Lecture Notes

Pearls of Algorithms

Part 2: Randomized Algorithms and Probabilistic Analysis

Prof. Dr. Heiko Röglin
Institut für Informatik I

Winter 2011/12

December 5, 2011

Contents

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

Chapter 1

Introduction to Smoothed Analysis

In theoretical computer science, an algorithm is typically judged by its *worst-case performance*. An algorithm with good worst-case performance is very desirable because it performs well on all possible inputs. On the other hand, a bad worst-case performance does not necessarily imply that the algorithm performs also badly in practice. The most prominent example is probably the simplex algorithm for solving linear programs. For most deterministic pivot rules that have been suggested, examples are known showing that in the worst case the simplex algorithm can take an exponential number of steps, but the simplex algorithm is still one of the most competitive algorithms for solving linear programs in practice. It is fast and reliable even for large-scale instances and for the pivot rules that have been shown to require an exponential number of iterations in the worst case. Examples on which the simplex algorithm needs many iterations are quite artificial and occur only very rarely in practice. This behavior is by no means an exceptional property of the simplex algorithm. There are many other examples of algorithms that perform badly in the worst case but well in practice, including algorithms for the knapsack problem and local search heuristics for various problems.

This might motivate to study the *average-case* performance rather than the worst case performance. But average-case analyses are often problematic because it is not clear how to choose a "reasonable" probability distribution on the set of inputs. Many average-case analyses assume a uniform distribution. However, for most problems, instances chosen uniformly at random do not reflect *typical* instances. For example, linear programs that are obtained by choosing each coefficient in the constraint matrix uniformly at random have very special properties with high probability and are very different from linear programs that occur in practical applications. Hence, if one shows that an algorithm works well on such random linear programs, it can still perform badly on typical linear programs that occur in practical applications. One very illustrative example showing the difference between random and real-world instances is given in Figure 1.1.

In order to capture the behavior of algorithms on practical inputs better than it is possible by a worst-case or average-case analysis alone, Spielman and Teng introduced a hybrid of these

Figure 1.1: On the left side, one can see a totally random TV image, which is obviously very different from a typical TV image, as shown on the right side.

two models, which they called *smoothed analysis* [ST04]. The input model in a smoothed analysis consists of two steps: In the first step, an adversary chooses an arbitrary input. After that, in the second step, this input is slightly perturbed at random. On the one hand, the random perturbation rules out artificial worst-case instances. On the other hand, smoothed analysis, unlike average-case analysis, is not dominated by completely random instances since the adversary can approximately determine the structure of the instance. Thus, smoothed analysis circumvents the drawbacks of worst-case and average-case analysis.

One natural way of perturbing linear programs is to add a Gaussian random variable to each coefficient. The magnitude of this perturbation is parametrized by the standard deviation σ. We assume that also in general the perturbation is parametrized by some value σ such that no perturbation occurs for $\sigma = 0$, and the (expected) magnitude of the perturbation grows with σ. The smoothed running time of an algorithm depends on the input size and the perturbation parameter σ, and it is defined to be the worst expected running time that the adversary can achieve. To make this more precise, let $\mathcal{A}$ denote an algorithm, let I denote an input for $\mathcal{A}$, and let $\mathcal{C}_{\mathcal{A}}(I)$ denote a complexity measure of algorithm $\mathcal{A}$ on input I, e.g., its running time on I. Let $\mathcal{I}_n$ denote the set of inputs of length n. The *worst-case complexity* for inputs of length n is defined as

$$\mathcal{C}_{\mathcal{A}}^{\text{worst}}(n) = \max_{I \in \mathcal{I}_n} \left(\mathcal{C}_{\mathcal{A}}(I)\right).$$

Given a probability distribution μ_n on $\mathcal{I}_n$, the *average-case complexity* of $\mathcal{A}$ for inputs of length n is

$$\mathcal{C}_{\mathcal{A}}^{\text{ave}}(n) = \mathbf{E}_{I \sim \mu_n} \left[\mathcal{C}_{\mathcal{A}}(I)\right],$$

where $I \sim \mu_n$ means that I is a random instance chosen according to the distribution μ_n. For an instance I and a magnitude parameter σ, let $\text{per}_\sigma(I)$ denote the random variable that describes the instance obtained from I by a perturbation with magnitude σ, e.g., if I is a linear program, then $\text{per}_\sigma(I)$ is the random linear program obtained from I by adding a Gaussian random variable with standard deviation σ to each coefficient. The *smoothed complexity* of algorithm $\mathcal{A}$ for inputs of length n and magnitude parameter σ is defined as

$$\mathcal{C}_{\mathcal{A}}^{\text{smooth}}(n, \sigma) = \max_{I \in \mathcal{I}_n} \mathbf{E}\left[\mathcal{C}_{\mathcal{A}}\left(\text{per}_\sigma(I)\right)\right].$$

These definitions are illustrated in Figure 1.2.

From the definition of smoothed complexity, one can see that it is a hybrid of worst-case and average-case analysis and that one can interpolate between these kinds of analyses by adjusting the parameter σ: For $\sigma \to 0$, the analysis becomes a worst-case analysis because the input specified by the adversary is not perturbed anymore. For $\sigma \to \infty$, the analysis becomes an average-case analysis because the perturbation is so large that the initial input specified by the adversary is not important anymore. We say that the *smoothed complexity of $\mathcal{A}$ is polynomial* if $\mathcal{C}_{\mathcal{A}}^{\text{smooth}}(n, \sigma)$ is polynomially bounded in n and σ^{-1}. If the smoothed complexity of an algorithm is polynomial, then one can hope that the algorithm performs well in practice, because worst-case instances might exist but they are very fragile with respect to random influences. With other words, if the smoothed complexity of an algorithm is low and instances are subject to random noise, then one must be extremely unlucky to hit a bad instance.

Spielman and Teng [ST04] showed that the smoothed complexity of the simplex algorithm is polynomial for a certain pivot rule. Since the invention of smoothed analysis in 2001, many different results on the smoothed analysis of algorithms have been obtained, including

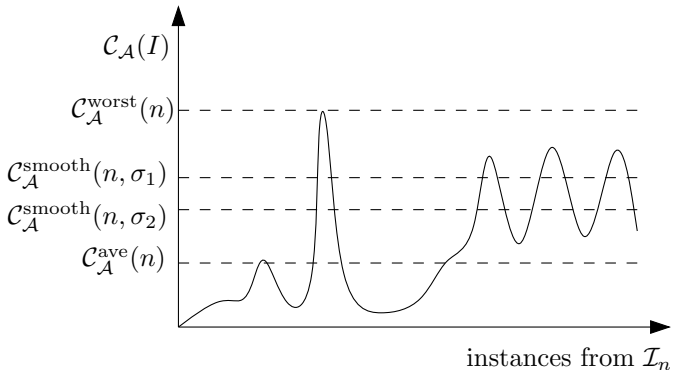

Figure 1.2: Illustration of the different complexity measures. The horizontal axis ranges over the set of inputs of length n, for some fixed n. It is assumed that $\sigma_1 < \sigma_2$. Hence, $C_{\mathcal{A}}^{\text{smooth}}(n,\sigma_1) > C_{\mathcal{A}}^{\text{smooth}}(n,\sigma_2)$.

results on different algorithms for solving linear programs, local search algorithms, various discrete optimizations problems, and the competitive ratio of online algorithms.

In the remainder of this introductory chapter, we first sketch Spielman and Teng's smoothed analysis of the simplex algorithm, and after that we review some facts from probability theory that will be essential for this part of the lecture. In the coming chapters we will discuss some results about the smoothed complexity of algorithms in more detail.

1.1 Smoothed Analysis of the Simplex Algorithm

Linear programming is one of the most important problems in mathematical optimization and operations research. It is interesting from a theoretical point of view because many problems are shown to be polynomial-time solvable by reducing them to a linear programming problem. Moreover, linear programming arises in numerous industrial applications. The importance of linear programming in industrial applications stems in part from the existence of fast and reliable algorithms for finding optimal solutions. In this section, we describe some results on the smoothed complexity of the simplex algorithm for solving linear programs. Since the probabilistic analyses of the simplex algorithm are quite involved, we cannot present them in full detail here. We merely state the main results and very roughly outline their proofs.

In a linear programming problem, one is asked to maximize or minimize a linear function over a polyhedral region. In the following, we assume that the goal is to maximize the linear objective function

$$c^T x = c_1 x_1 + \cdots + c_d x_d$$

subject to $x_1, \ldots, x_d \in \mathbb{R}$ and the n linear constraints

$$\begin{aligned} a_{1,1}x_1 \quad &+\ldots+ \quad a_{1,d}x_d \leq b_1 \\ &\vdots \\ a_{n,1}x_1 \quad &+\ldots+ \quad a_{n,d}x_d \leq b_n, \end{aligned}$$

which we will also express shortly as $Ax \leq b$ for $A \in \mathbb{R}^{n \times d}$ and $b \in \mathbb{R}^n$.

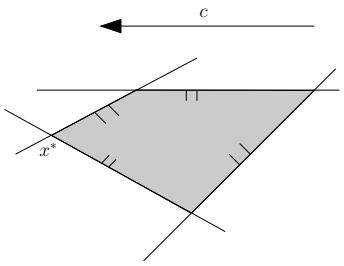

Figure 1.3: Graphical Interpretation of an LP with two variables and four constraints.

Geometry of Linear Programs

The set $\mathcal{P}$ of feasible solutions of a linear program is defined by a set of linear inequalities, i.e., $\mathcal{P} = \left\{x \in \mathbb{R}^d \mid Ax \leq b\right\}$. Let $a_1, \ldots, a_n$ denote the rows of A, and let $b_1, \ldots, b_n$ denote the entries of the vector b, the so-called *right-hand sides*. The set of points from $\mathbb{R}^d$ that satisfy a linear equation $a_i x = b_i$ is called a *hyperplane*. It is a $(d-1)$-dimensional *subspace* of $\mathbb{R}^d$. The set of points from $\mathbb{R}^d$ that satisfy a linear inequality $a_i x \leq b_i$ is called a *halfspace*. Observe that the set of feasible solutions $\mathcal{P}$ is the intersection of n halfspaces, a so-called *polytope*.

In this graphical interpretation, the objective function $c^T x$ determines a direction in the space $\mathbb{R}^d$, and the linear program asks to find a point x^* from the polytope $\mathcal{P}$ that lies as far in the direction c as possible. This is illustrated in Figure 1.3.

Let $\mathcal{HS}$ be a halfspace defined by a hyperplane $\mathcal{H}$. If the polytope $\mathcal{P}$ is entirely contained in $\mathcal{HS}$, then the set $f = \mathcal{P} \cap \mathcal{H}$ is called a *face* of $\mathcal{P}$. In the following, we use the term *vertex* to denote a face of dimension zero (i.e., a point), and we use the term *edge* to denote a face of dimension one (i.e., a line segment).

Algorithms for Linear Programming

The first practical method for solving linear programs was proposed in the late 1940's by Dantzig [Dan63]. Dantzig's *simplex algorithm* walks along neighboring vertices of the polytope $\mathcal{P}$ that is defined by the set of linear inequalities $Ax \leq b$. It is well known that if a linear program is neither infeasible nor unbounded, then there exists a vertex of the polytope that maximizes the objective function. Additionally, every vertex that is locally optimal in the sense that there does not exist a neighboring vertex with larger objective value can be shown to be also globally optimal. For a given initial vertex of the polytope, the simplex algorithm picks in each step a neighboring vertex with better objective value until either a locally optimal solution is found or unboundedness is detected. The initial feasible solution is found by the application of the simplex method to a different linear program for which an initial vertex is known and whose optimal solution is either a vertex of the original polytope defined by $Ax \leq b$ or shows that the linear program is infeasible. The simplex method as described above leaves open the question of which step is made when there is more than one neighboring vertex on the polytope with larger objective value. The policy according to which this decision is made is called the *pivot rule*.

For most deterministic pivot rules that have been suggested, examples are known showing that in the worst case the simplex algorithm can take an exponential number of steps (see,

e.g., [AZ99]). The observations made in practice tell a different story. The simplex algorithm is still one of the most competitive algorithms for solving linear programs that occur in practical applications. It is fast and reliable even for large-scale instances and for the pivot rules that have been shown to require an exponential number of iterations in the worst case. Examples on which the simplex algorithm needs many iterations occur only very rarely in practice.

The question whether optimal solutions of linear programs can be found in polynomial time has been settled in 1979 by Khachian [Kha79]. He applied the *ellipsoid method*, originally developed for solving non-linear optimization problems, to linear programming and proved that it converges in time polynomial in d, n, and L, where L denotes the number of bits needed to represent the linear program. Though from a theoretical point of view a breakthrough, the ellipsoid method is drastically outperformed by the simplex algorithm in practice. The *interior-point method*, another method for solving linear programs with polynomial worst-case complexity, was introduced in 1984 by Karmarkar [Kar84]. In contrast to the ellipsoid method, the interior point method is competitive with and occasionally superior to the simplex algorithm in practice.

Smoothed Linear Programs

Spielman and Teng [ST04] considered linear programs of the form

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & (\overline{A} + G)x \leq (\overline{b} + h), \end{aligned}$$

where $\overline{A} \in \mathbb{R}^{n \times d}$ and $\overline{b} \in \mathbb{R}^n$ are chosen arbitrarily by an adversary and the entries of the matrix $G \in \mathbb{R}^{n \times d}$ and the vector $h \in \mathbb{R}^n$ are independent Gaussian random variables that represent the perturbation. These Gaussian random variables have mean 0 and standard deviation $\sigma \cdot (\max_i \|(\overline{b}_i, \overline{a}_i)\|)$, where the vector $(\overline{b}_i, \overline{a}_i) \in \mathbb{R}^{d+1}$ consists of the i-th component of $\overline{b}$ and the i-th row of $\overline{A}$ and $\|\cdot\|$ denotes the Euclidean norm, that is, for a vector $u = (u_1, \ldots, u_l)$, $\|u\| = \sqrt{u_1^2 + \cdots + u_l^2}$. Without loss of generality, we can scale the linear program specified by the adversary and assume that $\max_i \|(\overline{b}_i, \overline{a}_i)\| = 1$. Then the perturbation consists of adding an independent Gaussian random variable with standard deviation σ to each entry of $\overline{A}$ and $\overline{b}$. Observe that we can replace this two-step model by a one-step model in which each entry is an independent Gaussian random variable and an adversary is allowed to choose the means of these random variables.

The Shadow Vertex Pivot Rule

Spielman and Teng analyzed the smoothed running time of the simplex algorithm using the *shadow vertex pivot rule*. This pivot rule has been proposed by Gass and Saaty [GS55] and it has a simple and intuitive geometric description which makes probabilistic analyses feasible. Let x_0 denote the given initial vertex of the polytope $\mathcal{P}$ of feasible solutions. Since x_0 is a vertex of the polytope, there exists an objective function $u^T x$ which is maximized by x_0 subject to the constraint $x \in \mathcal{P}$. In the first step, the shadow vertex pivot rule, computes an objective function $u^T x$ with this property. Using standard arguments from analytic geometry, one can show that such an objective function can be found efficiently. If x_0 is not the optimal solution of the linear program, then the vectors c and u are linearly independent and span a plane. The shadow vertex method projects the polytope $\mathcal{P}$ onto this

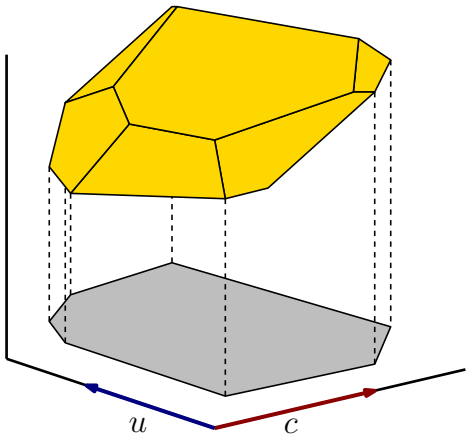

Figure 1.4: The polytope is projected onto the two-dimensional plane spanned by the vectors c and u.

plane. The *shadow*, that is, the projection, of $\mathcal{P}$ onto this plane is a possibly open polygon (see Figure 1.4). This polygon has a few useful properties:

- The vertex x_0 is projected onto a vertex of the polygon.
- The optimal solution x^* is projected onto a vertex of the polygon.
- Each vertex of the polygon is the image of a vertex of the polytope.
- Each edge of the polygon is the image of an edge between two adjacent vertices of the polytope.

Observe that the simplex algorithm in dimension two is very easy; it just follows the edges of the polygon. Due to the aforementioned properties, we can apply the two-dimensional simplex algorithm to the polygon obtained by the projection, and the walk along the edges of the polygon corresponds to a walk along the edges of the original polytope. Furthermore, once the optimal solution on the polygon is found, we can compute its pre-image on the polytope, which is an optimal solution of the linear program.

The number of steps performed by the simplex algorithm with shadow vertex pivot rule is upper bounded by the number of vertices of the two-dimensional projection of the polytope. Hence, bounding the expected number of vertices on the polygon is the crucial step for bounding the expected running time of the simplex algorithm with shadow vertex pivot rule. Spielman and Teng consider first the case that the polytope $\mathcal{P}$ is projected onto a fixed plane specified by two fixed vectors c and u. They show the following result on the expected *shadow size*, that is, the number of vertices of the polygon.

Theorem 1.1 ([ST04]). *Let $c \in \mathbb{R}^d$ and $u \in \mathbb{R}^d$ be independent vectors, and let $a_1, \ldots, a_n \in \mathbb{R}^d$ be independent Gaussian random vectors of standard deviation σ centered at points of norm at most 1. Let $\mathcal{P} = \left\{x \in \mathbb{R}^d \mid \forall i \in \{1, \ldots, n\} : a_i x \leq 1\right\}$ denote the polytope of feasible solutions. The number of vertices of the polygon obtained by projecting $\mathcal{P}$ onto the plane spanned by c and u is*

$$O\left(\frac{nd^3}{\min(\sigma, 1/\sqrt{d \ln n})^6}\right) = O\left(\text{poly}\left(n, d, \frac{1}{\sigma}\right)\right).$$

Spielman and Teng's Result

Though Theorem 1.1 is the main ingredient of the analysis, alone it does not yield a polynomial bound on the smoothed running time of the simplex algorithm. There are three main obstacles that one has to overcome. First, we have not yet described how the initial feasible solution is found. Since testing feasibility of a linear program and finding an arbitrary feasible solution are computationally as hard as solving a linear program to optimality (see, e.g., [PS98]), this question cannot be neglected. The last two problems that have to be addressed concern the assumptions in Theorem 1.1. It is assumed that the right-hand sides in the constraints are all 1 and furthermore, it is assumed that the vector u is fixed independently of the constraints. Both assumptions are not satisfied in the probabilistic model we consider. Spielman and Teng have shown in a very involved analysis that Theorem 1.1 can be applied, regardless of these technical difficulties. Their proof is, however, far too complex to present it in detail here.

Theorem 1.2 ([ST04])**.** *Let $c \in \mathbb{R}^d$ be chosen arbitrarily, let $a_1, \dots, a_n \in \mathbb{R}^d$ be independent Gaussian random vectors centered at $\overline{a}_1, \dots, \overline{a}_n$, and let $b_1, \dots, b_n$ be independent Gaussian random variables centered at $\overline{b}_1, \dots, \overline{b}_n$. Let the standard deviation of the Gaussian vectors and variables be $\sigma \cdot \max_i \|(\overline{b}_i, \overline{a}_i)\|$. Then there exists a polynomial P and a constant σ_0 such that for all $\sigma < \sigma_0$, $c \in \mathbb{R}^d$, $\overline{a}_1, \dots, \overline{a}_n \in \mathbb{R}^d$, and $\overline{b} \in \mathbb{R}^n$, the expected running time of the shadow vertex simplex method on the linear program $\max c^T x$ subject to $Ax \leq b$ is at most $P(n, d, 1/\sigma)$.*

Later, the smoothed analysis of the simplex algorithm was substantially improved by Vershynin [Ver06].

Theorem 1.3 ([Ver06])**.** *Under the same assumptions as in Theorem 1.2, the expected number of pivot steps of the shadow vertex simplex method is at most*

$$O\left(\max\left(d^5 \log^2 n, d^9 \log^4 d, d^3 \sigma^{-4}\right)\right) = O\left(\operatorname{poly}\left(\log n, d, \frac{1}{\sigma}\right)\right).$$

This is a remarkable result because the expected number of pivot steps is only polylogarithmic in the number of constraints n while the previous bound was polynomial in n.

1.2 Mathematical Background

In this section, we introduce some notation and review some facts from probability theory. This section is by no means intended to give an introduction to probability theory. Readers who are unfamiliar with probability theory are referred to [MR95] and [MU05] for computer science related introductions.

Notation

We define $\mathbb{N} = \{1, 2, 3, \dots\}$. For a natural number $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \dots, n\}$. We use $\mathbb{R}_+$ to denote the set $\{x \in \mathbb{R} \mid x \geq 0\}$. Given a vector $x \in \mathbb{R}^n$, we use $x_1, \dots, x_n$ to denote its entries, i.e., we assume $x = (x_1, \dots, x_n)$. Given two vectors $x, y \in \mathbb{R}^n$, we denote by $x \cdot y$ their *dot product*, i.e., $x \cdot y = x_1 y_1 + \cdots + x_n y_n$. The *norm* $\|x\|$ of a vector $x \in \mathbb{R}^n$ is always meant to be its Euclidean norm, i.e., $\|x\| = \sqrt{x_1^2 + \cdots + x_n^2} = \sqrt{x \cdot x}$.

Expected Values

Let X be a discrete random variables that takes only values in a countable set $M \subset \mathbb{R}$ (e.g., $M = \mathbb{N}$). For an element $a \in M$, we denote by $\mathbf{Pr}[X = a]$ the *probability* that X takes value a. The *expected value* of X is defined as

$$\mathbf{E}[X] = \sum_{a \in M} a \cdot \mathbf{Pr}[X = a].$$

If $M \subseteq \mathbb{N}$, we can also write this as

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

If X describes, for example, the outcome of a dice toss, we have $M = \{1, 2, \ldots, 6\}$ and $\mathbf{Pr}[X = a] = 1/6$ for each $a \in M$. Then the expected value of X is

$$\mathbf{E}[X] = \sum_{a=1}^{6} \frac{a}{6} = 3.5.$$

Let X and Y be two random variables that take only values in $M \subset \mathbb{R}$. We call X and Y *independent* if for any $a, b \in M$

$$\mathbf{Pr}[X = a \text{ and } Y = b] = \mathbf{Pr}[X = a] \cdot \mathbf{Pr}[Y = b].$$

A very helpful fact from probability theory is *linearity of expectation*. If X and Y are random variables, then

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

This equations holds not only for independent random variables X and Y, but also for dependent random variables.

Useful Inequalities

- The *Markov inequality* is one of the easiest tools to bound the probability that a random variable takes a large value: Let X be a random variable that takes only non-negative values in $\mathbb{R}_+$, and let $a \geq 1$. Then

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

- For random variables that are the sum of independent 0-1-random variables, the *Chernoff bound* yields a much better bound than the Markov inequality: Let $X_1, \ldots, X_n$ be independent random variables that take only values in $\{0, 1\}$, and let $p_i := \mathbf{Pr}[X_i = 1]$. Furthermore let $X = \sum_{i=1}^{n} X_i$. Then, by linearity of expectation,

$$\mathbf{E}[X] = \sum_{i=1}^{n} \mathbf{E}[X_i] = \sum_{i=1}^{n} (0 \cdot \mathbf{Pr}[X_i = 0] + 1 \cdot \mathbf{Pr}[X_i = 1]) = \sum_{i=1}^{n} p_i.$$

  Then, for any $0 < \delta < 1$,

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

- Let $\mathcal{F}_1, \ldots, \mathcal{F}_n$ be some events. The *union bound* gives an upper bound for the probability that at least one of these events occurs:

$$\mathbf{Pr}[\mathcal{F}_1 \cup \cdots \cup \mathcal{F}_n] \leq \sum_{i=1}^{n} \mathbf{Pr}[\mathcal{F}_i].$$

Continuous Random Variables

Now we consider continuous random variables that are not limited to a countable subset of $\mathbb{R}$. The *distribution* $F_X \colon \mathbb{R} \to [0,1]$ of a *real-valued random variable* X is the function defined by $F_X(x) = \mathbf{Pr}[X \leq x]$ for all $x \in \mathbb{R}$. If F_X is differentiable, then the derivative $f_X \colon \mathbb{R} \to \mathbb{R}_+$ of F_X is called the *density function* of X. For every $x \in \mathbb{R}$, it holds

$$F_X(x) = \mathbf{Pr}[X \leq x] = \int_{-\infty}^{x} f_X(t)\,dt.$$

For every $a, b \in \mathbb{R}$ with $a \leq b$, we have

$$\mathbf{Pr}[X \in [a,b]] = \int_{a}^{b} f_X(t)\,dt.$$

This immediately yields the following observation.

Observation 1.4. *Let X be a random variable whose density is bounded from above by $\phi > 0$, and let $I = [a, a+\varepsilon]$ denote an arbitrarily fixed interval of length $\varepsilon > 0$. The probability that X takes a value in the interval I is bounded from above by $\varepsilon\phi$.*

For a continuous random variable X with density f_X, the expected value is defined as

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

Linearity of expectation and the Markov inequality are also valid for continuous random variables.

Of particular interest are *Gaussian random variables*. A Gaussian random variable with expected value $\mu \in \mathbb{R}$ and standard deviation $\sigma > 0$ has density

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right). \tag{1.1}$$

We call the expected value of a Gaussian random variable its *mean* or *center*. We also use the terms *d-dimensional Gaussian (random) vector* and *Gaussian (random) vector in $\mathbb{R}^d$* with standard deviation σ to denote a d-dimensional vector whose entries are independent Gaussian random variables with standard deviation σ. We say that a Gaussian vector in $\mathbb{R}^d$ has *center* $\mu \in \mathbb{R}^d$ if, for $i \in [d]$, its i-th entry is a Gaussian random variable with mean μ_i.

Gaussian random variables are sharply concentrated around their centers, as the following lemma shows.

Lemma 1.5. *Suppose X is a Gaussian random variable with center 0 and standard deviation σ. Then, for every $t \geq 1$,*

$$\mathbf{Pr}[|X| > t] < \sigma \cdot \exp\left(-\frac{t^2}{2\sigma^2}\right).$$

Proof. For $t \geq 1$, we obtain

$$\begin{aligned}
\mathbf{Pr}\left[|X| \geq t\right] &= 2 \cdot \int_{z=t}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{z^2}{2\sigma^2}\right) dz \\
&< \int_{z=t}^{\infty} \frac{z}{\sigma} \exp\left(-\frac{z^2}{2\sigma^2}\right) dz \\
&= \frac{1}{\sigma}\left[-\sigma^2 \cdot \exp\left(-\frac{z^2}{2\sigma^2}\right)\right]_{z=t}^{\infty} \\
&= \sigma \cdot \exp\left(-\frac{t^2}{2\sigma^2}\right).
\end{aligned}$$

□

From Lemma 1.5, we can conclude the following corollary.

Corollary 1.6. *Suppose X is a Gaussian random variable with arbitrary center $\mu \in \mathbb{R}$ and standard deviation σ. Then, for every $t \geq 1$,*

$$\mathbf{Pr}\left[|X - \mu| > t\right] < \sigma \cdot \exp\left(-\frac{t^2}{2\sigma^2}\right).$$

Chapter 2

Smoothed Analysis of the 2-Opt Algorithm for the TSP

2.1 Preliminaries

An instance of the *traveling salesperson problem (TSP)* consists of a set $V = \{v_1, \dots, v_n\}$ of *vertices* (depending on the context, synonymously referred to as *points*) and a symmetric *distance function* $\operatorname{dist}\colon V \times V \to \mathbb{R}_+$ that associates with each pair $\{v_i, v_j\}$ of distinct vertices a distance $\operatorname{dist}(v_i, v_j) = \operatorname{dist}(v_j, v_i)$. 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 $\operatorname{dist}\colon V \times V \to \mathbb{R}_+$ is called a *metric space* if for all $x, y, z \in V$ the following properties are satisfied:

(a) $\operatorname{dist}(x, y) = 0$ if and only if $x = y$ *(reflexivity)*,

(b) $\operatorname{dist}(x, y) = \operatorname{dist}(y, x)$ *(symmetry)*,

(c) $\operatorname{dist}(x, z) \leq \operatorname{dist}(x, y) + \operatorname{dist}(y, z)$ *(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 $\operatorname{dist}_p(x, y)$ of two points $x \in \mathbb{R}^d$ and $y \in \mathbb{R}^d$ with respect to the L_p metric is given by $\operatorname{dist}_p(x, y) = \sqrt[p]{|x_1 - y_1|^p + \cdots + |x_d - y_d|^p}$. 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 $\operatorname{dist}_\infty(x, y) = \max\{|x_1 - y_1|, \dots, |x_d - y_d|\}$. 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.

A *tour construction heuristic* for the TSP incrementally constructs a tour and stops as soon as a valid tour is created. Usually, a tour constructed by such a heuristic is used as the initial solution 2-Opt starts with. A well-known class of tour construction heuristics for metric TSP instances are so-called *insertion heuristics*. These heuristics insert the vertices into the tour one after another, and every vertex is inserted between two consecutive vertices in the current tour where it fits best. To make this more precise, let T_i denote a subtour on a subset S_i of i vertices, and suppose $v \notin S_i$ is the next vertex to be inserted. If (x, y) denotes an edge in

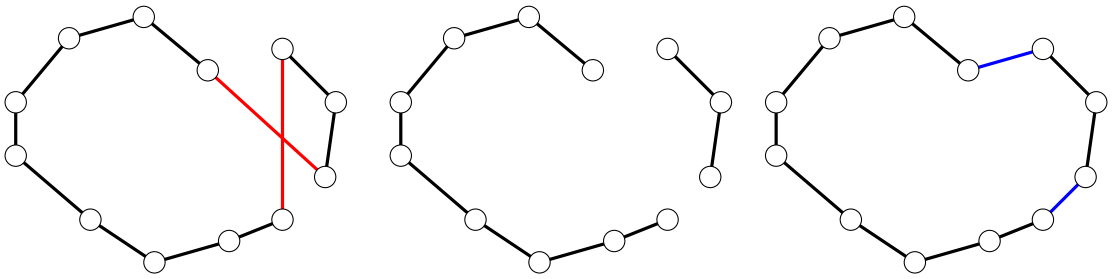

Figure 2.1: Example of a 2-change in which the red edges are exchanged with the blue edges.

T_i that minimizes $\text{dist}(x,v) + \text{dist}(v,y) - \text{dist}(x,y)$, then the new tour T_{i+1} is obtained from T_i by deleting the edge (x,y) and adding the edges (x,v) and (v,y). Depending on the order in which the vertices are inserted into the tour, one distinguishes between several different insertion heuristics. Rosenkrantz et al. [RSI77] show an upper bound of $\lceil \log n \rceil + 1$ on the approximation factor of any insertion heuristic on metric TSP instances. Furthermore, they show that two variants which they call *nearest insertion* and *cheapest insertion* achieve an approximation ratio of 2 for metric TSP instances. The nearest insertion heuristic always inserts the vertex with the smallest distance to the current tour (i.e., the vertex $v \notin S_i$ that minimizes $\min_{x \in S_i} \text{dist}(x,v)$), and the cheapest insertion heuristic always inserts the vertex whose insertion leads to the cheapest tour T_{i+1}.

2.2 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 observations. One important special case is the *Euclidean TSP* in which the vertices are points in $\mathbb{R}^d$, for some $d \in \mathbb{N}$, 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 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 most basic local search heuristic for the TSP. 2-Opt starts with an arbitrary initial tour and incrementally improves this tour by making 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 2.1). 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-Opt step. 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 2.1 ([ERV07])**.** *For every $p \in \{1, 2, 3, \ldots\} \cup \{\infty\}$ and $n \in \mathbb{N} = \{1, 2, 3, \ldots\}$, 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 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 $\phi \geq 1$. Our 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 since the larger ϕ is, 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. These results narrow the gap between the subquadratic number of improving steps observed in experiments [JM97] and the upper bounds from the probabilistic analysis.

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 model discussed above. It is assumed that the dimension $d \geq 2$ is an arbitrary constant.

Theorem 2.2 ([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.

Usually, 2-Opt is initialized with a tour computed by some tour construction heuristic. One particular class is that of *insertion heuristics*, which insert the vertices one after another into the tour. We show that also from a theoretical point of view, using such an insertion heuristic yields a significant improvement for metric TSP instances because the initial tour 2-Opt starts with is much shorter than the longest possible tour. This observation leads to the following theorem.

Theorem 2.3 ([ERV07]). *The expected number of steps performed by 2-Opt*

a) is $O(n^{4-1/d} \cdot \log n \cdot \phi)$ on ϕ-perturbed Manhattan instances with n points when 2-Opt is initialized with a tour obtained by an arbitrary insertion heuristic.

b) is $O(n^{4+1/3-1/d} \cdot \log^2(n\phi) \cdot \phi^{8/3})$ on ϕ-perturbed Euclidean instances with n points when 2-Opt is initialized with a tour obtained by an arbitrary insertion heuristic.

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 2.4 ([ERV07]). *Let $p \in \mathbb{N} \cup \{\infty\}$. For ϕ-perturbed L_p instances, the expected approximation ratio of the worst tour that is locally optimal for 2-Opt is $O(\sqrt[d]{\phi})$.*

2.3 Smoothed Analysis of 2-Opt

Is this section, we present a glimpse into the proof of Theorem 2.2. 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 the theorem in a simplified random input model. Instead of choosing n points at random, we assume that an undirected graph $G = (V, E)$ is given. An adversary can specify an arbitrary density $f_e : [0, 1] \to [0, \phi]$ for every edge $e \in E$. Then the length $\mathrm{dist}(e)$ of edge $e \in E$ is drawn independently from the other edge lengths according to the density f_e. For this input model, we prove the following theorem.

Theorem 2.5. *For any graph with n vertices, the expected length of the longest path in the 2-Opt state graph is $O(n^6 \log(n) \cdot \phi)$.*

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 knew 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 can show that the smallest possible improvement Δ is not too small, we have also shown 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, however, to show that on ϕ-perturbed instances 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

$$\Delta(e_1, e_2, e_3, e_4) = \operatorname{dist}(e_1) + \operatorname{dist}(e_2) - \operatorname{dist}(e_3) - \operatorname{dist}(e_4). \tag{2.1}$$

We define Δ as the smallest possible improvement made by any improving 2-change:

$$\Delta = \min_{\substack{e_1,e_2,e_3,e_4 \\ \Delta(e_1,e_2,e_3,e_4)>0}} \Delta(e_1, e_2, e_3, e_4).$$

The following lemma, which is proven below, is one crucial ingredient of the proof.

Lemma 2.6. *For any $\varepsilon > 0$,*

$$\mathbf{Pr}\left[\Delta \leq \varepsilon\right] \leq n^4 \varepsilon \phi.$$

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. Formally, let T denote the length of the longest path in the state graph. This number can only be greater or equal to t if

$$\frac{n}{\Delta} \geq t \quad \Longleftrightarrow \quad \Delta \leq \frac{n}{t}.$$

Hence,

$$\mathbf{Pr}\left[T \geq t\right] \leq \mathbf{Pr}\left[\Delta \leq \frac{n}{t}\right] \leq \frac{n^5 \phi}{t}.$$

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:

$$\mathbf{E}\left[T\right] = \sum_{t=1}^{n!} \mathbf{Pr}\left[T \geq t\right] \leq \sum_{t=1}^{n!} \frac{n^5 \phi}{t} = n^5 \phi \cdot \sum_{t=1}^{n!} \frac{1}{t}.$$

Using that

$$\sum_{t=1}^{n!} \frac{1}{t} = O(\log(n!)) = O(n \log(n))$$

yields

$$\mathbf{E}\left[T\right] = O(n^5 \phi \log(n!)) = O(n^6 \log(n) \cdot \phi). \qquad \square$$

To complete the proof of the Theorem, we have to prove Lemma 2.6.

Proof of Lemma 2.6. 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 the 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:

$$\mathbf{Pr}\left[\Delta(e_1, e_2, e_3, e_4) \in (0, \varepsilon]\right] = \mathbf{Pr}\left[\mathrm{dist}(e_1) + \mathrm{dist}(e_2) - \mathrm{dist}(e_3) - \mathrm{dist}(e_4) \in (0, \varepsilon]\right].$$

We use the principle of deferred decisions and assume that the lengths $\mathrm{dist}(e_2)$, $\mathrm{dist}(e_3)$, and $\mathrm{dist}(e_4)$ have already been fixed arbitrarily. Then the event $\Delta(e_1, e_2, e_3, e_4) \in (0, \varepsilon]$ is equivalent to the event that

$$\mathrm{dist}(e_1) \in (\kappa, \kappa + \varepsilon],$$

where $\kappa = \mathrm{dist}(e_4) + \mathrm{dist}(e_3) - \mathrm{dist}(e_2)$ is some fixed value. As $\mathrm{dist}(e_1)$ is a random variable whose density is bounded from above by ϕ, the probability that $\mathrm{dist}(e_1)$ 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 make a statement about every possible 2-change. We apply a union bound over all possible choices for the edges $e_1, \ldots, e_4$. As these edges are determined by four vertices, the number of choices is bounded from above by n^4, yielding

$$\mathbf{Pr}\left[\Delta \in (0, \varepsilon]\right] \leq \mathbf{Pr}\left[\exists e_1, e_2, e_3, e_4 : \Delta(e_1, e_2, e_3, e_4) \in (0, \varepsilon]\right] \leq n^4 \varepsilon \phi.$$

This concludes the proof of the lemma. □

Chapter 3

Smoothed Analysis of the Knapsack Problem

The *knapsack problem* is one of the classical NP-hard optimization problems. An instance of the problem consists of a capacity and n objects, each of which having a profit and a weight. The goal is to find a subset of the objects that obeys the capacity constraint and maximizes the profit. To make this precise, let $t \in \mathbb{R}_+$ denote the capacity, let $p = (p_1, \ldots, p_n) \in \mathbb{R}_+^n$ denote a vector of profits, and $w = (w_1, \ldots, w_n) \in \mathbb{R}_+^n$ a vector of weights. The goal is to find a vector $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ such that the objective function

$$p \cdot x = p_1 x_1 + \cdots + p_n x_n$$

is maximized under the constraint

$$w \cdot x = w_1 x_1 + \cdots + w_n x_n \leq t. \tag{3.1}$$

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. Theorists 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 only one linear constraint. On the other hand, knapsack-like problems often occur in practical applications, and hence practitioners have developed numerous heuristics for solving it. These heuristics work very well on random and real-world instances of the knapsack problem, and they find optimal solutions rather quickly. Hence, despite being NP-hard, the knapsack problem is easy to solve in practice. In this chapter, we present one heuristic for the knapsack problem and show that its smoothed complexity is polynomial.

3.1 The Nemhauser/Ullmann Algorithm

In the following, we use the term *solution* to denote a vector $x \in \{0,1\}^n$, and we say that a solution is *feasible* if it obeys the capacity constraint given in (3.1). We say that a solution x *contains the i-th object* if $x_i = 1$. One naive approach for solving the knapsack problem is to enumerate all feasible knapsack solutions and to select the solution with maximum payoff. Of course, this approach is not efficient as there are typically exponentially many feasible solutions. In order to decrease the number of solutions that are enumerated, we observe that a solution x cannot be optimal if it is *dominated* by another solution x', i.e., if the profit of x' is larger than the profit of x and the weight of x' is smaller than the weight of x. Hence, it suffices to enumerate only those solutions that are not dominated by other solutions, the *Pareto optimal* solutions.

Definition 3.1. *A solution x is called* Pareto optimal *if there does not exist a solution x' such that $p \cdot x \leq p \cdot x'$ and $w \cdot x \geq w \cdot x'$ with one inequality being strict. The* Pareto set *is the set of all Pareto optimal solutions.*

Nemhauser and Ullmann [NU69] proposed an algorithm for enumerating the Pareto set of a given knapsack instance. The running time of this algorithm is polynomially bounded in the size of the instance and the size of the Pareto set. That is, 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. Hence, not surprisingly, the Nemhauser/Ullmann algorithm is not polynomial in the worst case, but it works reasonably well in practice.

For a given knapsack instance with n objects, we consider, for $i \in \{0, \dots, n\}$, the modified instance in which only the first i objects are allowed to be inserted into the knapsack. We denote by $\mathcal{P}(i)$ the Pareto set of this modified instance, e.g., $\mathcal{P}(0)$ contains only the solution 0^n and $\mathcal{P}(n)$ is the Pareto set of the given instance. The algorithm that Nemhauser and Ullmann propose computes the sets $\mathcal{P}(0), \dots, \mathcal{P}(n)$ inductively. Since $\mathcal{P}(0)$ is easy to compute, we can assume that a set $\mathcal{P}(i-1)$ is given and that the goal is to compute $\mathcal{P}(i)$. Furthermore, we assume that the solutions in $\mathcal{P}(i-1)$ are sorted in non-decreasing order of their weights. We denote by $\mathcal{P}(i-1)+i$ the set of solutions that is obtained by adding the i-th object to each solution from $\mathcal{P}(i-1)$. Due to the following observation, $\mathcal{P}(i)$ must be a subset of $\mathcal{P}(i)' = \mathcal{P}(i-1) \cup (\mathcal{P}(i-1)+i)$.

Observation 3.2. *Let $x \in \mathcal{P}(i)$. If x does not contain the i-th object, then $x \in \mathcal{P}(i-1)$. If x contains the i-th object, then the solution obtained from x by removing the i-th object belongs to $\mathcal{P}(i-1)$.*

Since this observation implies that $\mathcal{P}(i)$ is a subset of $\mathcal{P}(i)'$, the set $\mathcal{P}(i)$ can be computed by computing $\mathcal{P}(i)'$ and then removing all solutions that are dominated by other solutions from $\mathcal{P}(i)'$. The set $\mathcal{P}(i)'$ is obtained by merging the two sets $\mathcal{P}(i-1)$ and $\mathcal{P}(i-1)+i$. Both of these sets are sorted in non-decreasing order of weights due to our assumption on $\mathcal{P}(i-1)$. Hence, we can compute $\mathcal{P}(i)'$ in linear time with respect to the size of $\mathcal{P}(i-1)$ such that it is also sorted in non-decreasing order of weights. Given this order of solutions in $\mathcal{P}(i)'$, the set $\mathcal{P}(i)$ of Pareto optimal solutions can be found in linear time. Summarizing, the Nemhauser/Ullmann algorithm can be formulated as follows:

Algorithm 1 The Nemhauser/Ullmann algorithm

Set $\mathcal{P}(0) = \{0^n\}$.
for $i = 1, \dots, n$ **do**
  Merge $\mathcal{P}(i-1)$ and $\mathcal{P}(i-1)+i$ into $\mathcal{P}(i)'$...
    ...such that $\mathcal{P}(i)'$ is sorted in non-decreasing order of weights.
  $\mathcal{P}(i) = \{x \in \mathcal{P}(i)' \mid \nexists x' \in \mathcal{P}(i)' \colon x' \text{ dominates } x\}$.
return $x \in \mathcal{P}(n)$ with $p \cdot x = \max\{p \cdot y \mid y \in \mathcal{P}(n) \wedge w \cdot y \leq t\}$.

For the purpose of illustration and a better understanding, let us take a different view on the algorithm. For $i \in [n]$, let $f_i \colon \mathbb{R} \to \mathbb{R}$ be a mapping from weights to profits such that $f_i(x)$ is the maximum profit over all solutions in $\mathcal{P}(i)$ with weight at most x. Observe that f_i is a non-decreasing step function changing its value only at those weights that correspond to

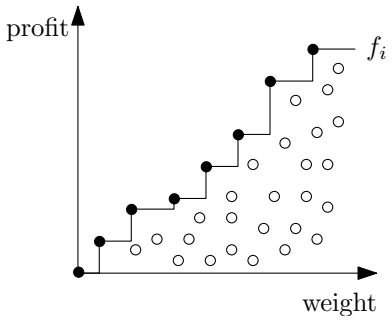

Figure 3.1: The dots correspond to solutions that contain only a subset of the first i elements. Black dots correspond to solutions from $\mathcal{P}(i)$.

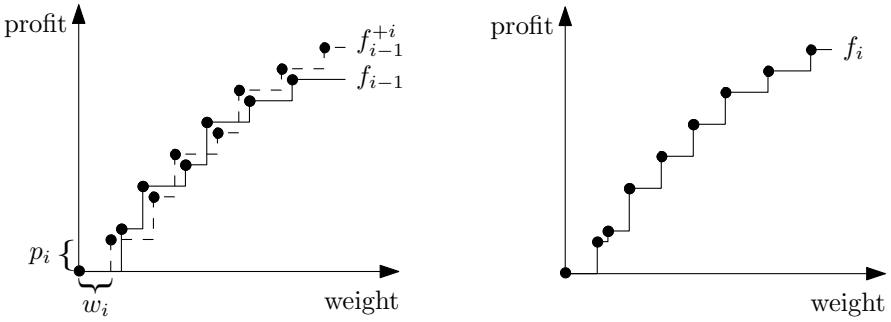

Figure 3.2: The function f_i is the upper envelope of the functions f_{i-1} and f_{i-1}^{+i}.

solutions from $\mathcal{P}(i)$. In particular, the number of steps of f_i equals the number of solutions in $\mathcal{P}(i)$. Figure 3.1 shows such a step function.

Now we describe how one can construct f_i if f_{i-1} is known. Therefore, observe that the set $\mathcal{P}(i-1)+i$ corresponds to a function f_{i-1}^{+i} which is a copy of f_{i-1} that is shifted by (w_i, p_i). The function f_i is the upper envelope of the functions f_{i-1} and f_{i-1}^{+i} (see Figure 3.2).

We have already argued that the time it takes to compute $\mathcal{P}(i)$ from $\mathcal{P}(i-1)$ is linear in the size of $\mathcal{P}(i-1)$. This yields the following lemma.

Lemma 3.3. *For $i \in \{0, \ldots, n-1\}$, we set $q(i) = |\mathcal{P}(i)|$. The running time of the Nemhauser/Ullmann algorithm is bounded from above by*

$$O\left(\sum_{i=0}^{n-1} q(i)\right).$$

If the number $q(i)$ of Pareto optimal solutions does not decrease with i, i.e., $q(0) \leq q(1) \leq \ldots \leq q(n)$, then the running time of the Nemhauser/Ullmann algorithm simplifies to $O(n \cdot q(n))$. That is, the running time depends linearly on the number of Pareto optimal solutions.

3.2 Probabilistic Input Model

Our goal is to analyze the expected number of Pareto optimal solutions in a smoothed input model in which an adversary specifies an arbitrary instance of the knapsack problem

which is subsequently perturbed at random. Since we are only interested in the number of Pareto optimal solutions, the capacity is not important and we can assume that the adversary specifies only the profits and weights of the objects. In our analysis it is not necessary that both the profits and the weights are perturbed, and hence we strengthen the adversary by assuming that only the weights are perturbed. As the running time of the Nemhauser/Ullmann algorithm depends linearly on the number of Pareto optimal solutions, a bound on the expected number of Pareto optimal solutions directly implies a bound on the expected running time of the algorithm and hence on its smoothed complexity.

In the introduction, we have argued that a linear program can be perturbed by adding a Gaussian random variable to each coefficient. In principle, we can use the same perturbation model also for the knapsack problem, that is, each weight is perturbed by adding an independent Gaussian random variable. In this perturbation model, weights can, however, become negative. In order to avoid this problem, we consider a more general perturbation model. First of all, note that we can describe the two-step input model for linear programs as a one-step model. Instead of saying that an adversary specifies a coefficient which is perturbed by adding a Gaussian random variable with standard deviation σ, we say that the adversary is allowed to choose a probability distribution for each coefficient according to which it is chosen. In the input model for linear programs, the adversary is restricted to probability distributions that correspond to Gaussian random variables with standard deviation σ, that is, the adversary can only determine the mean of the random variables.

In our perturbation model for the knapsack problem, the adversary is not restricted to Gaussian distributions. Of course, we cannot allow the adversary to specify arbitrary distributions for the weights because this would allow deterministic inputs as a special case. We restrict the adversary to distributions that can be represented by densities that are bounded by some value ϕ. To make this formal, we assume that for each weight w_i a probability density $f_i \colon \mathbb{R} \to [0, \phi]$ is given, and that each weight w_i is chosen independently according to density f_i. The adversary could, for instance, choose for each coefficient an arbitrary interval of length $1/\phi$ from which it is chosen uniformly at random. The larger the parameter ϕ is chosen, the more concentrated the random variables can be. Hence, analogously to σ^{-1} for Gaussian distributions, the larger the parameter ϕ is chosen, the closer is the smoothed analysis to a worst-case analysis. For Gaussian perturbations, we have $\phi = (\sigma\sqrt{2\pi})^{-1}$.

3.3 The Expected Number of Pareto Optimal Solutions

In this section, we show that the expected number of Pareto optimal solutions for the knapsack problem is polynomially bounded in n and ϕ. As argued above, this directly implies that the expected running time of the Nemhauser/Ullmann algorithm is polynomially bounded in n and ϕ as well. For the sake of simplicity, we assume that all weights lie in the interval $[0, 1]$, that is, for all $x \notin [0, 1]$ and for all $i \in [n]$, we have $f_i(x) = 0$. Beier et al. [BRV07] show that this restriction is not necessary, but we present only a simplified version of their result here.

Theorem 3.4 ([BRV07]). *For an instance of the knapsack problem with n objects whose profits are specified arbitrarily and whose weights are chosen independently according to densities $f_1, \ldots, f_n$ with $f_i \colon [0, 1] \to [0, \phi]$, the expected number of Pareto optimal solutions is upper bounded by $\phi n^2 + 1$.*

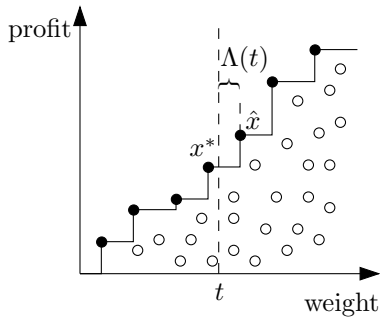

Figure 3.3: Definitions of the winner x^*, the loser $\hat{x}$, and the random variable $\Lambda(t)$.

Proof. We denote the set of Pareto optimal solutions by $\mathcal{P}$ and its size by q, i.e., $q = |\mathcal{P}|$. Every solution has a weight in the interval $[0, n]$ because the weights of the objects lie in the interval $[0, 1]$. Since in the probabilistic input model no two solutions have exactly the same weight, we can partition the interval $[0, n]$ into small intervals such that each of the small intervals contains at most one Pareto optimal solution. Formally, we can write the expected number of Pareto optimal solutions as

$$\mathbf{E}[q] = 1 + \lim_{k\to\infty} \sum_{i=0}^{k-1} \mathbf{Pr}\left[\exists x \in \mathcal{P}\colon w \cdot x \in \left(\frac{ni}{k}, \frac{n(i+1)}{k}\right]\right], \tag{3.2}$$

where the additional 1 corresponds to the solution 0^n, which always belongs to $\mathcal{P}$. In order to estimate the probability in (3.2), we consider the case that an arbitrary $t \in [0, n]$ and an arbitrary $\varepsilon > 0$ are given, and we bound the probability that there exists a Pareto optimal solution with weight in the interval $(t, t+\varepsilon]$. For this, we define a random variable $\Lambda(t)$ such that

$$\Lambda(t) \leq \varepsilon \iff \exists x \in \mathcal{P}\colon w \cdot x \in (t, t+\varepsilon]. \tag{3.3}$$

In order to define $\Lambda(t)$, we define the *winner* x^* to be the most valuable solution satisfying $w \cdot x \leq t$, i.e.,

$$x^* = \operatorname{argmax}\{p \cdot x \mid x \in \{0,1\}^n \wedge w \cdot x \leq t\}.$$

For $t \geq 0$, such a solution x^* must always exist. We say that a solution x is a *loser* if it has a higher profit than x^* but does not satisfy the constraint $w \cdot x \leq t$. We denote by $\hat{x}$ the loser with the smallest weight (see Figure 3.3), i.e.,

$$\hat{x} = \operatorname{argmin}\{w \cdot x \mid x \in \{0,1\}^n \wedge p \cdot x > p \cdot x^*\}.$$

If there does not exist a solution x with $p \cdot x > p \cdot x^*$, then we set $\hat{x} = \perp$. Based on $\hat{x}$, we define the random variable $\Lambda(t)$ as

$$\Lambda(t) = \begin{cases} w \cdot \hat{x} - t & \text{if } \hat{x} \neq \perp, \\ \perp & \text{if } \hat{x} = \perp. \end{cases}$$

Assume that there exists a Pareto optimal solution with weight 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 $\Lambda(t) = w \cdot y - t \in (0, \varepsilon]$. Conversely, if $\Lambda(t) \leq \varepsilon$, then $\hat{x}$ must be a Pareto optimal solution whose weight lies in the interval $(t, t+\varepsilon]$. This yields Equivalence (3.3), and hence we can write the expected number of Pareto optimal solutions as

$$\mathbf{E}[q] = 1 + \lim_{k\to\infty} \sum_{i=0}^{k-1} \mathbf{Pr}\left[\Lambda\left(\frac{ni}{k}\right) \leq \frac{n}{k}\right]. \tag{3.4}$$

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 one of the auxiliary random variables. Then we analyze the auxiliary random variables and use the union bound to conclude the desired bound for $\Lambda(t)$. Let $i \in [n]$ be fixed arbitrarily. For $j \in \{0,1\}$, we define

$$\mathcal{S}^{x_i=j} = \{x \in \{0,1\}^n \mid x_i = j\},$$

and we define $x^{*,i}$ to be

$$x^{*,i} = \operatorname{argmax}\{p \cdot x \mid x \in \mathcal{S}^{x_i=0} \wedge w \cdot x \leq t\}.$$

That is, $x^{*,i}$ is the winner among the solutions that do not contain the i-th element. We restrict our attention to losers that contain the i-th element and define

$$\hat{x}^i = \operatorname{argmin}\{w \cdot x \mid x \in \mathcal{S}^{x_i=1} \wedge p \cdot x > p \cdot x^{*,i}\}.$$

If there does not exist a solution $x \in \mathcal{S}^{x_i=1}$ with $p \cdot x > p \cdot x^{*,i}$, then $\hat{x}^i$ is undefined, i.e., $\hat{x}^i = \perp$. Based on $\hat{x}^i$, we define the random variable $\Lambda^i(t)$ by

$$\Lambda^i(t) = \begin{cases} w \cdot \hat{x}^i - t & \text{if } \hat{x}^i \neq \perp, \\ \perp & \text{if } \hat{x}^i = \perp . \end{cases}$$

Summarizing, $\Lambda^i(t)$ is defined similarly to $\Lambda(t)$, but only solutions that do not contain the i-th element are eligible as winners and only solutions containing the i-th element are eligible as losers.

Lemma 3.5. *For every choice of profits and weights, either $\Lambda(t) = \perp$ or there exists an index $i \in [n]$ such that $\Lambda(t) = \Lambda^i(t)$.*

Proof. Assume that $\Lambda(t) \neq \perp$. Then there exists a winner x^* and a loser $\hat{x}$. Since $x^* \neq \hat{x}$, there must exist an index $i \in [n]$ with $x^*_i \neq \hat{x}_i$. Since all weights are positive and $w \cdot x^* < w \cdot \hat{x}$, there must even exist an index $i \in [n]$ with $x^*_i = 0$ 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 $x^* = x^{*,i}$. This follows because x^* is the solution with the highest profit among all solutions with weight at most t, and 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 the i-th element and have weight at most t. Since $x^* = x^{*,i}$, by similar arguments it follows that $\hat{x} = \hat{x}^i$. This directly implies that $\Lambda(t) = \Lambda^i(t)$. □

Lemma 3.6. *For every $i \in [n]$ and every $\varepsilon \geq 0$,*

$$\mathbf{Pr}\left[\Lambda^i(t) \in (0, \varepsilon]\right] \leq \phi\varepsilon.$$

Proof. In order to prove the lemma, it suffices to exploit the randomness of the weight w_i. Therefore, 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 $x^{*,i}$ are fixed. If the solution $x^{*,i}$ is fixed, then also the set of losers $\{x \in \mathcal{S}^{x_i=1} \mid p \cdot x > p \cdot x^{*,i}\}$ is fixed. Since the weight w_i affects all solutions from $\mathcal{S}^{x_i=1}$ in the same manner, the solution $\hat{x}^i$ does not depend on w_i. This implies that, given the fixed values of the weights w_j with $j \neq i$, we can rewrite the event $\Lambda^i(t) \in (0, \varepsilon]$ as $w \cdot \hat{x}^i - t \in (0, \varepsilon]$ for a fixed solution $\hat{x}^i$. For a constant $\kappa \in \mathbb{R}$ depending on the fixed values of the weights w_j with $j \neq i$, we can rewrite this event as $w_i \in (\kappa, \kappa + \varepsilon]$. By Observation 1.4, the probability of this event is upper bounded by $\phi\varepsilon$. □

Combining Lemmas 3.5 and 3.6 yields

$$\mathbf{Pr}\left[\Lambda(t) \leq \varepsilon\right] \leq \mathbf{Pr}\left[\exists i \in [n] \colon \Lambda^i(t) \in (0, \varepsilon]\right] \leq \sum_{i=1}^{n} \mathbf{Pr}\left[\Lambda^i(t) \in (0, \varepsilon]\right] \leq \phi n \varepsilon.$$

Combining this with (3.4) yields

$$\begin{aligned}\mathbf{E}\left[q\right] &= 1 + \lim_{k\to\infty} \sum_{i=0}^{k-1} \mathbf{Pr}\left[\Lambda\left(\frac{ni}{k}\right) \leq \frac{n}{k}\right] \\ &\leq 1 + \lim_{k\to\infty} \sum_{i=0}^{k-1} \frac{\phi n^2}{k} \\ &= 1 + \phi n^2.\end{aligned}$$

□

Finally, we obtain the following result on the running time of the Nemhauser/Ullmann algorithm.

Corollary 3.7. *For an instance of the knapsack problem with n objects whose profits are specified arbitrarily and whose weights are chosen independently according to densities $f_1, \ldots, f_n$ with $f_i \colon [0,1] \to [0,\phi]$, the expected running time of the Nemhauser/Ullmann algorithm is upper bounded by $O(\phi n^3)$.*

3.4 Extensions

The result proven in [BRV07] is much more general than the one stated in the previous section. First, it is not necessary that every profit lies in the interval $[0,1]$. In fact, any density function $f : \mathbb{R} \to [0,\phi]$ is allowed for which

$$\int_{-\infty}^{\infty} |t| \cdot f(t)\, dt$$

is bounded by some constant. Intuitively this means that the expected absolute value of a random variable drawn according to f should be constant. This way, also Gaussian perturbations and other densities that are not defined on a bounded interval are allowed. The upper bound of $O(n^2\phi)$ on the expected number of Pareto-optimal solutions remains valid, and hence also the bound of $O(n^3\phi)$ on the smoothed running time of the Nemhauser/Ullmann algorithm.

Second, the proof of Theorem 3.4 does not essentially use the fact that we consider the knapsack problem. In fact, the result by Beier et al. [BRV07] does not only apply to the knapsack problem, but to any problem with a linear objective function. In the general model, one only assumes that there is an arbitrary set of solutions $\mathcal{S} \subseteq \{0,1\}^n$, an arbitrary weight function $w : \mathcal{S} \to \mathbb{R}$ that assigns a weight to every solution, and a linear objective function $p_1x_1 + \cdots + p_nx_n$. If the coefficients $p_1, \ldots, p_n$ are drawn according to probability densities that are bounded by ϕ, the bound of $O(n^2\phi)$ on the expected number of Pareto-optimal solutions still applies, regardless of the choices for $\mathcal{S}$, w, and the densities.

For a given graph with n edges $e_1, \ldots, e_n$, one can, for example, identify every vector $x \in \{0,1\}^n$ with the 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 $\mathcal{S}$ of feasible solutions as the set of all incidence vectors of paths from s to t in the given graph. This way, the result implies that the smoothed number of Pareto-optimal paths in the bicriteria shortest-path problem is $O(n^2\phi)$, where n denotes the number of edges. 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 $\mathcal{S}$ as the set of incidence vectors of all spanning trees of a given graph. Then the result implies that there are only $O(n^2\phi)$ Pareto-optimal spanning trees in expectation in the bicriteria spanning tree problem.

These results can even be generalized to d-criteria problems [RT09, MO10], in which there are d linear objective functions with perturbed coefficients, for some constant d. For these problems the bound becomes $O(n^{2d}\phi^{d(d+1)/2})$.

Bibliography

[Aro98] Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

[AZ99] Nina Amenta and Günter M. Ziegler. *Deformed Products and Maximal Shadows of Polytopes*, volume 223 of *Contemporary Mathematics*, pages 57–90. American Mathematics Society, 1999.

[BRV07] Rene Beier, Heiko Röglin, and Berthold Vöcking. The smoothed number of Pareto optimal solutions in bicriteria integer optimization. In *12th Int. Conf. on Integer Programming and Combinatorial Optimization (IPCO)*, 2007. to appear.

[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(6):1998–2029, 1999.

[Dan63] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

[ERV07] Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP. In *18th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1295–1304, 2007.

[GS55] S. Gass and T.L. Saaty. The computational algorithm for the parametric objective function. *Naval Research Logistics Quarterly*, 2:39, 1955.

[JM97] David S. Johnson and Lyle A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*. John Wiley and Sons, 1997.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–104. Plenum Press, New York, 1972.

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984.

[Ker89] Walter Kern. A probabilistic analysis of the switching algorithm for the Euclidean TSP. *Mathematical Programming*, 44(2):213–219, 1989.

[Kha79] L. G. Khachian. A polynomial algorithm in linear programming. *Dokl. Akad. Nauk SSSR*, 244:1093–1096, 1979.

[Lue75] George S. Lueker. Unpublished manuscript, 1975. Princeton University.

[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(4):1298–1309, 1999.

[MO10] Ankur Moitra and Ryan O'Donnell. Pareto optimal solutions for smoothed analysts. arXiv:1011.2249v1 [cs.DS], 2010.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. 1995.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing*. 2005.

[NU69] George L. Nemhauser and Zev Ullmann. Discrete dynamic programming and capital allocation. *Management Science*, 15:494–505, 1969.

[Pap77] Christos H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theoretical Computer Science*, 4(3):237–244, 1977.

[PS98] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization*. Dover Publications, Inc., 1998.

[Rei91] Gerhard Reinelt. TSPLIB – A traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[RSI77] Daniel J. Rosenkrantz, Richard Edwin Stearns, and Philip M. Lewis II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977.

[RT09] Heiko Röglin and Shang-Hua Teng. Smoothed analysis of multiobjective optimization. In *50th Ann. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 681–690. IEEE, 2009.

[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(3):385–463, 2004.

[Ver06] Roman Vershynin. Beyond hirsch conjecture: walks on random polytopes and smoothed complexity of the simplex method. In *47th Ann. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 133–142, 2006.

[vLS81] Jan van Leeuwen and Anneke A. Schoon. Untangling a traveling salesman tour in the plane. In *7th Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 87–98, 1981.