2$ are two distinct u-v paths. Since K has costs in $[0, \varepsilon]$, the costs of P_1 and $\overleftarrow{P}_2$ differ by at most ε. Now an analogous argument as in the proof of Lemma 8.6 concludes the proof. □

We can assume that the following property holds because it holds with a probability of 1. This follows from Lemma 8.14 by considering the limit for $\varepsilon \to 0$.

Property 8.15. *There does not exist a residual network that contains a cycle of cost 0.*

With Property 8.15 we can show that the minimum-cost flow is unique for each value.

Lemma 8.16. *For each value $B \quad \mathsf{R}_{\geq 0}$ there either exists no flow f with $|f| = B$ or there exists a unique minimum-cost flow f with $|f| = B$.*

Proof. Assume that there exists a value $B \quad \mathsf{R}_{\geq 0}$ and two distinct minimum-cost flows f and $f\ $ with $|f| = |f\ | = B$. Let $E_\Delta := \{e \quad E \mid f(e) = f\ (e)\}$ be the set of edges on which f and $f\ $ differ. We show in the following that the set E_Δ contains at least one undirected cycle K. Since f and $f\ $ are distinct flows, the set E_Δ cannot be empty. For $v \quad V$, let us denote by $f_-(v) = \sum_{e=(u,v) \quad E} f(e)$ the flow entering v and by $f_+(v) = \sum_{e=(v,w) \quad E} f(e)$ the flow going out of v ($f_-(v)$ and $f_+(v)$ are defined analogously). Flow conservation and $|f| = |f\ |$ imply $f_-(v) - f_-(v) = f_+(v) - f_+(v)$ for all $v \quad V$. Now let us assume E_Δ does not contain an undirected cycle. In this case there must exist a vertex $v \quad V$ with exactly one incident edge in E_Δ. We will show that this cannot happen.

Assume $f_-(v) - f_-(v) = 0$ for some $v \quad V$. Then the flows f and $f\ $ differ on at least one edge $e = (u, v) \quad E$. Since this case implies $f_+(v) - f_+(v) = 0$, they also differ on at least one edge $e\ = (v, w) \quad E$ and both these edges belong to E_Δ. It remains to consider nodes $v \quad V$ with $f_-(v) - f_-(v) = f_+(v) - f_+(v) = 0$ and at least one incident edge in E_Δ. For such a node v there exists an edge $e = (u, v) \quad E$ (or $e = (v, w) \quad E$) with $f(e) = f\ (e)$. It follows $\sum_{e\ =(u\ ,v) \quad E, e\ =e} f(e\ ) - f\ (e\ ) = 0$ (or $\sum_{e\ =(v,w\ ) \quad E, e\ =e} f(e\ ) - f\ (e\ ) = 0$) which implies that there exists another edge $e\ = (u\ , v) = e$ (or $e\ = (v, w\ ) = e$) with $f(e\ ) = f\ (e\ )$. Hence every node $v \quad V$ has either no incident edge from E_Δ or at least 2. This implies that E_Δ must contain an undirected cycle K.

For the flow $f\ \ = \frac{1}{2}f + \frac{1}{2}f\ $, which has the same costs as f and $f\ $ and is hence a minimum-cost flow with $|f\ \ | = B$ as well, we have $f\ \ (e) \quad (0, u(e))$ for all $e \quad E_\Delta$. The flow $f\ $ can therefore be augmented in both directions along K. Due to Property 8.15, augmenting $f\ $ in one of the two directions along K will result in a better flow. This is a contradiction. □

8.2.4 Proof of Lemma 8.7

Lemma 8.17. *Let $d \geq 0$, let P $\mathsf{P}(G)$ be the first path encountered by the SSP algorithm with $c(P) > d$, and let f F denote the flow that induces P (i.e., P is a shortest path in the residual network G_f). Additionally, let e be an empty arc on P and let e_0 be the edge in G that corresponds to e. Now change the cost of e_0 to $c\,(e_0) = 1$ if $e_0 = e$ and to $c\,(e_0) = 0$ if $e_0 = e^{-1}$. In any case, the cost of arc e increases. We denote the resulting flow network by G (i.e., G and G coincide except for the cost of edge e_0). Then f $\mathsf{F}(G\ )$ and f is also on the network G the first flow encountered by the SSP algorithm whose induced path P has length strictly larger than d.*

Additionally let, Q $\mathsf{P}(G)$ and Q $\mathsf{P}(G\ )$ denote the paths that the SSP algorithm encounters directly before the paths P and P, respectively. If Q and Q exist, then $c(Q\ ) \leq c(Q)$.

Proof. Let C and C be the cost functions of the original network G and the modified network G, respectively. Both functions are of the form described in Lemma 8.11. We analyze the cases $e_0 = e$ and $e_0 = e^{-1}$ separately. For both cases, we introduce a function C with $\mathsf{C} \quad \geq \mathsf{C}$ and $\mathsf{C} \quad (|f|) = \mathsf{C}\,(|f|)$.

We start with analyzing the case $e_0 = e$. In this case, we set $\mathsf{C} \quad = \mathsf{C}$ and observe that by increasing the cost of edge e_0 to 1 the cost of no flow can decrease. Hence, $\mathsf{C} \quad \geq \mathsf{C}$. Since flow f does not use arc e, its cost remains unchanged, i.e., $\mathsf{C} \quad (|f|) = \mathsf{C}\,(|f|)$.

If $e_0 = e^{-1}$, then we set $\mathsf{C} \quad = \mathsf{C} \ + \Delta(e_0)$ for $\Delta(e_0) = u(e_0) \cdot c(e_0)$. This function is also piecewise linear and has the same breakpoints and slopes as C . Since the flow on edge e_0 cannot exceed the capacity $u(e_0)$ of edge e_0 and since the cost on that edge has been reduced by $c(e_0)$ in G compared to G, the cost of each flow is reduced by at most $\Delta(e_0)$ in G . Furthermore, this gain is only achieved for flows that entirely use edge e_0 like f does. Hence, $\mathsf{C} \quad \geq \mathsf{C}$ and $\mathsf{C} \quad (|f|) = \mathsf{C}\,(|f|)$.

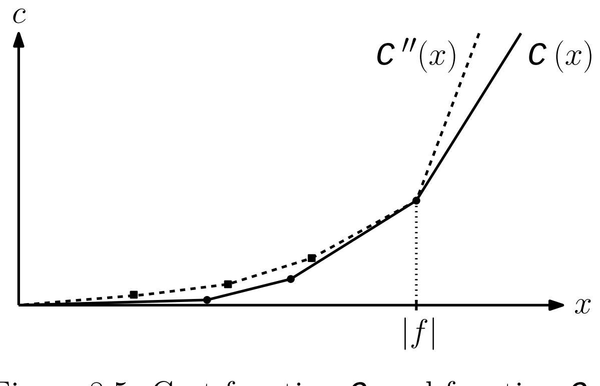


Figure 8.5: Cost function C and function C .

Due to $\mathsf{C} \quad \geq \mathsf{C}$, $\mathsf{C} \quad (|f|) = \mathsf{C}\,(|f|)$, and the form of both functions, the left-hand derivative of C at $|f|$ is at most the left-hand derivative of C at $|f|$. Similarly, the right-hand derivative of C at $|f|$ is at least the right-hand derivative of C at $|f|$ (see Figure 8.5). Since $|f|$ is a breakpoint of C, this implies that $|f|$ is also a breakpoint of C . These properties carry over to C (which either equals C or is a shifted copy of C). Hence, Lemma 8.16 implies $f \quad \mathsf{F}(G\,)$.

By the choice of f, the slope of C to the left of $|f|$ is at most d while the slope of C to the right of $|f|$ is larger than d. By the previous discussion about the derivatives of C and C , the same is true for C . This implies that f is the first flow in $\mathsf{F}(G\,)$ whose induced path has length strictly larger than d. Since the left-hand derivative of C at $|f|$ is at most the left-hand derivative of C at $|f|$, it follows that $c(Q\,) \leq c(Q)$. □

Lemma 8.17 suggests that we can identify the first path in P that is longer than d without any information about the value of $c(e)$, where e is some empty arc on P. We formalize this in the following algorithm Reconstruct, which gets as input an empty arc e of P and a threshold d. The crucial fact that we will later exploit is that for this reconstruction the cost $c(e_0)$ of edge e_0 does not have to be known. (Note that we need Reconstruct only for the analysis.)

```
Reconstruct$(e, d)$.
let $e_0$ be the edge that corresponds to arc $e$ in the original network $G$
change the cost of edge $e_0$ to $c\,(e_0) = 1$ if $e$ is a forward arc or to $c\,(e_0) = 0$ if $e$ is
  a backward arc
start running the SSP algorithm on the modified network $G$
stop when the length of the shortest $s$-$t$ path in the residual network of the current
  flow $f$ exceeds $d$
output the shortest $s$-$t$-path in $G_f$ that uses arc $e$ (if such a path exists)
```

Corollary 8.18. *Let $P \quad$ P, let e be an empty arc on P, and let $d \quad \big[c(Q), c(P)\big)$, where Q denotes the path that the SSP algorithm encounters directly before P. If no such path Q exists, let $c(Q) = 0$. Then* Reconstruct(e, d) *outputs path P.*

Proof. Let $f \quad$ F denote the flow that induces P. By Lemma 8.17, we obtain that $f \quad \mathsf{F}(G\,)$ and that f is the first flow encountered by the SSP algorithm whose induced path has length greater than d. This implies that Reconstruct(e, d) does not stop before encountering flow f and stops once it encounters f. Since P is the shortest path in G_f and contains arc e, Reconstruct(e, d) outputs P. □

Proof of Lemma 8.7. Let P denote the first path encountered by the SSP algorithm whose length is larger than d. If no such path exists, then let $P \; = \;$. Then the event that there exists at least one path $P \quad$ P with $c(P) \quad (d, d+\varepsilon]$ is equivalent to the event that $c(P\,) \quad (d, d+\varepsilon]$. We denote by P_e the path returned by Reconstruct(e, d). Let $E^{-1} = \{e^{-1} \mid e \quad E\}$. By Lemma 8.13, the path P contains an empty arc e . By Corollary 8.18, $P_e \; = P$ and hence,

$$\begin{aligned}\mathbf{Pr}\,(\; P \quad \mathsf{P} \mid c(P) \quad (d, d+\varepsilon]) &= \mathbf{Pr}\,(c(P\,) \quad (d, d+\varepsilon]) \\ &= \mathbf{Pr}\,(c(P_e\,) \quad (d, d+\varepsilon]) \\ &\leq \mathbf{Pr}\Big(\; e \quad E \quad E^{-1} \mid c(P_e) \quad (d, d+\varepsilon]\Big)\end{aligned}$$

$$\leq \sum_{e \quad E \quad E^{-1}} \mathbf{Pr}\left(c(P_e) \quad (d, d+\varepsilon]\right). \tag{8.3}$$

Hence, it remains to analyze the probability of the event $c(P_e) \quad (d, d+\varepsilon]$ for $e \quad E \quad E^{-1}$. For this we use the principle of deferred decisions. It is crucial that P_e is independent of $c(e_0)$, where $e_0 \quad E$ denotes the edge corresponding to the arc e. Hence, if all costs $c(e\ )$ with $e\ = e_0$ are fixed, the cost of P_e can be written as either $\kappa + c(e_0)$ (if e is a forward arc) or $\kappa - c(e_0)$ (if e is a backward arc) for some constant κ that depends on the fixed values of the $c(e\ )$ with $e\ = e_0$. Since $c(e_0)$ is a random variable whose density is bounded from above by ϕ, we obtain $\mathbf{Pr}\left(c(P_e) \quad (d, d+\varepsilon]\right) \leq \varepsilon\phi$. Together with (8.3) this implies

$$\mathbf{Pr}\left(\quad P \quad \mathsf{P} \mid c(P) \quad (d, d+\varepsilon]\right) \leq 2m\phi\varepsilon. \qquad \square$$

Chapter 9

The 2-Opt Algorithm for the TSP

An instance of the *traveling salesman problem (TSP)* consists of a set $V = \{v_1, \ldots, v_n\}$ of *vertices* (depending on the context, synonymously referred to as *points*) and a symmetric *distance function* $\mathrm{dist}\colon V \times V \to \mathbb{R}_{\geq 0}$ that assigns to each pair $\{v_i, v_j\}$ of distinct vertices a distance $\mathrm{dist}(v_i, v_j) = \mathrm{dist}(v_j, v_i)$. The goal is to find a Hamiltonian cycle (i.e., a cycle that visits every vertex exactly once) of minimum length. We also use the term *tour* to denote a Hamiltonian cycle.

A pair (V, dist) of a nonempty set V and a function $\mathrm{dist}\colon V \times V \to \mathbb{R}_{\geq 0}$ is called a *metric space* if for all $x, y, z \in V$ the following properties are satisfied:

(a) $\mathrm{dist}(x, y) = 0$ if and only if $x = y$ *(reflexivity)*,

(b) $\mathrm{dist}(x, y) = \mathrm{dist}(y, x)$ *(symmetry)*,

(c) $\mathrm{dist}(x, z) \leq \mathrm{dist}(x, y) + \mathrm{dist}(y, z)$ *(triangle inequality)*.

If (V, dist) is a metric space, then dist is called a *metric on V*. A TSP instance with vertices V and distance function dist is called *metric TSP instance* if (V, dist) is a metric space.

A well-known class of metrics on $\mathbb{R}^d$ is the class of *L_p metrics*. For $p \in \mathbb{N}$, the distance $\mathrm{dist}_p(x, y)$ of two points $x \in \mathbb{R}^d$ and $y \in \mathbb{R}^d$ with respect to the L_p metric is given by $\mathrm{dist}_p(x, y) = \sqrt[p]{|x_1 - y_1|^p + \cdots + |x_d - y_d|^p}$. The L_1 metric is often called *Manhattan metric*, and the L_2 metric is well-known as *Euclidean metric*. For $p \to \infty$, the L_p metric converges to the L_∞ metric defined by the distance function $\mathrm{dist}_\infty(x, y) = \max\{|x_1 - y_1|, \ldots, |x_d - y_d|\}$. A TSP instance (V, dist) with $V \subseteq \mathbb{R}^d$ in which dist equals dist_p restricted to V is called an *L_p instance*. We also use the terms *Manhattan instance* and *Euclidean instance* to denote L_1 and L_2 instances, respectively. Furthermore, if p is clear from context, we write dist instead of dist_p.

9.1 Overview of Results

Despite many theoretical analyses and experimental evaluations of the TSP, there is still a considerable gap between the theoretical results and the experimental obser-

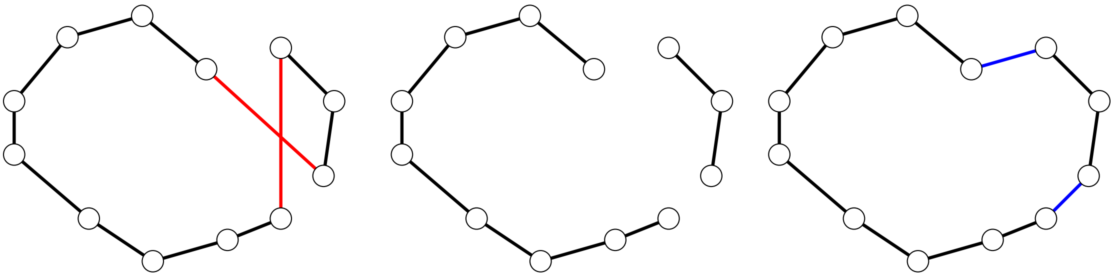


Figure 9.1: Example of a 2-change in which the red edges are exchanged with the blue edges.

vations. One important special case is the ***Euclidean TSP*** in which the vertices are points in $\mathbb{R}^d$, for some $d \quad \mathbb{N}$, and the distances are measured according to the Euclidean metric. This special case is known to be NP-hard in the strong sense [Pap77], but for any constant dimension d it admits a polynomial time approximation scheme (PTAS), shown independently in 1996 by Arora [Aro98] and Mitchell [Mit99]. These approximation schemes are based on dynamic programming. However, the most successful algorithms on practical instances rely on the principle of local search and very little is known about their complexity.

The ***2-Opt*** algorithm is probably the simplest local search heuristic for the TSP. 2-Opt starts with an arbitrary initial tour and incrementally improves this tour by successive improvements that exchange two of the edges in the tour with two other edges. More precisely, in each ***improving step*** the 2-Opt algorithm selects two edges $\{u_1, u_2\}$ and $\{v_1, v_2\}$ from the tour such that u_1, u_2, v_1, v_2 are distinct and appear in this order in the tour, and it replaces these edges by the edges $\{u_1, v_1\}$ and $\{u_2, v_2\}$, provided that this change decreases the length of the tour (see Figure 9.1). The algorithm terminates in a local optimum in which no further improving step is possible. We use the term ***2-change*** to denote a local improvement made by 2-Opt. This simple heuristic performs amazingly well on “real-life” Euclidean instances like, e.g., the ones in the well-known TSPLIB [Rei91]. Usually the 2-Opt heuristic needs a clearly subquadratic number of improving steps until it reaches a local optimum and the computed solution is within a few percentage points of the global optimum [JM97].

There are numerous experimental studies on the performance of 2-Opt. However, the theoretical knowledge about this heuristic is still very limited. Let us first discuss the number of local improvement steps made by 2-Opt before it finds a locally optimal solution. When talking about the number of local improvements, it is convenient to consider the ***state graph***. The vertices in this graph correspond to the possible tours and an arc from a vertex v to a vertex u is contained if u is obtained from v by performing an improving 2-change. On the positive side, van Leeuwen and Schoone consider a 2-Opt variant for the Euclidean plane in which only steps are allowed that remove a crossing from the tour. Such steps can introduce new crossings, but van Leeuwen and Schoone [vLS81] show that after $O(n^3)$ steps, 2-Opt finds a tour without any crossing. On the negative side, Lueker [Lue75] constructs TSP instances whose state graphs contain exponentially long paths. Hence, 2-Opt can take an exponential number of steps before it finds a locally optimal solution. This result is generalized to

k-Opt, for arbitrary $k \geq 2$, by Chandra, Karloff, and Tovey [CKT99]. These negative results, however, use arbitrary graphs that cannot be embedded into low-dimensional Euclidean space. Hence, they leave open the question as to whether it is possible to construct Euclidean TSP instances on which 2-Opt can take an exponential number of steps. This question is resolved by Englert et al. [ERV07] by constructing such instances in the Euclidean plane. In chip design applications, often TSP instances arise in which the distances are measured according to the Manhattan metric. Also for this metric and for every other L_p metric, Englert et al. construct instances with exponentially long paths in the 2-Opt state graph.

Theorem 9.1 ([ERV07])**.** *For every p N $\{\infty\}$ and n N, there is a two-dimensional TSP instance with $16n$ vertices in which the distances are measured according to the L_p metric and whose state graph contains a path of length $2^{n+4} - 22$.*

For Euclidean instances in which n points are placed independently uniformly at random in the unit square, Kern [Ker89] shows that the length of the longest path in the state graph is bounded by $O(n^{16})$ with probability at least $1 - c/n$ for some constant c. Chandra, Karloff, and Tovey [CKT99] improve this result by bounding the *expected* length of the longest path in the state graph by $O(n^{10} \log n)$. That is, independent of the initial tour and the choice of the local improvements, the expected number of 2-changes is bounded by $O(n^{10} \log n)$. For instances in which n points are placed uniformly at random in the unit square and the distances are measured according to the Manhattan metric, Chandra, Karloff, and Tovey show that the expected length of the longest path in the state graph is bounded by $O(n^6 \log n)$.

Englert et al. consider a more general probabilistic input model, which is a canonical extension of ϕ-perturbed numbers to higher dimensions, and improve the previously known bounds. The probabilistic model underlying their analysis allows different vertices to be placed independently according to different continuous probability distributions in the unit hypercube $[0,1]^d$, for some constant *dimension* $d \geq 2$. The distribution of a vertex v_i is defined by a density function $f_i \colon [0,1]^d \to [0,\phi]$ for some given $\phi \geq 1$. The upper bounds depend on the number n of vertices and the upper bound ϕ on the density. We denote instances created by this input model as *ϕ-perturbed Euclidean* or *Manhattan instances*, depending on the underlying metric. The parameter ϕ can be seen as a parameter specifying how close the analysis is to a worst case analysis because the larger ϕ, the better worst-case instances can be approximated by the distributions. For $\phi = 1$ and $d = 2$, every point has a uniform distribution over the unit square, and hence the input model equals the uniform model analyzed before.

Englert et al. also consider a model in which an arbitrary graph $G = (V, E)$ is given and for each edge e E, a probability distribution according to which the edge length $\mathrm{dist}(e)$ is chosen independently of the other edge lengths. Again, we restrict the choice of distributions to distributions that can be represented by density functions $f_e \colon [0,1] \to [0,\phi]$ with maximal density at most ϕ for a given $\phi \geq 1$. We denote inputs created by this input model as *ϕ-perturbed graphs*. Observe that in this input model only the distances are perturbed whereas the graph structure is not changed by the randomization. This can be useful if one wants to explicitly prohibit certain edges. However, if the graph G is not complete, one has to initialize 2-Opt with a Hamiltonian cycle to start with.

Englert et al. prove the following theorem about the expected length of the longest path in the 2-Opt state graph for the probabilistic input models discussed above. It is assumed that the dimension $d \geq 2$ is an arbitrary constant.

Theorem 9.2 ([ERV07]). *The expected length of the longest path in the 2-Opt state graph*

a) is $O(n^4 \cdot \phi)$ for ϕ-perturbed Manhattan instances with n points,

b) is $O(n^{4+1/3} \cdot \log(n\phi) \cdot \phi^{8/3})$ for ϕ-perturbed Euclidean instances with n points,

c) is $m^{1+o(1)} \cdot n \cdot \phi$ for ϕ-perturbed graphs with n vertices and m edges.

Similar to the running time, the good approximation ratios obtained by 2-Opt on practical instances cannot be explained by a worst-case analysis. In fact, there are quite negative results on the worst-case behavior of 2-Opt. For example, Chandra, Karloff, and Tovey [CKT99] show that there are Euclidean instances in the plane for which 2-Opt has local optima whose costs are $\Omega\left(\frac{\log n}{\log\log n}\right)$ times larger than the optimal costs. However, the same authors also show that the expected approximation ratio of the worst local optimum for instances with n points drawn uniformly at random from the unit square is bounded from above by a constant. Their result can be generalized to the input model in which different points can have different distributions with bounded density ϕ and to all L_p metrics.

Theorem 9.3 ([ERV07]). *Let p N $\{\infty\}$. For ϕ-perturbed L_p instances, the expected approximation ratio of the worst tour that is locally optimal for 2-Opt is $O(\sqrt[d]{\phi})$.*

9.2 Polynomial Bound for ϕ-Perturbed Graphs

Is this section, we present a glimpse into the proof of Theorem 9.2. The complete proof is too technical to be presented in the lecture in detail, but we will outline the main ideas by proving a weaker version of part c) of the theorem.

Theorem 9.4. *For any ϕ-perturbed graph with n vertices and m edges, the expected length of the longest path in the 2-Opt state graph is $O(n^2 m^2 \log(n) \cdot \phi)$.*

Proof. How can we prove an upper bound on the (expected) number of steps made by 2-Opt? For this, we use the length of the current tour as a *potential function*. As all edge lengths lie in the interval $[0, 1]$, any tour (in particular the one 2-Opt starts with) has length at most n. If we know that every 2-change decreases the length of the current tour by at least $\Delta > 0$, then we can bound the number of 2-changes that can be made before reaching a local optimum from above by n/Δ because after that many steps the length of the tour must have decreased to zero. As it cannot get negative, no further local improvement can be possible.

Hence, if we show that the smallest possible improvement Δ is not too small, then it follows that 2-Opt cannot make too many steps. Unfortunately, in the worst case one can easily construct a set of points that allows a 2-change that decreases the length of the tour only by an arbitrarily small amount. Our goal is to show that on ϕ-perturbed graphs, Δ does not become too small with high probability.

Let us first consider a fixed 2-change in which the edges e_1 and e_2 are exchanged with the edges e_3 and e_4. This 2-change decreases the length of the tour by

$$\Delta(e_1, e_2, e_3, e_4) = \operatorname{dist}(e_1) + \operatorname{dist}(e_2) - \operatorname{dist}(e_3) - \operatorname{dist}(e_4). \tag{9.1}$$

We define Δ as the smallest possible improvement made by any improving 2-change:

$$\Delta = \min_{\substack{e_1,e_2,e_3,e_4 \\ \Delta(e_1,e_2,e_3,e_4)>0}} \Delta(e_1, e_2, e_3, e_4),$$

where the minimum is taken over all tuples $(e_1, e_2, e_3, e_4) \quad E^4$ for which e_1, e_3, e_2, e_4 is a 4-cycle in G because only these tuples could possibly form a 2-change.

The following lemma, which is proven below, is one crucial ingredient of the proof.

Lemma 9.5. *For any* $\varepsilon > 0$,

$$\mathbf{Pr}\left(\Delta \le \varepsilon\right) \le m^2 \varepsilon \phi.$$

With the help of this lemma we can prove the theorem. We have argued above that the number of steps that 2-Opt can make is bounded from above by n/Δ. Let T denote the maximal number of steps 2-Opt can make, i.e., the length of the longest path in the state graph. This number can only be greater than or equal to t if $n/\Delta \ge t$, which is equivalent to $\Delta \le n/t$. Hence,

$$\mathbf{Pr}\left(T \ge t\right) \le \mathbf{Pr}\left(\Delta \le \frac{n}{t}\right) \le \frac{nm^2\phi}{t}.$$

One important observation is that T is always bounded from above by $n!$ because this is an upper bound on the number of possible tours. Hence, we obtain the following bound for the expected value of T:

$$\mathbf{E}\left[T\right] = \sum_{t=1}^{n!} \mathbf{Pr}\left(T \ge t\right) \le \sum_{t=1}^{n!} \frac{nm^2\phi}{t} = nm^2\phi \cdot \sum_{t=1}^{n!} \frac{1}{t}.$$

Using that

$$\sum_{t=1}^{n!} \frac{1}{t} = O(\log(n!)) = O(n \log(n))$$

yields

$$\mathbf{E}\left[T\right] = O(nm^2\phi \log(n!)) = O(n^2m^2 \log(n) \cdot \phi). \qquad \square$$

To complete the proof of the Theorem, we have to prove Lemma 9.5.

Proof of Lemma 9.5. Again we first consider a fixed 2-change in which the edges e_1 and e_2 are exchanged with the edges e_3 and e_4. We would like to bound the probability that this fixed 2-change is improving, but yields an improvement of at most ε, for some $\varepsilon > 0$. That is, we want to bound the following probability from above:

$$\mathbf{Pr}\left(\Delta(e_1, e_2, e_3, e_4) \quad (0, \varepsilon]\right) = \mathbf{Pr}\left(\text{dist}(e_1) + \text{dist}(e_2) - \text{dist}(e_3) - \text{dist}(e_4) \quad (0, \varepsilon]\right).$$

We use the principle of deferred decisions and assume that the lengths $\text{dist}(e_2)$, $\text{dist}(e_3)$, and $\text{dist}(e_4)$ have already been fixed arbitrarily. Then the event $\Delta(e_1, e_2, e_3, e_4) \quad (0, \varepsilon]$ is equivalent to the event that

$$\text{dist}(e_1) \quad (\kappa, \kappa + \varepsilon],$$

where $\kappa = \text{dist}(e_4) + \text{dist}(e_3) - \text{dist}(e_2)$ is some fixed value. As $\text{dist}(e_1)$ is a random variable whose density is bounded from above by ϕ, the probability that $\text{dist}(e_1)$ assumes a value in a fixed interval of length ε is at most $\varepsilon\phi$.

This bound makes only a statement about the improvement made by a particular step in which the edges e_1 and e_2 are exchanged with the edges e_3 and e_4, but we would like to make a statement about every possible 2-change. For this we apply a union bound over all possible 2-changes. There are $\binom{m}{2} < \frac{m^2}{2}$ choices for the set $\{e_1, e_2\}$ and, once this set is fixed, there are two choices for the set $\{e_3, e_4\}$ because e_1, e_3, e_2, e_4 has to be a 4-cycle. Hence, the total number of different 2-changes is bounded from above by m^2, which yields

$$\mathbf{Pr}\left(\Delta \quad (0, \varepsilon]\right) \leq \mathbf{Pr}\left(\; e_1, e_2, e_3, e_4 : \Delta(e_1, e_2, e_3, e_4) \quad (0, \varepsilon]\right) \leq m^2\varepsilon\phi,$$

where the existential quantifier in the second probability is over all $(e_1, e_2, e_3, e_4) \quad E^4$ for which e_1, e_3, e_2, e_4 is a 4-cycle in G. This concludes the proof of the lemma. □

9.3 Improved Analysis

The bound in Theorem 9.4 is only based on the smallest improvement Δ made by any of the 2-Opt steps. Intuitively, this is too pessimistic because most of the steps performed by 2-Opt yield a larger improvement than Δ. In particular, two consecutive steps yield an improvement of at least Δ plus the improvement Δ of the second smallest step. This observation alone, however, does not suffice to improve the bound substantially.

To obtain a significantly better bound, we show that it is possible to regroup most of the 2-changes in any sufficiently long sequence to pairs such that each pair of 2-changes is *linked* by an edge, i.e., one edge added to the tour in the first 2-change is removed from the tour in the second 2-change. Then we analyze the smallest improvement made by any pair of linked 2-changes. Obviously, this improvement is at least $\Delta + \Delta$ but one can hope that it is much larger because it is unlikely that the 2-change that yields the smallest improvement and the 2-change that yields the second smallest improvement form a pair of linked steps. It can be shown that this is indeed the case. This result is then used to prove another weaker version of part c) of Theorem 9.2 that improves upon Theorem 9.4.

Theorem 9.6. *For any ϕ-perturbed graph with n vertices and m edges, the expected length of the longest path in the 2-Opt state graph is $O(nm^{3/2}\phi)$.*

Construction of Pairs of Linked 2-Changes

Consider an arbitrary sequence of t consecutive 2-changes. The following lemma guarantees that the number of disjoint linked pairs of 2-changes in every such sequence increases linearly with the length t.

Lemma 9.7. *In every sequence of t consecutive 2-changes performed by 2-Opt, we can find at least $(2t - n)/7$ disjoint pairs of 2-changes that are linked by an edge, i.e., pairs where there exists an edge added to the tour in the first 2-change of the pair and removed from the tour in the second 2-change of the pair.*

Proof. Let $S_1, \ldots, S_t$ denote an arbitrary sequence of consecutive 2-changes. The sequence is processed step by step and a list L of linked pairs of 2-changes is created. The pairs in L are not necessarily disjoint. Hence, after the list has been created, pairs have to be removed from the list until there are no non-disjoint pairs left. Assume that the 2-changes $S_1, \ldots, S_{i-1}$ have already been processed and that now 2-change S_i has to be processed. Assume further that in step S_i the edges e_1 and e_2 are exchanged with the edges e_3 and e_4. Let j denote the smallest index with $j > i$ such that edge e_3 is removed from the tour in step S_j if such a step exists. In this case, the pair (S_i, S_j) is added to the list L. Analogously, let j denote the smallest index with $j > i$ such that edge e_4 is removed from the tour in step S_j if such a step exists. In this case, also the pair (S_i, S_j) is added to the list L.

After the sequence has been processed completely, each pair in L is linked by an edge but we still have to identify a subset L of L consisting only of pairwise disjoint pairs. This subset is constructed in a greedy fashion. We process the list L step by step, starting with an empty list L. For each pair in L, we check whether it is disjoint from all pairs that have already been inserted into L or not. In the former case, the current pair is inserted into L. This way, we obtain a list L of disjoint pairs such that each pair is linked by an edge. The number of pairs in L is at least $2t - n$ because each of the t steps gives rise to 2 pairs, unless an edge is added to the tour that is never removed again. The tour C obtained after the 2-changes $S_1, \ldots, S_t$ contains exactly n edges. For every edge e C, only the last step in which e enters the tour (if such a step exists) does not create a pair of linked 2-changes involving e.

Each 2-change occurs in at most 4 different pairs in L. In order to see this, consider a 2-change S in which the edges e_1 and e_2 are exchanged with the edges e_3 and e_4. Then L contains the following pairs involving S (if they exist): (S, S) where S is either the first step after S in which e_3 gets removed or the first step after S in which e_4 gets removed, and (S, S) where S is either the last step before S in which e_1 enters the tour or the last step before S in which e_2 enters the tour. With similar reasoning, one can argue that each pair in L is non-disjoint from at most 6 other pairs in L. This implies that L contains at most 7 times as many pairs as L, which concludes the proof. □

Analysis of Pairs of Linked 2-Changes

The following lemma gives a bound on the probability that there exists a linked pair of 2-changes in which both steps are small improvements.

Lemma 9.8. *In any ϕ-perturbed graph with n vertices and m edges, the probability that there exists a pair of linked 2-changes that are both improvements by at most ε is $O(m^3\phi^2\varepsilon^2)$.*

Proof. First consider a fixed pair of linked 2-changes. Assume that in the first 2-change the edges e_1 and e_2 are replaced by the edges e_3 and e_4 and that in the second 2-change the edges e_4 and e_5 are replaced by the edges e_6 and e_7. The sets $\{e_1, e_2, e_3\}$ and $\{e_5, e_6, e_7\}$ are not necessarily disjoint. They are, however, distinct because otherwise the second 2-change would be exactly the reverse of the first 2-change (i.e., the edges e_3 and e_4 are replaced by the edges e_1 and e_2). In this case, at least one of the 2-changes is not an improvement.

Let us assume without loss of generality that $e_6 \;/\; \{e_1, e_2, e_3\}$ and $e_1 \;/\; \{e_5, e_6, e_7\}$. Our goal is to give an upper bound on the probability that both

$$\Delta(e_1, e_2, e_3, e_4) := \mathrm{dist}(e_1) + \mathrm{dist}(e_2) - \mathrm{dist}(e_3) - \mathrm{dist}(e_4) \quad (0, \varepsilon]$$

and

$$\Delta(e_4, e_5, e_6, e_7) := \mathrm{dist}(e_4) + \mathrm{dist}(e_5) - \mathrm{dist}(e_6) - \mathrm{dist}(e_7) \quad (0, \varepsilon].$$

We use the principle of deferred decisions and assume that the lengths of the edges e_2, e_3, e_4, e_5, and e_7 are already revealed. For any fixed realization of these edge lengths, the events

$$\Delta(e_1, e_2, e_3, e_4) \quad (0, \varepsilon] \quad \text{and} \quad \Delta(e_4, e_5, e_6, e_7) \quad (0, \varepsilon]$$

can be written as

$$\mathrm{dist}(e_1) \quad (\kappa_1, \kappa_1 + \varepsilon] \quad \text{and} \quad \mathrm{dist}(e_6) \quad [\kappa_2 - \varepsilon, \kappa_2),$$

where $\kappa_1 = \mathrm{dist}(e_4) + \mathrm{dist}(e_3) - \mathrm{dist}(e_2)$ and $\kappa_2 = \mathrm{dist}(e_4) + \mathrm{dist}(e_5) - \mathrm{dist}(e_7)$ are constants depending on the realization of the edges e_2, e_3, e_4, e_5, and e_7. Hence, for any fixed realization these events are independent. Each of them occurs with a probability of at most $\phi\varepsilon$ because $\mathrm{dist}(e_1)$ and $\mathrm{dist}(e_6)$ are random variables whose densities are bounded from above by ϕ. In summary, for any linked pair of 2-changes the probability that both steps are improvements by at most ε is bounded from above by $\phi^2\varepsilon^2$.

To conclude the proof it suffices to apply a union bound over all possible pairs of linked 2-changes. As argued in the proof of Theorem 9.4, there are at most m^2 choices for the first 2-change. Once this 2-change is fixed, there are only two choices for the edge that links the first and the second 2-change. There are at most m choices for the other edge removed from the tour in the second step. Once the edges are fixed that are removed from the tour in the second step, there are only two possibilities for the edges added to the tour in that step. In total, there are only $O(m^3)$ different pairs of linked 2-changes. This completes the proof. □

Expected Number of 2-Changes

Based on Lemmas 9.7 and 9.8, we are now able to prove Theorem 9.6.

Proof of Theorem 9.6. Let T denote the random variable that describes the length of the longest path in the state graph. If $T \geq t$, then there must exist a sequence $S_1, \ldots, S_t$ of t consecutive 2-changes in the state graph. We start by identifying a set of disjoint linked pairs of 2-changes in this sequence. Due to Lemma 9.7, we know that we can find at least $z = (2t - n)/7$ such pairs. Let $\Delta^\star$ denote the smallest improvement made by any pair of linked 2-changes. If $T \geq t$, then $\Delta^\star \leq n/z$ as the initial tour has length at most n and every pair of linked 2-changes decreases the length of the tour by at least $\Delta^\star$. For $t \geq n$, we have $z = (2t - n)/7 \geq t/7$ and hence due to Lemma 9.8,

$$\mathbf{Pr}\left(T \geq t\right) \leq \mathbf{Pr}\left(\Delta^\star \leq \frac{n}{z}\right) \leq \mathbf{Pr}\left(\Delta^\star \leq \frac{7n}{t}\right) = O\left(\frac{m^3 n^2 \phi^2}{t^2}\right).$$

Using the fact that probabilities are bounded from above by one, we obtain, for some constant κ,

$$\mathbf{Pr}\left(T \geq t\right) \leq \min\left\{\frac{\kappa m^3 n^2 \phi^2}{t^2}, 1\right\}.$$

Since T cannot exceed $n!$, this implies the following bound on the expected number of 2-changes:

$$\begin{aligned}
\mathbf{E}\left[T\right] &= \sum_{t=1}^{n!} \mathbf{Pr}\left(T \geq t\right) \\
&\leq \sum_{t=1}^{n!} \min\left\{\frac{\kappa m^3 n^2 \phi^2}{t^2}, 1\right\} \\
&\leq m^{3/2} n\phi + \sum_{t=m^{3/2}n\phi}^{n!} \frac{\kappa m^3 n^2 \phi^2}{t^2} \\
&\leq m^{3/2} n\phi + \int_{m^{3/2}n\phi}^{\infty} \frac{\kappa m^3 n^2 \phi^2}{t^2}\, dt \\
&= m^{3/2} n\phi + \left[-\frac{\kappa m^3 n^2 \phi^2}{t}\right]_{m^{3/2}n\phi}^{\infty} \\
&= O(m^{3/2} n\phi).
\end{aligned}$$

This concludes the proof of the theorem. □

Part a) and b) of Theorem 9.2 can be proven with similar arguments that we used to prove Theorem 9.6. However, the calculations are much more involved because in Manhattan and Euclidean instances the lengths of different edges are not independent anymore if they share a vertex. The main idea of the proof of part c) is to analyze not only pairs of linked 2-changes but longer sequences of linked 2-changes.

Chapter 10

The k-Means Method

Clustering is a fundamental problem in computer science with applications ranging from biology to information retrieval and data compression. In a clustering problem, a set of objects, usually represented as points in a high-dimensional space R^d, is to be partitioned such that objects in the same group share similar properties. The k-means method is a very simple heuristic for clustering. It is used to partition a finite set X R^d of n d-dimensional data points into k clusters, where the number k of clusters is fixed in advance.

In k-means clustering, our goal is not only to get a clustering of the data points, but also to get a ***center*** c_i for each cluster C_i of the clustering $C_1, \ldots, C_k$. This center can be viewed as a representative of its cluster. We do not require the centers to be among the data points, but they can be arbitrary points in R^d.

The goal is to find a clustering that minimizes the objective function

$$\Psi = \sum_{i=1}^{k} \sum_{x \quad C_i} \quad x - c_i \quad ^2 .$$

Here, $x - c_i$ denotes the Euclidean distance between x and c_i. We will refer to the objective value Ψ also as ***potential***.

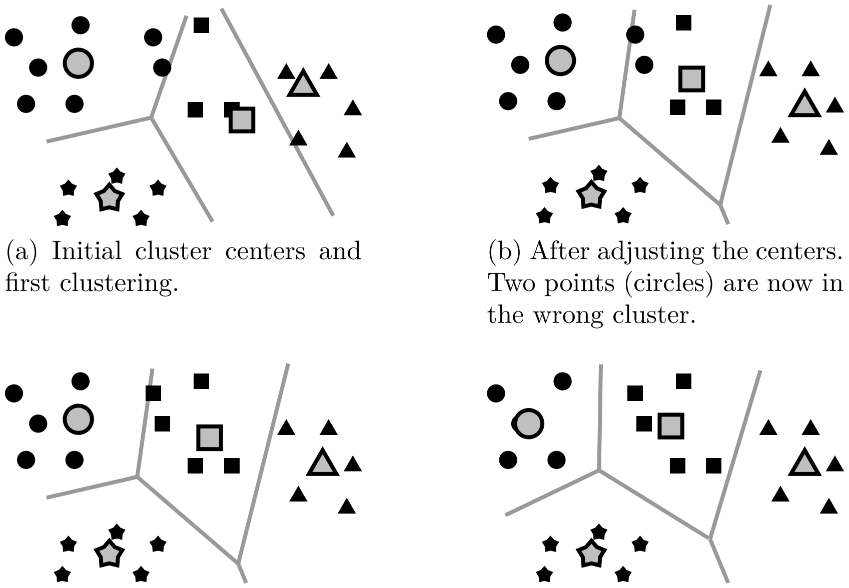
Given the cluster centers $c_1, \ldots, c_k$ R^d, each point x X should be assigned to the cluster C_i that is represented by its closest center c_i. On the other hand, given a clustering $C_1, \ldots, C_k$ of the data points, each center c_i should be chosen as the center of mass $\mathrm{cm}(C_i) := \frac{1}{|C_i|} \cdot \sum_{x \quad C_i} x$ of C_i in order to minimize the objective function (Lemma 10.3).

The ***k-means method*** (often called ***k-means*** for short or ***Lloyd's method*** because it is based on ideas by Lloyd [Llo82]) exploits this duality between clustering and centers: Given the centers, it is clear how the clustering should be chosen. And given the clustering, we know where the centers should be. We start with some clustering and then alternatingly optimize centers and clustering until this process stabilizes and we have reached a local optimum. Algorithm 1 states this more formally. Figure 10.1

provides an example run of the k-means method in two dimensions. Given the k centers, the Voronoi cell of a center c_i consists of all points closer to c_i than to any other center. The gray lines in Figure 10.1 are the borders between the Voronoi cells.

Algorithm 1 The k-means method.

```
Input: set $X$ of $n$ data points in $\mathsf{R}^d$, number $k$ of clusters
Output: clustering $C_1, \dots, C_k$ and centers $c_1, \dots, c_k$.
1: Choose initial cluster centers $c_1, \dots, c_k$   $\mathsf{R}^d$ arbitrarily.
2: For each point $x$   $X$: If $c_i$ is the center closest to $x$, then assign $x$ to $C_i$.
   (We assume a consistent tie-breaking if the closest center is not unique.)
3: For each $i$   $\{1, \dots, k\}$: Set $c_i = \mathrm{cm}(C_i)$.
4: If anything has changed in steps 2 and 3, then go back to step 2.
```



(a) Initial cluster centers and first clustering.

(b) After adjusting the centers. Two points (circles) are now in the wrong cluster.

(c) After reassigning the two points.

(d) After adjusting the cluster centers. This clustering is a local optimum of k-means.

Figure 10.1: An example of k-means.

In the following, an *iteration* of k-means refers to one execution of step 2 followed by step 3. A slight technical subtlety in the implementation of the algorithm is the possible event that a cluster loses all its points in step 2. There exist different strategies to deal with this case. For simplicity, we use the strategy of removing clusters that serve no points and continuing with the remaining clusters.

The k-means method is one of the most popular clustering algorithms used in scientific and industrial applications. The main reason for its popularity is its speed. This, however, is in stark contrast to its performance in theory: The worst-case running time of k-means has recently been shown to be exponential in the number k of clusters [Vat11]. The only known upper bound for its running time is $(k^2n)^{kd}$ [IKI00], but this bound is far from explaining the speed of k-means. It is solely based on the observation that the k-means method cannot visit the same clustering twice because the objective function Ψ decreases monotonically during the execution of the k-means method.

In order to explain the practical performance of k-means and to reconcile theory and practice, a smoothed analysis of k-means has been conducted. Here, an adversary specifies a set X $[0, 1]^d$ of n points. Then each point from X is independently perturbed by a normal distribution with mean 0 and standard deviation σ, yielding a new set X of points. We call X a *σ-smooth point set* (formally a σ-smooth point set is not a set of points but a set of random vectors). In a series of papers [AV09, MR09, AMR11], it has been shown that the smoothed running time (i.e., the expected running time on σ-smooth point sets) of the k-means method is bounded from above by a polynomial in the number n of data points and $1/\sigma$, i.e., it is bounded by $\text{poly}(n, 1/\sigma)$, where the degree of the polynomial depends neither on k nor on d. As the proof of this statement is rather involved, we prove in this lecture only the following weaker version from [AV09]. We assume for the sake of simplicity that $2 \leq d \leq n$, $k \leq n$, and $\sigma \leq 1$ even though none of these restrictions is crucial for the following result.

Theorem 10.1 ([AV09])**.** *Let X be a σ-smooth point set of cardinality n. Then the expected running time of k-means on X is $O((n^k/\sigma)^c)$, where the constant c is independent of d and k.*

Theorem 10.1 shows that the smoothed running time of the k-means method is polynomially bounded in n^k and $1/\sigma$. This is already a significant improvement over the best known worst-case upper bound, which is polynomial in n^{kd} instead of n^k.

In order to prove the theorem, we follow an approach similar to the analysis of 2-Opt in the previous chapter. That is, we bound the smallest improvement Δ of the objective value Ψ in any possible iteration of the k-means method from below. Before we discuss this in more detail, we will first discuss which events lead to a decrease of Ψ. Then we analyze in Sections 10.2 and 10.3 the probability that Δ is small for iterations in which many or few points change their assignment, respectively.

10.1 Potential Drop in an Iteration of k-Means

During an iteration of the k-means method there are two possible events that can lead to a significant potential drop: either one cluster center moves significantly, or a data point is reassigned from one cluster to another and this point has a significant distance from the bisector of the clusters (the bisector is the hyperplane that bisects the two cluster centers). In the following we quantify the potential drops caused by these events.

The potential drop caused by reassigning a data point x from one cluster to another can be expressed in terms of the distance of x from the bisector of the two cluster centers and the distance between these two centers.

Lemma 10.2. *Assume that in an iteration of k-means a point $x \in X$ switches from C_i to C_j. Let c_i and c_j be the centers of these clusters and let H be their bisector. Then reassigning x decreases the potential by $2 \cdot \|c_i - c_j\| \cdot \mathrm{dist}(x, H)$.*

Proof. The potential decreases by

$$\begin{aligned}
\|c_i - x\|^2 - \|c_j - x\|^2 &= \sum_{\ell=1}^{d} \left[[(c_i)_\ell - x_\ell]^2 - [(c_j)_\ell - x_\ell]^2 \right] \\
&= \sum_{\ell=1}^{d} \left[(c_i)_\ell^2 - (c_j)_\ell^2 - 2x_\ell \left[(c_i)_\ell - (c_j)_\ell \right] \right] \\
&= \sum_{\ell=1}^{d} [[(c_i)_\ell - (c_j)_\ell] [(c_i)_\ell + (c_j)_\ell] - 2x_\ell [(c_i)_\ell - (c_j)_\ell]] \\
&= \sum_{\ell=1}^{d} [[(c_i)_\ell - (c_j)_\ell] [(c_i)_\ell + (c_j)_\ell - 2x_\ell]] \\
&= (c_i - c_j) \cdot (c_i + c_j - 2x) \\
&= (2x - c_i - c_j) \cdot (c_j - c_i).
\end{aligned}$$

Let v be the unit vector in the $c_j - c_i$ direction. Then $(2x - c_i - c_j) \cdot v = 2 \cdot \mathrm{dist}(x, H)$ because v is orthogonal to H and c_i and c_j have the same distance to H, which implies $c_i \cdot v = -c_j \cdot v$. The observation $c_j - c_i = \|c_i - c_j\| \cdot v$ completes the proof. □

The following lemma, which also follows from basic linear algebra, reveals how moving a cluster center to the center of mass decreases the potential.

Lemma 10.3 ([KMN⁺04]). *Assume that the center of a cluster C moves from c to $\mathrm{cm}(C)$ during an iteration of k-means, and let $|C|$ denote the number of points in C when the movement occurs. Then the potential decreases by $|C| \cdot \|c - \mathrm{cm}(C)\|^2$.*

Proof. It is well-known that it is possible to express the contribution of C to the current potential based on the center of C. We will see this in the following. Then, we can compute how much changing the center changes the potential.

We first recall the following calculation rule that holds for all $x, y \in \mathbb{R}^d$:

$$||x + y||^2 = (x + y) \cdot (x + y) = x \cdot x + 2 \cdot x \cdot y + y \cdot y = ||x||^2 + 2 \cdot x \cdot y + ||y||^2.$$

This allows us to conveniently rewrite the squared distance of a point $x \in C$ to an arbitrary center $z \in \mathbb{R}^d$ as $||x - z||^2 = ||x - \mathrm{cm}(C) + \mathrm{cm}(C) - z||^2 = ||x - \mathrm{cm}(C)||^2 + 2(\mathrm{cm}(C) - z) \cdot (x - \mathrm{cm}(C)) + ||\mathrm{cm}(C) - z||^2$. We would like to get rid of the middle term. Luckily, it disappears when we compute the squared distances of *all* points to z. More precisely, we observe that

$$\sum_{x \in C} (x - \mathrm{cm}(C)) = \left(\sum_{x \in C} x \right) - |C| \cdot \mathrm{cm}(C) = \left(\sum_{x \in C} x \right) - \left(\sum_{x \in C} x \right) = 0,$$

which helps us to compute C's contribution to the potential with center z as:

$$\begin{aligned}
&\sum_{x \in C} \|x - z\|^2 \\
&= \sum_{x \in C} \|x - \mathrm{cm}(C)\|^2 + \sum_{x \in C} 2(x - \mathrm{cm}(C)) \cdot (\mathrm{cm}(C) - z) + \sum_{x \in C} \|z - \mathrm{cm}(C)\|^2 \\
&= \sum_{x \in C} \|x - \mathrm{cm}(C)\|^2 + 2(\mathrm{cm}(C) - z) \cdot \sum_{x \in C} (x - \mathrm{cm}(C)) + |C| \cdot \|z - \mathrm{cm}(C)\|^2 \\
&= \sum_{x \in C} \|x - \mathrm{cm}(C)\|^2 + 2(\mathrm{cm}(C) - z) \cdot 0 + |C| \cdot \|z - \mathrm{cm}(C)\|^2 \\
&= \sum_{x \in C} \|x - \mathrm{cm}(C)\|^2 + |C| \cdot \|z - \mathrm{cm}(C)\|^2.
\end{aligned}$$

We see that $\sum_{x \in C} \|x - \mathrm{cm}(C)\|^2$ appears in the potential independently of the center. For center $\mathrm{cm}(C)$, the additional term is zero. For center c, the additional term is $|C| \cdot \|c - \mathrm{cm}(C)\|^2$. So changing the center from c to $\mathrm{cm}(C)$ decreases the potential by exactly $|C| \cdot \|c - \mathrm{cm}(C)\|^2$. □

10.2 Iterations with Large Cluster Changes

In this section, we consider iterations in which one cluster C gains or loses in total at least $2kd$ points. In this case, there must exist another cluster C' such that C and C' exchange at least $2d + 1$ points. We show that with high probability at least one of these points is not too close to the bisector of C and C'. Together with Lemma 10.2 this implies that the potential decreases significantly.

Definition 10.4. *Let $X \subseteq \mathbb{R}^d$ be a set of points. We say that X is* δ-separated *if for any hyperplane H there are at most $2d$ points in X within distance δ of H.*

Lemma 10.5. *Assume that X is δ-separated. If one cluster gains or loses in total at least $2kd$ points within a single iteration, then the potential drops by at least δ^2/n during this iteration.*

Proof. If a cluster i gains or loses in total at least $2kd$ points in a single iteration, then there exists another cluster j with which i exchanges at least $2d+1$ points. Let $C_i \subseteq X$ and $C_j \subseteq X$ denote the sets of points assigned to cluster i and cluster j before the point exchange, respectively. Furthermore, let $c_i = \mathrm{cm}(C_i)$ and $c_j = \mathrm{cm}(C_j)$ be the cluster centers before the point exchange. Since X is δ-separated, one of the switching points, say, x, must be at a distance of at least δ from the hyperplane H bisecting c_i and c_j. Assume that x switches from C_i to C_j. Then the potential decreases by at least $\|c_i - x\|^2 - \|c_j - x\|^2 = 2 \cdot \|c_i - c_j\| \cdot \mathrm{dist}(x, H) \geq 2 \cdot \|c_i - c_j\| \cdot \delta$ by Lemma 10.2.

Now let H' denote the hyperplane bisecting the centers of the clusters i and j in the previous iteration. While H' does not necessarily bisect c_i and c_j, it divides the data points belonging to C_i and C_j correctly. Hence, C_i and C_j lie on different sides of H', which implies that $\|c_i - c_j\| \geq \mathrm{dist}(c_i, H') + \mathrm{dist}(c_j, H')$. Consider the at least $2d + 1$ data points switching between clusters i and j. One of them, say, y, must have a distance of at least δ from H because X is δ-separated. Let us assume w.l.o.g. that this point y belongs to C_i.

Since y is in C_i, the average distance of the points in C_i to H is at least δ/n: even if C_i contained n points and all except y had distance zero to H , the sum of all distances is at least δ because of y's contribution. Since $\mathrm{dist}(c_i, H\ )$ is equal to the average distance of the points from C_i to H , this means that $\ \ c_i - c_j \ \ \geq \delta/n$, so the potential decreases by at least $2 \cdot \ \ c_i - c_j \ \ \cdot \delta \geq 2\delta^2/n$. □

Lemma 10.6 ([AV09])**.** *Let $X \quad \mathrm{R}^d$ be a set of at least d points and let H denote an arbitrary hyperplane. Then there exists a hyperplane H passing through d points of X that satisfies*

$$\max_{x \quad X} \mathrm{dist}(x, H\ ) \leq 2d \cdot \max_{x \quad X} \mathrm{dist}(x, H).$$

Lemma 10.7. *Let X be a set of σ-smooth points with $n = |X|$. The probability that X is not δ-separated is bounded from above by*

$$n^{2d}\left(\frac{4d\delta}{\sigma}\right)^d.$$

Proof. If X is not δ-separated then there exists a hyperplane H and a subset $Y \quad X$ of points with $|Y| = 2d$ such that all points in Y have distance at most δ from H (in fact, there exists even such a set $Y \quad X$ with $|Y| = 2d+1$). According to Lemma 10.6 this implies that there exists a hyperplane H that passes through d points of Y and from which all points in Y have a distance of at most $2d\delta$. Hence, it suffices to bound the probability that such a hyperplane H exists.

We apply a union bound over all choices for the set $Y \quad X$ with $|Y| = 2d$ and the hyperplane H . There are $\binom{n}{2d}$ choices for the set Y and, once this set is fixed, there are $\binom{2d}{d}$ choices for the points that H passes through. In total there are at most

$$\binom{n}{2d} \cdot \binom{2d}{d} = \frac{n!}{(n-2d)! \cdot (2d)!} \cdot \frac{(2d)!}{(d!)^2} = \frac{n!}{(n-2d)! \cdot (d!)^2} \leq \frac{n!}{(n-2d)!} \leq n^{2d}$$

choices. Now assume that the set Y and the d points from Y that H passes trough are fixed. Let $Y_0 \quad Y$ denote the d points that H passes through. We use the principle of deferred decisions and reveal the positions of all points from Y_0. Then also H is determined (here we use implicitly that any subset of X of size d is linearly independent with probability 1 for σ-smooth sets X). Let $Y_1 = Y \setminus Y_0$. Since H is fixed and the positions of the points in Y_1 are independent, we obtain

$$\mathbf{Pr}\left(\ y \quad Y_1 : \mathrm{dist}(y, H\ ) \leq 2d\delta\right) = \prod_{y \quad Y_1} \mathbf{Pr}\left(\mathrm{dist}(y, H\ ) \leq 2d\delta\right).$$

Each y is a Gaussian random vector with standard deviation σ. Hence, it remains to analyze the probability that such a random vector is within distance δ of some fixed hyperplane H . Let v denote the normal vector of H . Then $\mathrm{dist}(y, H\ ) = |y \cdot v|$. One extremely useful property of Gaussian random vectors is that any orthogonal projection of a Gaussian random vector to an affine subspace is again a Gaussian random vector with the same standard deviation in that subspace. In particular, $y \cdot v$ is a one-dimensional Gaussian random variable with standard deviation σ. Hence, the density of $y \cdot v$ is bounded from above by $1/(\sigma\sqrt{2\pi}) \leq 1/\sigma$. This implies

$$\prod_{y \in Y_1} \mathbf{Pr}\left(\operatorname{dist}(y, H) \leq 2d\delta\right) = \prod_{y \in Y_1} \mathbf{Pr}\left(y \cdot v \in [-2d\delta, 2d\delta]\right) \leq \left(\frac{4d\delta}{\sigma}\right)^{|Y_1|} = \left(\frac{4d\delta}{\sigma}\right)^{d}.$$

Now the union bound over all at most n^{2d} choices for Y_0 and Y_1 concludes the proof. □

10.3 Iterations with Small Cluster Changes

We will now consider sequences of 2^k consecutive iterations in which in every iteration every cluster gains or loses at most $2kd$ points. In the following, we will use the term *configuration of a cluster* to refer to the set of points that are assigned to that cluster.

Lemma 10.8. *Consider a sequence of 2^k consecutive iterations of the k-means method and let $\mathcal{C}_1, \ldots, \mathcal{C}_{2^k+1}$ denote the corresponding sequence of clusterings (including the initial clustering $\mathcal{C}_1$ before the first iteration in the sequence). Then there exists at least one cluster that takes on at least three different configurations in $\mathcal{C}_1, \ldots, \mathcal{C}_{2^k+1}$.*

Proof. During the considered sequence the k-means method encounters $2^k + 1$ clusterings. Assume that every cluster takes on at most two different configurations. Then one clustering must repeat in the sequence $\mathcal{C}_1, \ldots, \mathcal{C}_{2^k+1}$. Since the potential Ψ decreases in each iteration, this cannot happen. □

For two sets A and B let $A \triangle B$ denote their *symmetric difference*, i.e.,

$$A \triangle B := (A \setminus B) \cup (B \setminus A).$$

Definition 10.9. *Let $X \subseteq \mathbb{R}^d$ be a set of points. We say that X is δ-*sparse *if there do not exist pairwise distinct sets $S_1, S_2, S_3 \subseteq X$ with $|S_1 \triangle S_2| \leq 2kd$ and $|S_2 \triangle S_3| \leq 2kd$ for which $\|\operatorname{cm}(S_1) - \operatorname{cm}(S_2)\| \leq \delta$ and $\|\operatorname{cm}(S_2) - \operatorname{cm}(S_3)\| \leq \delta$.*

Lemma 10.10. *Assume that X is δ-sparse. Then every sequence of 2^k consecutive iterations in which in every iteration every cluster gains or loses in total at most $2kd$ points improves the potential by at least δ^2.*

Proof. According to Lemma 10.8, there is one cluster that assumes three different configurations S_1, S_2, and S_3 in this sequence. Due to the assumption in Lemma 10.10, we have $|S_1 \triangle S_2| \leq 2kd$ and $|S_2 \triangle S_3| \leq 2kd$. Hence, due to the definition of δ-sparse, we have $\|\operatorname{cm}(S_i) - \operatorname{cm}(S_{i+1})\| > \delta$ for one $i \in \{1, 2\}$. Combining this with Lemma 10.3 concludes the proof. □

We denote by $\mathcal{B}(z, \varepsilon)$ the d-dimensional hyperball with center $z \in \mathbb{R}^d$ and radius $\varepsilon \geq 0$.

Lemma 10.11. *Let $z \quad \mathsf{R}^d$ and $\varepsilon \geq 0$ be arbitrary and let x be a d-dimensional Gaussian random vector with standard deviation σ and arbitrary mean. Then*

$$\mathbf{Pr}\left(x \quad \mathsf{B}(z,\varepsilon)\right) \leq \left(\frac{\varepsilon}{\sigma}\right)^d .$$

Proof. The probability density function of x is bounded from above by $1/(\sqrt{2\pi}\sigma)^d$. The hyperball $\mathsf{B}(z,\varepsilon)$ is contained in a hypercube of side length 2ε and volume $(2\varepsilon)^d$. Hence, by the same reasoning as in Lemma 5.6 it follows that the probability that x lies in $\mathsf{B}(z,\varepsilon)$ is bounded from above by

$$\left(\frac{1}{\sqrt{2\pi}\sigma}\right)^d \cdot (2\varepsilon)^d \leq \left(\frac{\varepsilon}{\sigma}\right)^d . \qquad \square$$

Lemma 10.12. *Let X be a set of σ-smooth points with $n = |X|$. The probability that X is not δ-sparse is bounded from above by*

$$(7n)^{4kd}\left(\frac{4n^3\delta}{\sigma}\right)^d .$$

Proof. Given sets S_1, S_2, and S_3 with $|S_1 \quad S_2| \leq 2kd$ and $|S_2 \quad S_3| \leq 2kd$, we can write S_i as the disjoint union of a common ground set $A = S_1 \cap S_2 \cap S_3$ with the set $B_i = S_i \setminus A$. Then $B_1 \cap B_2 \cap B_3 = \quad$. Furthermore,

$$B_1 \quad B_2 \quad B_3 = (S_1 \quad S_2 \quad S_3) \setminus A = (S_1 \quad S_2) \quad (S_2 \quad S_3).$$

Hence, $|B_1 \quad B_2 \quad B_3| = |(S_1 \quad S_2) \quad (S_2 \quad S_3)| \leq 4kd$.

We use a union bound over all choices for the sets B_1, B_2, and B_3. The number of choices for these sets is bounded from above by $7^{4kd}\binom{n}{4kd} \leq (7n)^{4kd}$, which can be seen as follows: We choose $4kd$ candidate points for $B_1 \quad B_2 \quad B_3$, and then for each point, we choose which set(s) it belongs to (it does not belong to all of them, but we allow that it belongs to none of them because otherwise we would not cover the case that $B_1 \quad B_2 \quad B_3$ contains fewer than $4kd$ points). We assume in the following that the sets B_1, B_2, and B_3 are fixed. For $i \quad \{1,2\}$, we can write $\mathrm{cm}(S_i) - \mathrm{cm}(S_{i+1})$ as

$$\left(\frac{|A|}{|A|+|B_i|} - \frac{|A|}{|A|+|B_{i+1}|}\right)\cdot \mathrm{cm}(A) + \frac{|B_i|}{|A|+|B_i|}\cdot \mathrm{cm}(B_i) - \frac{|B_{i+1}|}{|A|+|B_{i+1}|}\cdot \mathrm{cm}(B_{i+1}) . \tag{10.1}$$

Let us first consider the case that we have $|B_i| = |B_{i+1}|$ for one $i \quad \{1,2\}$. Then $\mathrm{cm}(S_i) - \mathrm{cm}(S_{i+1})$ simplifies to

$$\frac{|B_i|}{|A|+|B_i|}\cdot(\mathrm{cm}(B_i) - \mathrm{cm}(B_{i+1})) = \frac{1}{|A|+|B_i|}\cdot\left(\sum_{x \quad B_i \setminus B_{i+1}} x - \sum_{x \quad B_{i+1}\setminus B_i} x\right) .$$

Since $B_i = B_{i+1}$, there exists a point $y \quad B_i \quad B_{i+1}$. Let us assume without loss of generality that $y \quad B_i \setminus B_{i+1}$ and that the positions of all points in $(B_i \quad B_{i+1}) \setminus \{y\}$ are fixed arbitrarily. Then the event that $\mathrm{cm}(S_i) - \mathrm{cm}(S_{i+1}) \leq \delta$ is equivalent to the event that y lies in a fixed hyperball of radius $(|A|+|B_i|)\delta \leq n\delta$. Hence, the probability is bounded from above by $(n\delta/\sigma)^d \leq (4n^3\delta/\sigma)^d$ according to Lemma 10.11.

Now assume that $|B_1| = |B_2| = |B_3|$. For $i \quad \{1, 2\}$, we set

$$r_i = \left(\frac{|A|}{|A|+|B_i|} - \frac{|A|}{|A|+|B_{i+1}|}\right)^{-1} = \frac{(|A|+|B_i|)\cdot(|A|+|B_{i+1}|)}{|A|\cdot(|B_{i+1}|-|B_i|)}$$

and

$$Z_i = \frac{|B_{i+1}|}{|A|+|B_{i+1}|}\cdot \mathrm{cm}(B_{i+1}) - \frac{|B_i|}{|A|+|B_i|}\cdot \mathrm{cm}(B_i)\,.$$

According to Equation (10.1), the event $\quad \mathrm{cm}(S_i) - \mathrm{cm}(S_{i+1}) \quad \leq \delta$ is equivalent to the event that $\mathrm{cm}(A)$ falls into the hyperball with radius $|r_i|\delta$ and center $r_i Z_i$. Hence, the event that both $\quad \mathrm{cm}(S_1) - \mathrm{cm}(S_2) \quad \leq \delta$ and $\quad \mathrm{cm}(S_2) - \mathrm{cm}(S_3) \quad \leq \delta$ can only occur if the hyperballs $B(r_1 Z_1, |r_1|\delta)$ and $B(r_2 Z_2, |r_2|\delta)$ intersect. This event occurs if and only if the centers $r_1 Z_1$ and $r_2 Z_2$ have a distance of at most $(|r_1| + |r_2|)\delta$ from each other. Hence,

$$\begin{aligned} &\mathbf{Pr}\left((\quad \mathrm{cm}(S_1) - \mathrm{cm}(S_2) \quad \leq \delta) \quad (\quad \mathrm{cm}(S_2) - \mathrm{cm}(S_3) \quad \leq \delta)\right) \\ \leq\, &\mathbf{Pr}\left(\quad r_1 Z_1 - r_2 Z_2 \quad \leq (|r_1| + |r_2|)\delta\right)\,. \end{aligned}$$

After some algebraic manipulations, we can write the vector $r_1 Z_1 - r_2 Z_2$ as

$$\begin{aligned} &- \frac{|A|+|B_2|}{|A|\cdot(|B_2|-|B_1|)}\cdot \sum_{x \quad B_1} x - \frac{|A|+|B_2|}{|A|\cdot(|B_3|-|B_2|)}\cdot \sum_{x \quad B_3} x \\ &+ \left(\frac{|A|+|B_1|}{|A|\cdot(|B_2|-|B_1|)} + \frac{|A|+|B_3|}{|A|\cdot(|B_3|-|B_2|)}\right)\cdot \sum_{x \quad B_2} x\,. \end{aligned}$$

Since $B_1 = B_3$, there must be a $y \quad B_1 \quad B_3$. We can assume that $y \quad B_1 \setminus B_3$. If $y \,/\, B_2$, we let an adversary choose all positions of the points in $B_1 \quad B_2 \quad B_3 \setminus \{y\}$. Then the event $\quad r_1 Z_1 - r_2 Z_2 \quad \leq (|r_1| + |r_2|)\delta$ is equivalent to y falling into a fixed hyperball of radius

$$\begin{aligned} &\left|\frac{|A|\cdot(|B_2|-|B_1|)}{|A|+|B_2|}\right|\cdot(|r_1|+|r_2|)\delta \\ =&\left|(|B_2|-|B_1|)\cdot\left(\left|\frac{|A|+|B_1|}{|B_2|-|B_1|}\right| + \left|\frac{|A|+|B_3|}{|B_3|-|B_2|}\right|\right)\right|\delta \leq 4kdn\delta \leq 4n^3\delta, \end{aligned}$$

where we used for the second to last inequality that $|B_2| - |B_1| \leq 2kd$ and that $|A| + |B_i| \leq n$ for every $i \quad \{1, 2, 3\}$. The probability of this event is thus bounded from above by $(4n^3\delta/\sigma)^d$.

It remains to consider the case that $x \quad (B_1 \cap B_2) \setminus B_3$. Also in this case we let an adversary choose the positions of the points in $B_1 \quad B_2 \quad B_3 \setminus \{x\}$. Now the event $\quad r_1 Z_1 - r_2 Z_2 \quad \leq (|r_1| + |r_2|)\delta$ is equivalent to x falling into a fixed hyperball of radius

$$\left|-\frac{|A|+|B_2|}{|A|\cdot(|B_2|-|B_1|)} + \left(\frac{|A|+|B_1|}{|A|\cdot(|B_2|-|B_1|)} + \frac{|A|+|B_3|}{|A|\cdot(|B_3|-|B_2|)}\right)\right|^{-1}\cdot(|r_1|+|r_2|)\delta$$

$$= \left|\frac{|A| \cdot (|B_3| - |B_2|)}{|A| + |B_2|}\right| \cdot (|r_1| + |r_2|)\delta$$
$$= \left|(|B_3| - |B_2|) \cdot \left(\left|\frac{|A| + |B_1|}{|B_2| - |B_1|}\right| + \left|\frac{|A| + |B_3|}{|B_3| - |B_2|}\right|\right)\right| \delta \leq 4kdn\delta \leq 4n^3\delta.$$

Hence, the probability is bounded from above by $(4n^3\delta/\sigma)^d$ also in this case.

This concludes the proof because there are at most $(7n)^{4kd}$ choices for B_1, B_2, and B_3 and, for every choice, the probability that both $\ \mathrm{cm}(S_1) - \mathrm{cm}(S_2)\ \ \leq \delta$ and $\ \mathrm{cm}(S_2) - \mathrm{cm}(S_3)\ \ \leq \delta$ is at most $(4n^3\delta/\sigma)^d$. □

10.4 Proof of Theorem 10.1

Lemma 10.13. *Let $D_{\mathsf{max}} := \sigma\sqrt{8kd\ln(n)} + 1$ and let X be a set of n Gaussian random vectors in R^d with mean values in $[0,1]^n$ and standard deviation σ (i.e., X is a σ-smooth point set). Let F denote the event that $X \quad [-D_{\mathsf{max}}, D_{\mathsf{max}}]^d$. Then*

$$\mathbf{Pr}\left(F\right) \leq \frac{1}{n^{3kd}}.$$

We need the following lemma before we can prove Lemma 10.13.

Lemma 10.14. *Let X be a Gaussian random variable with mean 0 and standard deviation 1. Then $\mathbf{Pr}\left(|X| > x\right) \leq \exp\left(-x^2/2\right)$ for all $x \geq 1$.*

Proof. The density function of X is symmetric around 0. Observe that in the following calculation $t \geq x \geq 1$:

$$\mathbf{Pr}\left(|X| > x\right) = 2\int_x^\infty \frac{1}{\sqrt{2\pi}} \cdot \exp\left(-\frac{t^2}{2}\right) \mathrm{d}t \leq \frac{2}{\sqrt{2\pi}} \int_x^\infty t \cdot \exp\left(-\frac{t^2}{2}\right) \mathrm{d}t$$
$$= \frac{\sqrt{2}}{\sqrt{\pi}} \cdot \left[-\exp(-t^2/2)\right]_x^\infty = \frac{\sqrt{2} \cdot \exp(-x^2/2)}{\sqrt{\pi}} \leq \exp(-x^2/2). \qquad \square$$

Proof of Lemma 10.13. We use a union bound over all nd coordinates of the points in X. Let x be such a coordinate. Then x is a Gaussian with mean $\mu \quad [0,1]$ and standard deviation σ. We apply Lemma 10.14 to the random variable $y = (x - \mu)/\sigma$, which is a Gaussian with mean 0 and standard deviation 1:

$$\mathbf{Pr}\left(|x| > D_{\mathsf{max}}\right) \leq \mathbf{Pr}\left(|x - \mu| > \sigma\sqrt{8kd\ln(n)}\right) = \mathbf{Pr}\left(|y| > \sqrt{8kd\ln(n)}\right)$$
$$\leq \exp\left(-4kd\ln(n)\right) = \frac{1}{n^{4kd}} \leq \frac{1}{nd \cdot n^{3kd}}.$$

Now the lemma follows from the union bound over the nd coordinates. □

In the following we will use several inequalities that are only true for sufficiently large values of n (i.e., n must be larger than a certain constant). We will assume without further mention that n is sufficiently large and that $2 \le d \le n$.

Let Δ denote the smallest total potential decrease made by any sequence of 2^k consecutive iterations of the k-means method.

Lemma 10.15. *For every* $\varepsilon \ge 0$,

$$\mathbf{Pr}(\Delta \le \varepsilon) \le \frac{n^{16k}\varepsilon}{\sigma^2}.$$

Proof. If X is $\sqrt{\varepsilon}$-sparse and $\sqrt{n\varepsilon}$-separated then according to Lemma 10.10 and Lemma 10.5 every sequence of 2^k iterations decreases the potential by at least ε. Hence,

$$\begin{aligned}
\mathbf{Pr}(\Delta \le \varepsilon) &\le \mathbf{Pr}\left(X \text{ is not } \sqrt{\varepsilon}\text{-sparse or } X \text{ is not } \sqrt{n\varepsilon}\text{-separated}\right)\\
&\le \mathbf{Pr}\left(X \text{ is not } \sqrt{\varepsilon}\text{-sparse}\right) + \mathbf{Pr}\left(X \text{ is not } \sqrt{n\varepsilon}\text{-separated}\right)\\
&\le (7n)^{4kd}\left(\frac{4n^3\sqrt{\varepsilon}}{\sigma}\right)^d + n^{2d}\left(\frac{4d\sqrt{n\varepsilon}}{\sigma}\right)^d\\
&\le 2\cdot(7n)^{4kd}\left(\frac{4n^3\sqrt{\varepsilon}}{\sigma}\right)^d\\
&= 2\cdot\left(\frac{(7n)^{4k}\cdot 4n^3\sqrt{\varepsilon}}{\sigma}\right)^d\\
&\le 2\cdot\left(\frac{(n^{1/5}n)^{4k}\cdot 4n^3\sqrt{\varepsilon}}{\sigma}\right)^d\\
&\le \left(\frac{n^{5k+3}\sqrt{\varepsilon}}{\sigma}\right)^d\\
&\le \left(\frac{n^{8k}\sqrt{\varepsilon}}{\sigma}\right)^d.
\end{aligned}$$

From this we can conclude

$$\mathbf{Pr}(\Delta \le \varepsilon) \le \frac{n^{16k}\varepsilon}{\sigma^2}.$$

For $\frac{n^{8k}\sqrt{\varepsilon}}{\sigma} \le 1$ this follows from $d \ge 2$ and for $\frac{n^{8k}\sqrt{\varepsilon}}{\sigma} > 1$ it follows because $\mathbf{Pr}(\Delta \le \varepsilon)$ is always bounded from above by 1. □

Proof of Theorem 10.1. Without loss of generality we assume that the initial centers are in the convex hull of X (if this is not the case initially then it is true after the first iteration). Remember our assumption that $\sigma \le 1$. For sufficiently large n, we have

$$\begin{aligned}
D_{\mathsf{max}} &= \sigma\sqrt{8kd\ln(n)} + 1\\
&\le \sqrt{8kd\ln(n)} + 1\\
&\le 2\sqrt{8n^3}\\
&\le n^2/2.
\end{aligned}$$

If the failure event F does not occur (i.e., if $X \quad [-D_{\max}, D_{\max}]^d$) and if all centers are in the convex hull of X then the distance between any point from X and any center is at most $2\sqrt{d}D_{\max}$. Hence, the potential is bounded from above by

$$4ndD_{\max}^2 \leq 4nd(n^2/2)^2 \leq n^6.$$

According to the definition of Δ, every sequence of 2^k consecutive iterations decreases the potential by at least Δ. Hence, the number of iterations T can only exceed $2^k t$ if $\Delta \leq n^6/t$ or if F occurs. According to Lemma 10.13 and Lemma 10.15,

$$\begin{aligned}\mathbf{Pr}\left(T \geq 2^k t\right) &\leq \mathbf{Pr}\left(F\right) + \mathbf{Pr}\left(\Delta \leq \frac{n^6}{t}\right)\\ &\leq \frac{1}{n^{3kd}} + \frac{n^{16k}n^6}{t\sigma^2} \leq \frac{1}{n^{3kd}} + \frac{n^{22k}}{t\sigma^2}.\end{aligned}$$

Using the worst-case upper bound of $(k^2n)^{kd} \leq n^{3kd}$ for the number of iterations, this implies the following bound on the expected value of T:

$$\begin{aligned}\mathbf{E}\left[T\right] &= \sum_{t=1}^{n^{3kd}} \mathbf{Pr}\left(T \geq t\right)\\ &\leq 2^k + \sum_{t=1}^{n^{3kd}} 2^k \cdot \mathbf{Pr}\left(T \geq 2^k t\right)\\ &\leq 2^k + \sum_{t=1}^{n^{3kd}} 2^k \left(\frac{1}{n^{3kd}} + \frac{n^{22k}}{t\sigma^2}\right)\\ &\leq 2^{k+1} + \sum_{t=1}^{n^{3kd}} \frac{n^{23k}}{t\sigma^2}\\ &= 2^{k+1} + \frac{n^{23k}}{\sigma^2} + \sum_{t=2}^{n^{3kd}} \frac{n^{23k}}{t\sigma^2}\\ &\leq 2^{k+1} + \frac{n^{23k}}{\sigma^2} + \int_1^{n^{3kd}} \frac{n^{23k}}{t\sigma^2}\,dt\\ &\leq 2^{k+1} + \frac{n^{23k}}{\sigma^2} + \left[\frac{n^{23k}\ln(t)}{\sigma^2}\right]_1^{n^{3kd}}\\ &= O\left(\frac{n^{26k}}{\sigma^2}\right).\end{aligned}$$

This concludes the proof of the theorem. □

In the previous analysis, we did not optimize the degree of the polynomial. The reader has probably noticed that many estimates were rather generous and that a more careful analysis would lead to a polynomial with smaller degree. However, our goal here was only to illustrate the method and the main proof ideas that lead to a bound that is polynomial in n^k and $1/\sigma$.

Bibliography

[AMR11] David Arthur, Bodo Manthey, and Heiko Röglin, *Smoothed analysis of the k-means method*, Journal of the ACM **58** (2011), no. 5, 19:1–19:31.

[Aro98] Sanjeev Arora, *Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems.*, Journal of the ACM **45** (1998), no. 5, 753–782.

[AV09] David Arthur and Sergei Vassilvitskii, *Worst-case and smoothed analysis of the ICP algorithm, with an application to the k-means method*, SIAM Journal on Computing **39** (2009), no. 2, 766–782.

[Bat96] John Bather, *A conversation with Herman Chernoff*, Statistical Science **11** (1996), no. 4, 335–350, https://projecteuclid.org/euclid.ss/1032280306.

[BCMR13] Tobias Brunsch, Cornelissen, Bodo Manthey, and Heiko Röglin, *Smoothed analysis of the successive shortest path algorithm*, Proceedings of the 24th ACM-SIAM Symp. on Discrete Algorithms (SODA), 2013, pp. 1180–1189.

[BFP$^+$73] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan, *Time bounds for selection*, Journal of Computer and System Sciences **7** (1973), no. 4, 448–461.

[BR15] Tobias Brunsch and Heiko Röglin, *Improved smoothed analysis of multiobjective optimization*, Journal of the ACM **62** (2015), no. 1, 4:1–4:58.

[BRV07] René Beier, Heiko Röglin, and Berthold Vöcking, *The smoothed number of Pareto optimal solutions in bicriteria integer optimization*, Proceedings of the 12th Intl. Conf. on Integer Programming and Combinatorial Optimization (IPCO), 2007, pp. 53–67.

[BV04a] René Beier and Berthold Vöcking, *Probabilistic analysis of knapsack core algorithms*, Proceedings of the 15th ACM-SIAM Symp. on Discrete Algorithms (SODA), 2004, pp. 468–477.

[BV04b] ———, *Random knapsack in expected polynomial time*, Journal of Computer and System Sciences **69** (2004), no. 3, 306–329.

[BV06a] ———, *An experimental study of random knapsack problems*, Algorithmica **45** (2006), no. 1, 121–136.

[BV06b] ———, *Typical properties of winners and losers in discrete optimization*, SIAM J. Comput. **35** (2006), no. 4, 855–881.

[Che04] Herman Chernoff, *Some reminiscences of my friendship with Herman Rubin*, Institute of Mathematical Statistics Lecture Notes **45** (2004), 1–4, `http://projecteuclid.org/euclid.lnms/1196285375`.

[CKT99] Barun Chandra, Howard J. Karloff, and Craig A. Tovey, *New results on the old k-Opt algorithm for the traveling salesman problem.*, SIAM Journal on Computing **28** (1999), no. 6, 1998–2029.

[CRT05] Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan, *Approximating the minimum spanning tree weight in sublinear time*, SIAM Journal on Computing **34** (2005), no. 6, 1370–1379.

[CS09] Artur Czumaj and Christian Sohler, *Estimating the weight of metric minimum spanning trees in sublinear time*, SIAM Journal on Computing **39** (2009), no. 3, 904–922.

[ERV07] Matthias Englert, Heiko Röglin, and Berthold Vöcking, *Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP*, Proceedings of the 18th ACM-SIAM Symp. on Discrete Algorithms (SODA), 2007, pp. 1295–1304.

[GW94] Michel X. Goemans and David P. Williamson, *.879-approximation algorithms for MAX CUT and MAX 2sat*, Proceedings of the 26th STOC, 1994, pp. 422–431.

[GW95] ———, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, Journal of the ACM **42** (1995), no. 6, 1115–1145.

[Her14] Timon Hertli, *3-SAT faster and simpler - unique-sat bounds for PPSZ hold in general*, SIAM Journal on Computing **43** (2014), no. 2, 718–729.

[HSSW07] Thomas Hofmeister, Uwe Schöning, Rainer Schuler, and Osamu Watanabe, *Randomized algorithms for 3-SAT*, Theory of Computing Systems **40** (2007), no. 3, 249–262.

[IKI00] Mary Inaba, Naoki Katoh, and Hiroshi Imai, *Variance-based k-clustering algorithms by Voronoi diagrams and randomization*, IEICE Trans. Information and Systems **E83-D** (2000), no. 6, 1199–1206.

[JM97] David S. Johnson and Lyle A. McGeoch, *The traveling salesman problem: A case study in local optimization.*, Local Search in Combinatorial Optimization (E. H. L. Aarts and J. K. Lenstra, eds.), John Wiley and Sons, 1997.

[Kar72] Richard M. Karp, *Reducibility among combinatorial problems*, Complexity of Computer Computations, Plenum Press, 1972, pp. 85–104.

[Kar93] David R. Karger, *Global min-cuts in rnc, and other ramifications of a simple min-out algorithm*, Proceedings of the 4th ACM-SIAM Symp. on Discrete Algorithms (SODA), 1993, pp. 21–30.

[Ker89] Walter Kern, *A probabilistic analysis of the switching algorithm for the Euclidean TSP*, Mathematical Programming **44** (1989), no. 2, 213–219.

[KKT91] Christos Kaklamanis, Danny Krizanc, and Thanasis Tsantilas, *Tight bounds for oblivious routing in the hypercube*, Mathematical systems theory **24** (1991), no. 1, 223–232.

[KMN$^+$04] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu, *A local search approximation algorithm for k-means clustering*, Computational Geometry: Theory and Applications **28** (2004), no. 2-3, 89–112.

[Kov15] Péter Kovács, *Minimum-cost flow algorithms: an experimental evaluation*, Optimization Methods and Software **30** (2015), no. 1, 94–127.

[KS96] David R. Karger and Clifford Stein, *A new approach to the minimum cut problem*, Journal of the ACM **43** (1996), no. 4, 601–640.

[KV18] Bernhard Korte and Jens Vygen, *Combinatorial optimization: Theory and algorithms*, Springer, 2018, 6th edition.

[Llo82] Stuart P. Lloyd, *Least squares quantization in PCM*, IEEE Transactions on Information Theory **28** (1982), no. 2, 129–137.

[Lue75] George S. Lueker, *Unpublished manuscript*, 1975, Princeton University.

[Mit70] Dragoslav S. Mitrinović, *Analytic inequalities*, Springer, 1970.

[Mit99] Joseph S. B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems.*, SIAM Journal on Computing **28** (1999), no. 4, 1298–1309.

[MR95] Rajeev Motwani and Prabhakar Raghavan, *Randomized algorithms*, Cambridge, 1995.

[MR09] Bodo Manthey and Heiko Röglin, *Improved smoothed analysis of k-means clustering*, Proceedings of the 20th ACM-SIAM Symp. on Discrete Algorithms (SODA), SIAM, 2009, pp. 461–470.

[MS11] Robin A. Moser and Dominik Scheder, *A full derandomization of Schöning's k-sat algorithm*, Proceedings of the 43rd STOC, 2011, pp. 245–252.

[MTY13] Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto, *Derandomizing the HSSW algorithm for 3-SAT*, Algorithmica **67** (2013), no. 2, 112–124.

[MU05] Michael Mitzenmacher and Eli Upfal, *Probability and computing*, Cambridge, 2005.

[Mut05] S. Muthukrishnan, *Data streams: Algorithms and applications*, Foundations and Trends in Theoretical Computer Science **1** (2005), no. 2, 117–236.

[NI92] Hiroshi Nagamochi and Toshihide Ibaraki, *Computing edge-connectivity in multigraphs and capacitated graphs*, SIAM Journal on Discrete Mathematics **5** (1992), no. 1, 54–66.

[NU69] George L. Nemhauser and Zev Ullmann, *Discrete dynamic programming and capital allocation*, Management Science **15** (1969), 494–505.

[Pap77] Christos H. Papadimitriou, *The Euclidean traveling salesman problem is NP-complete.*, Theoretical Computer Science **4** (1977), no. 3, 237–244.

[Pap91] ———, *On selecting a satisfying truth assignment*, Proceedings of the 32nd FOCS, 1991, pp. 163–169.

[Pit85] Leonard Brian Pitt, *A simple probabilistic approximation algorithm for vertex cover*, Tech. Report YaleU/DCS/TR-404, Yale University, 1985.

[Rei91] Gerhard Reinelt, *TSPLIB – A traveling salesman problem library.*, ORSA Journal on Computing **3** (1991), no. 4, 376–384.

[RT09] Heiko Röglin and Shang-Hua Teng, *Smoothed analysis of multiobjective optimization*, Proceedings of the 50th Ann. IEEE Symp. on Foundations of Computer Science (FOCS), IEEE, 2009, pp. 681–690.

[Sch02] Uwe Schöning, *A probabilistic algorithm for k -sat based on limited local search and restart*, Algorithmica **32** (2002), no. 4, 615–623.

[SG76] Sartaj Sahni and Teofilo F. Gonzalez, *P-complete approximation problems*, Journal of the ACM **23** (1976), no. 3, 555–565.

[ST04] Daniel A. Spielman and Shang-Hua Teng, *Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time.*, Journal of the ACM **51** (2004), no. 3, 385–463.

[SW97] Mechthild Stoer and Frank Wagner, *A simple min-cut algorithm*, Journal of the ACM **44** (1997), no. 4, 585–591.

[Val82] Leslie G. Valiant, *A scheme for fast parallel communication*, SIAM Journal on Computing **11** (1982), no. 2, 350 – 361.

[Vat11] Andrea Vattani, *k-means requires exponentially many iterations even in the plane*, Discrete and Computational Geometry **45** (2011), no. 4, 596–616.

[VB81] Leslie G. Valiant and Gordon J. Brebner, *Universal schemes for parallel communication*, Proceedings of the 13th STOC, 1981, pp. 263 – 277.

[Vit85] Jeffrey Scott Vitter, *Random sampling with a reservoir*, ACM Trans. Math. Softw. **11** (1985), no. 1, 37–57.

[vLS81] Jan van Leeuwen and Anneke A. Schoon, *Untangling a traveling salesman tour in the plane.*, Proceedings of the 7th Intl. Workshop on Graph-Theoretic Concepts in Computer Science (WG), 1981, pp. 87–98.

[Wor15] Thomas Worsch, *Randomisierte Algorithmen, WS 2014/2015*, Available online at `http://liinwww.ira.uka.de/~thw/vl-rand-alg/skript-2014.pdf` (last accessed June 2nd 2016), June 2015, Lecture Notes, in German.

[Zad73] Norman Zadeh, *A bad network problem for the simplex method and other minimum cost flow algorithms*, Mathematical Programming **5** (1973), no. 1, 255–266.