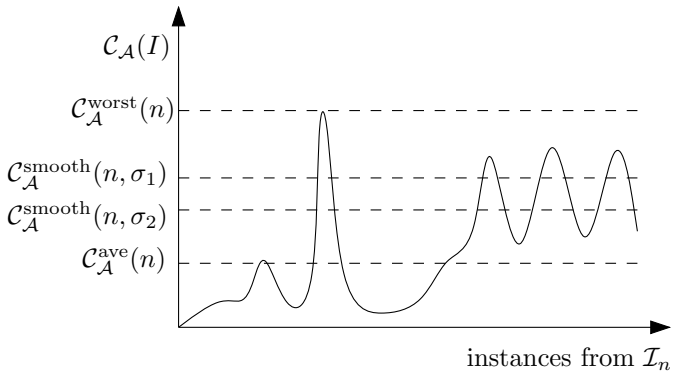
Minicourse on Smoothed Analysis

Lecture Notes

Version 1.01



Heiko Röglin
roeglin@cs.rwth-aachen.de
Department of Computer Science
RWTH Aachen

Preface

These are the lecture notes for a minicourse on "Smoothed Analysis" given as part of the "Randomized Algorithms" lecture in the summer term 2007 at RWTH Aachen. The material presented in this course is mainly based on the following research papers:

- David Arthur and Sergei Vassilvitskii. **Worst-case and smoothed analysis of the ICP algorithm, with an application to the k-means method.** In *Proc. of the 47th Ann. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 153–164, 2006.
- Rene Beier, Heiko Röglin, and Berthold Vöcking. **The smoothed number of Pareto optimal solutions in bicriteria integer optimization**. In *Proc. of the 12th Int. Conf. on Integer Programming and Combinatorial Optimization (IPCO)*, 2007. To appear.

I will be grateful for any suggestions on how to improve the lecture notes and also the lecture itself.

June 14, 2007

Changes to Version 1.0:

- Minor change in Theorem 4.1.
- Proof of Theorem 4.2: $2|A|D \to 4|A|D^2$.

Contents

1 Introduction **4**

2 Mathematical Background **7**
2.1 Notations 7
2.2 Random Variables and Random Vectors 7
2.3 Facts from Linear Algebra 9

3 The Smoothed Number of Pareto Optimal Solutions **11**
3.1 The Nemhauser/Ullmann Algorithm 11
3.2 Probabilistic Input Model 13
3.3 The Expected Number of Pareto Optimal Solutions 14

4 Smoothed Analysis of Local Search Heuristics **18**
4.1 The Iterative Closest Point Algorithm 18
4.2 Detailed Description of the ICP Algorithm 19
4.3 Smoothed Analysis of the ICP Algorithm 20
4.3.1 Case 1: Small changes in N 21
4.3.2 Case 2: Large changes in N 23
4.3.3 Putting the pieces together 25

Bibliography **27**

Chapter 1

Introduction

In the theory of algorithms, an algorithm is typically judged by its *worst-case performance*. An algorithm with good worst-case performance is the ideal case 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 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 quite 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 on the set of inputs. 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 are typically 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. Therefore, average-case analyses tend to yield too optimistic results.

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 introduce a hybrid of these two models, which they call *smoothed analysis* [ST04]. The input model in a smoothed analysis consists of two steps. In the first step, an adversary specifies an arbitrary input. After that, in the second step, this input is slightly perturbed at random. The idea behind this input model is that instances that occur in practical applications typically have a special structure, which depends on the concrete application, but that they are also subject to some random influences. The adversary can specify, for example, an arbitrary linear program with a certain structure, which is only slightly perturbed in the second step.

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

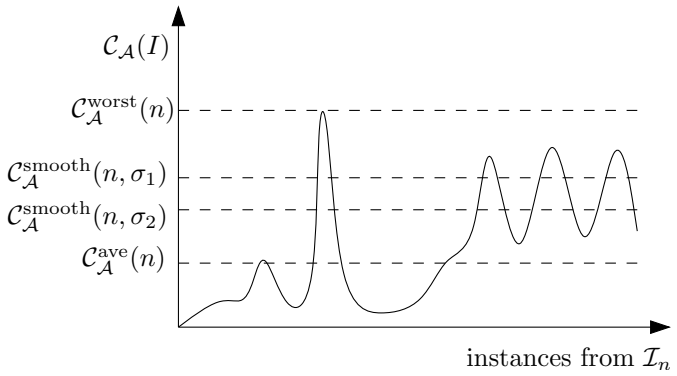


Figure 1.1: Illustration of the different complexity measures. The horizontal axis ranges over the set of inputs of length n, for some fixed n. It is assumed that $\sigma_1 < \sigma_2$. Hence, $\mathcal{C}_{\mathcal{A}}^{\text{smooth}}(n, \sigma_1) > \mathcal{C}_{\mathcal{A}}^{\text{smooth}}(n, \sigma_2)$.

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

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

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

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

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

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

These definitions are illustrated in Figure 1.1.

From the definition of smoothed complexity, one can see that it is a hybrid between 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 also well in practice, because worst-case instances might exist but they are very fragile with respect to random influences.

Spielman and Teng [ST04] show that the smoothed complexity of the simplex algorithm is polynomial for a certain pivot rule. Since the invention of smoothed analysis by Spielman

and Teng in 2001, many different results on the smoothed analysis of algorithms have been obtained, including results on different algorithms for solving linear programs, local search algorithms, various discrete optimizations problems, and the competitive ratio of online algorithms. As Spielman and Teng's analysis is very involved, we do not present it here. After recalling some facts from probability theory and linear algebra in Chapter 2, we present the smoothed analysis of a heuristic for the knapsack problem in Chapter 3 and the smoothed analysis of a local search heuristic for a pattern matching problem in Chapter 4.

Chapter 2

Mathematical Background

In this chapter, we introduce some notations and review some facts from probability theory and linear algebra. It is advisable to skim through this chapter and to read it in more detail when it is referenced in Chapters 3 and 4.

2.1 Notations

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., $\|x\| = \sqrt{x_1^2 + \cdots + x_n^2} = \sqrt{x \cdot x}$.

2.2 Random Variables and Random Vectors

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

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

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

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

This immediately yields the following observation.

Observation 2.1. *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 $\varepsilon\phi$.*

In Chapter 4, we also encounter *real-valued random vectors*, i. e., vectors whose entries are real-valued random variables. We always assume that the entries of a random vector are *independent* random variables. Then the distribution of a d-dimensional random vector X can be specified by a distribution $F_X \colon \mathbb{R}^d \to [0, 1]$ with

$$F_X(x) = \mathbf{Pr}[\forall i \in [d] \colon X_i \leq x_i] = \prod_{i \in [d]} \mathbf{Pr}[X_i \leq x_i]$$

for all $x \in \mathbb{R}^d$. The density function of a random vector X with independent entries is the function $f_X \colon \mathbb{R}^d \to \mathbb{R}_+$ with

$$f_X(x) = \prod_{i \in [d]} f_{X_i}(x_i) \tag{2.1}$$

for all $x \in \mathbb{R}^d$, where f_{X_i} denotes the density of the random variable describing the i-th entry of X. For every measurable set $C \subseteq \mathbb{R}^d$, the probability of X falling into C can be written as

$$\mathbf{Pr}\left[X \in C\right] = \int_C f_X(x)\, dx \ .$$

This implies the following generalization of Observation 2.1.

Observation 2.2. *Let X be a random vector whose density is bounded from above by $\phi > 0$, and let C denote an arbitrarily fixed measurable set of volume $\varepsilon > 0$. The probability that X takes a value in the interval I is bounded from above by $\varepsilon\phi$.*

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

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) \ . \tag{2.2}$$

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. Observation 2.2 yields the following result for Gaussian vectors.

Lemma 2.3. *Suppose $x \in \mathbb{R}^d$ is a Gaussian vector with arbitrary center and standard deviation σ. Then x lies in a fixed ball of radius ε with probability at most $(\varepsilon/\sigma)^d$.*

Proof. Due to Equations (2.1) and (2.2), the density of x has a maximum of $(\sigma\sqrt{2\pi})^{-d}$. Furthermore, a ball with radius ε has a volume of less than $(2\varepsilon)^d$ because it is contained in a hypercube with side length 2ε. Hence, the probability that x lies in such a ball is bounded by $(2\varepsilon)^d \cdot (\sqrt{2\pi}\sigma)^{-d} < (\varepsilon/\sigma)^d$ due to Observation 2.2. □

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

Lemma 2.4. *Suppose $x \in \mathbb{R}^d$ is a Gaussian vector with center 0^d and standard deviation σ. Then, for every $t \geq 1$,*

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

Proof. First we analyze a Gaussian random variable y with mean 0 and standard deviation

σ. For $t \geq 1$, we obtain

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

Now we consider the Gaussian random vector x. If $|x_i| \leq t/\sqrt{d}$ for all its entries $i \in [d]$, then $\|x\| \leq t$. Hence, it follows

$$\mathbf{Pr}\left[\|x\| > t\right] \leq \mathbf{Pr}\left[\exists i \in [d] \colon |x_i| > \frac{t}{\sqrt{d}}\right] < d\sigma \cdot \exp\left(-\frac{t^2}{2d\sigma^2}\right) \ .$$

□

From Lemma 2.4, we can conclude the following corollary.

Corollary 2.5. *Suppose $x \in \mathbb{R}^d$ is a Gaussian vector with arbitrary center $\mu \in \mathbb{R}^d$ and standard deviation σ. Then, for every $t \geq 1$,*

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

2.3 Facts from Linear Algebra

For a finite set of vectors $V = \{v_1, \ldots, v_n\} \subseteq \mathbb{R}^d$, we denote by $\operatorname{span}(V)$ the set of vectors that can be obtained as linear combinations of elements from V, i. e.,

$$\operatorname{span}(V) = \left\{ u \in \mathbb{R}^d \;\middle|\; \exists c_1, \ldots, c_n \in \mathbb{R} \colon u = \sum_{i=1}^{n} c_i v_i \right\} \ .$$

Since $\mathbb{R}^d$ is a d-dimensional vector space, we can identify, for every set $V \subseteq \mathbb{R}^d$, a subset $V_0 \subseteq V$ with $|V_0| \leq d$ such that every vector $v \in V$ can be expressed as linear combination of the vectors from V_0. The following lemma shows that an appropriate choice of V_0 guarantees that the coefficients in the linear combinations have absolute value at most 1.

Lemma 2.6. *Let $V \subseteq \mathbb{R}^d$ denote a set of vectors with $|V| \geq d$. Then there exists a subset $V_0 \subseteq V$ with $|V_0| = d$ such that any $v \in V$ can be expressed as $v = \sum_{u \in V_0} c_{uv} u$ for scalars $c_{uv} \in [-1, 1]$.*

Proof. We may assume without loss of generality that $\operatorname{span}(V) = \mathbb{R}^d$. Otherwise, if $\operatorname{span}(V)$ is a k-dimensional subspace with $k < d$, the proof yields a set V_0 with $|V_0| = k$ that possesses the desired properties. Then we can add $d - k$ arbitrary vectors to V_0. For an arbitrary point set $X \subseteq \mathbb{R}^d$ with $|X| = d$, we denote by $\mathcal{S}(X)$ the simplex with vertices $X \cup \{0\}$. We choose V_0 to be the subset of V of size d that maximizes the volume of $S(V_0)$. This ensures that $\operatorname{span}(V_0) = \mathbb{R}^d$. Hence, any given $v \in V$ can be written as

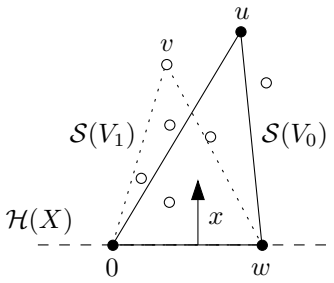


Figure 2.1: Definitions from the proof of Lemma 2.6: $V_0 = \{w, u\}$, $V_1 = \{w, v\}$, $X = \{0, w\}$.

$v = \sum_{u \in V_0} c_{uv} u$ for some scalars $c_{uv} \in \mathbb{R}$. It remains to show $|c_{uv}| \leq 1$ for all $u \in V_0$ and $v \in V$.

Fix an arbitrary $u \in V_0$ and an arbitrary $v \in V$. We denote by X the set $V_0 \setminus \{u\}$ and by V_1 the set $X \cup \{v\}$. Due to the choice of the set V_0, we know that the volume of $\mathcal{S}(V_1)$ cannot be larger than the volume of $\mathcal{S}(V_0)$. Both simplices $\mathcal{S}(V_0)$ and $\mathcal{S}(V_1)$ contain the face $\mathcal{S}(X)$, and hence they can be seen as cones with base $\mathcal{S}(X)$ and apexes u and v, respectively. Let $\mathcal{H}(X)$ denote the hyperplane that passes through the points $X \cup \{0\}$, i. e., the hyperplane that contains the face $\mathcal{S}(X)$ (see Figure 2.1 for an illustration of these definitions). The volume of the cones $\mathcal{S}(V_0)$ and $\mathcal{S}(V_1)$ can be computed as

$$\mathrm{Vol}(\mathcal{S}(V_0)) = \frac{\mathrm{Vol}(\mathcal{S}(X)) \cdot \mathrm{dist}(u, \mathcal{H}(X))}{d}$$

and

$$\mathrm{Vol}(\mathcal{S}(V_1)) = \frac{\mathrm{Vol}(\mathcal{S}(X)) \cdot \mathrm{dist}(v, \mathcal{H}(X))}{d} \ .$$

Since V_0 is chosen to maximize the volume of $\mathcal{S}(V_0)$, this implies that the distance $\mathrm{dist}(v, \mathcal{H}(X))$ cannot be larger than the distance $\mathrm{dist}(u, \mathcal{H}(X))$, which in turn implies, for any vector x that is orthogonal to $\mathcal{H}(x)$,

$$\left| x \cdot \sum_{u \in V_0} c_{uv} u \right| = |x \cdot v| \leq |x \cdot u| \ . \tag{2.3}$$

Since x is orthogonal to $\mathcal{H}(X)$, it is orthogonal to all $u' \in V_0 \setminus \{u\}$. Hence, Equation (2.3) simplifies to $|x \cdot (c_{uv} u)| \leq |x \cdot u|$, which implies the lemma. □

Chapter 3

The Smoothed Number of Pareto Optimal Solutions

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. 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 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 naïve 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] propose 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 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 $\mathcal{P}(i)' \dots$
    $\dots$ 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 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).

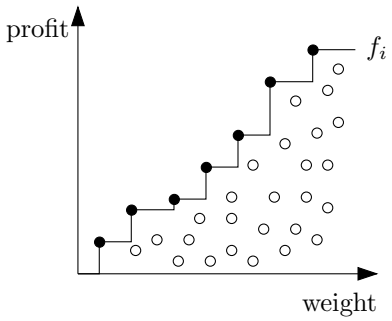


Figure 3.1: The dots correspond to solutions that contain only a subset of the first i elements. Black dots correspond to solutions from $\mathcal{P}(i)$.

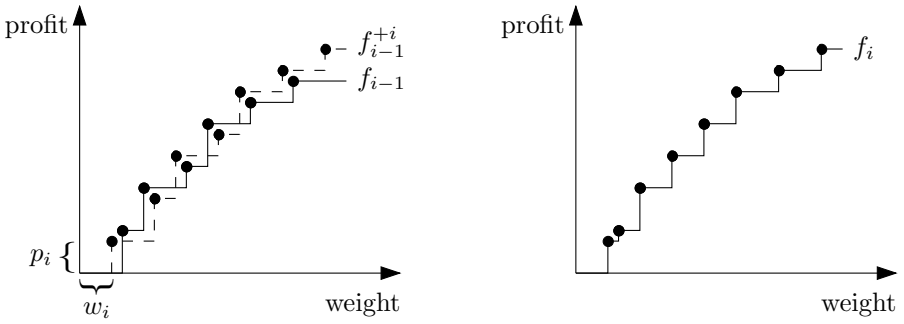


Figure 3.2: The function f_i is the upper envelope of the functions f_{i-1} and f_{i-1}^{+i}.

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*

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

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 can the random variables 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 $\phi n^2 + 1$.*

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

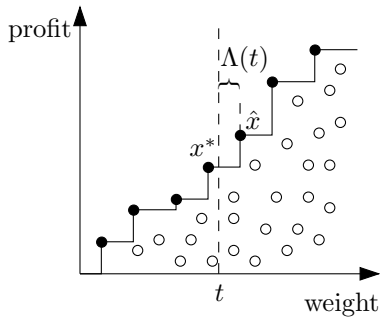


Figure 3.3: Definitions of the winner x^*, the loser $\hat{x}$, and the random variable $\Lambda(t)$.

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. 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 $(t, t+\varepsilon]$. Therefore, 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.,

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

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 (cf. Figure 3.3), i. e.,

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

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 a 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 2.1, 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 $O(\phi n^3)$.*

Chapter 4

Smoothed Analysis of Local Search Heuristics

A common and often successful approach to tackle NP-hard optimization problems is *local search*. A local search algorithm usually computes an initial feasible solution by some simple heuristic and then performs local improvements until a locally optimal solution is found. For many local search heuristics examples are known on which they perform very poorly, both with respect to running time and approximation ratio. On the other hand, many of these heuristics yield good solutions rather quickly in practice, suggesting that considering only the worst case is not the appropriate perspective for most local search algorithms.

Currently, we are aware of two papers on the smoothed complexity of local search algorithms. Both papers consider geometric problems in which the input consists of a set of points in the Euclidean space. Arthur and Vassilvitskii [AV06] analyze the smoothed complexity of local search algorithms for pattern matching and clustering, and Englert et al. [ERV07] analyze the smoothed complexity of the 2-Opt heuristic, a simple and commonly used heuristic for the Euclidean TSP. All these analyses rely essentially on the same observation, namely that in perturbed instances the smallest improvement made by any of the possible local improvements is relatively large. This in turn implies that the number of local improvement steps cannot be very large. In this chapter, we present the analysis of the pattern matching algorithm in detail.

4.1 The Iterative Closest Point Algorithm

One problem that often arises in the field of *computer vision* is *image registration*, which is the process of merging data from different images into a common "coordinate system". Image registration is used extensively in *medical imaging*, where it is common to have images from different sources that need to be combined appropriately. Another application area is the processing of satellite images, e. g., finding an arrangement of two satellite images of overlapping regions.

In many situations this problem boils down to translating (and possibly rotating or scaling) a given point set $A \subseteq \mathbb{R}^d$ until it is as close as possible to another given point set $B \subseteq \mathbb{R}^d$. To make this more precise, the goal is to find a translation $x \in \mathbb{R}^d$ and a function $N\colon A \to B$ such that the potential

$$\Phi = \sum_{a \in A} \|a + x - N(a)\|^2$$

is minimized.

This problem is NP-hard, and Besl and McKay propose a local search algorithm for finding approximate solutions, the *Iterative Closest Point (ICP) algorithm* [BM92]. This

algorithm starts with an arbitrary translation and improves it consecutively until a locally optimal solution is found. One iteration of this algorithm consists of two phases. In the first phase, every point $a \in A$ is assigned to its nearest neighbor in B given the current translation $x \in \mathbb{R}^d$. In the second phase a new translation $x' \in \mathbb{R}^d$ is computed that minimizes the potential for the assignment made in the first phase.

In practical applications it has been observed that the ICP algorithm typically reaches a locally optimal solution quickly. Contradicting these observations, Arthur and Vassilvitskii show that the worst-case running time of the ICP algorithm is fairly bad.

Theorem 4.1 ([AV06])**.** *For every $d \in \mathbb{N}$ and every $n \in \mathbb{N}$, there exist point sets $A \subseteq \mathbb{R}^d$ and $B \subseteq \mathbb{R}^d$ with $|A| = |B| = n$ for which the ICP algorithm requires $\Omega((n/d)^{d+1})$ iterations.*

In order to reconcile the theoretical results with the observations made in practice, Arthur and Vassilvitskii analyze the ICP algorithm on perturbed instances and show that its smoothed complexity is polynomial. They assume that the position of each point corresponds to a Gaussian random vector whose center can be specified by an adversary.

Theorem 4.2 ([AV06])**.** *Let the elements in A and B be independent Gaussian vectors in $\mathbb{R}^d$ with centers of norm at most 1 and standard deviation σ. Then the expected running time of the ICP algorithm is polynomially bounded in d, $|A|$, $|B|$, and σ^{-1}.*

4.2 Detailed Description of the ICP Algorithm

The main difficulty in the aforementioned image registration problem is that neither the translation x nor the correspondence N between the points in A and B is known in advance. If the translation x is fixed, the potential Φ is minimized by choosing, for all $a \in A$, $N(a)$ to be the point in B that is closest to $a + x$. Conversely, if the assignment N is fixed, a translation x that minimizes the potential Φ can be found efficiently, as the following reasoning shows.

Definition 4.3. *Let $S \subseteq \mathbb{R}^d$ be an arbitrary finite set of points. The* center of mass *of S is defined as*

$$\mathcal{C}(S) = \frac{1}{|S|} \sum_{s \in S} s \ .$$

Lemma 4.4. *Let $S \subseteq \mathbb{R}^d$ be an arbitrary finite set of points, and let $x \in \mathbb{R}^d$ be an arbitrary point. Then*

$$\sum_{s \in S} \|s - x\|^2 = \sum_{s \in S} \|s - \mathcal{C}(S)\|^2 + |S| \cdot \|\mathcal{C}(S) - x\|^2 \ .$$

Proof. Let $x' = \mathcal{C}(S)$. The lemma follows from the following calculation:

$$\begin{aligned}
\sum_{s \in S} \|s - x\|^2 &= \sum_{s \in S} (s - x) \cdot (s - x) \\
&= \sum_{s \in S} ((s - x') + (x' - x)) \cdot ((s - x') + (x' - x)) \\
&= \sum_{s \in S} (s - x') \cdot (s - x') + \sum_{s \in S} (x' - x) \cdot (x' - x) + 2 \sum_{s \in S} (s - x') \cdot (x' - x) \\
&= \sum_{s \in S} \|s - x'\|^2 + |S| \cdot \|x' - x\|^2 + 2(x' - x) \cdot \underbrace{\sum_{s \in S} (s - x')}_{=0} \ .
\end{aligned}$$

□

Lemma 4.5. *For a fixed assignment $N\colon A \to B$, the potential Φ is minimized by choosing the translation*

$$x = \frac{1}{|A|} \sum_{a \in A} (N(a) - a) \ .$$

Proof. We define the multiset $S = \{N(a) - a \mid a \in A\}$ and write the potential Φ as

$$\Phi = \sum_{a \in A} \|a + x - N(a)\|^2 = \sum_{s \in S} \|x - s\|^2 \ .$$

Due to Lemma 4.4, we have

$$\Phi = \sum_{s \in S} \|s - \mathcal{C}(S)\|^2 + |S| \cdot \|\mathcal{C}(S) - x\|^2 \ .$$

The first term in this summation is not affected by the translation x. Hence, in order to minimize the potential, it suffices to choose a translation x that minimizes the second term, which is the case for

$$x = \mathcal{C}(S) = \frac{1}{|S|} \sum_{s \in S} s = \frac{1}{|A|} \sum_{a \in A} (N(a) - a) \ .$$

□

Based on Lemma 4.5, we can formulate the ICP algorithm as follows:

Algorithm 2 The ICP algorithm

1: Choose an arbitrary starting translation $x \in \mathbb{R}^d$.
2: **repeat**
3: **for all** $a \in A$ **do** Set $N(a)$ to the point in B that is closest to $a + x$.
4: Set the translation to $x = \frac{1}{|A|} \sum_{a \in A} (N(a) - a)$.
5: **until** N and x remain unchanged in steps 3 and 4.

If ties occur in step 3 of the ICP algorithm, they can be broken arbitrarily. It is only important that they are broken in a consistent manner, i. e., once a tie is broken in favor of one point in B, it must always be broken in favor of that point. This ensures that whenever a point $a \in A$ changes its assigned point $N(a)$ in step 3, the potential Φ strictly decreases. Furthermore, due to Lemma 4.5, if the translation is changed in step 4, then the potential decreases strictly as well. This implies that no assignment $N\colon A \to B$ can appear twice during the execution of the ICP algorithm. Since there are only $|B|^{|A|}$ different assignments, the ICP algorithm must terminate after a finite number of iterations, where we use the term *iteration* to denote a consecutive execution of steps 3 and 4.

4.3 Smoothed Analysis of the ICP Algorithm

In this section, we prove Theorem 4.2, that is, we show that the expected number of iterations of the ICP algorithm is polynomially bounded if all points are chosen according to Gaussian random vectors.

The important part of the proof is to show that every iteration of the ICP algorithm decreases the potential by a polynomially large amount $1/\text{poly}(|A|, |B|, d, \sigma^{-1})$ with

high probability. Combining this with the observation that the potential is polynomially bounded after the first iteration with high probability yields the desired polynomial bound on the expected number of iterations. In order to show that every iteration of the ICP algorithm decreases the potential by a polynomially large amount, we distinguish between two cases: either at most $2d$ points $a \in A$ change their assigned point $N(a)$ in B, or at least $2d$ points change their assignment.

4.3.1 Case 1: Small changes in N

Let $k \in \mathbb{N}$ be a constant to be defined later. In this section, we analyze the case that at most k points change their assignment in step 3. In the following we say that a *multiset* B' is a *subset* of B (denoted as $B' \subseteq B$) if it contains only points from B. Each of these points can, however, be contained more than once in B'. We say that B' is a *size-k subset* if it contains *at most* k points, counting multiplicities. For two multisets $B_1, B_2 \subseteq B$, we denote by $B_1 \cap B_2$ the multiset in which the multiplicity of each element is the minimum of its multiplicities in B_1 and B_2. We denote by $B_1 + B_2$ the multiset in which the multiplicity of each element is the sum of the multiplicities in B_1 and B_2. By $B_1 - B_2$ we denote the multiset in which the multiplicity of each element is the difference of the multiplicities in B_1 and B_2 if this difference is positive and 0 otherwise.

Definition 4.6. *A point set $B \subseteq \mathbb{R}^d$ is* (k, δ)-sparse *if no pair of distinct size-k multisets $B_1, B_2 \subseteq B$ satisfies*

$$\left\| \sum_{b \in B_1} b - \sum_{b \in B_2} b \right\| \leq \delta \ .$$

In the following we show that if B is (k, δ)-sparse, then every iteration in which N changes for at most k points yields a significant change of the translation, which in turns implies a significant potential drop. Then it only remains to show that B is likely to be (k, δ)-sparse for an appropriately chosen δ.

Lemma 4.7. *Consider two consecutive iterations of the ICP algorithm, and let x_1 and x_2 denote the translations after the first and second iteration, respectively. Then the potential decreases during the second iteration by at least $|A| \cdot \|x_1 - x_2\|^2$.*

Proof. As in the proof of Lemma 4.5, we write the potential after the execution of step 3 of the second iteration as

$$\Phi = \sum_{s \in S} \|s - \mathcal{C}(S)\|^2 + |A| \cdot \|\mathcal{C}(S) - x_1\|^2$$

for $S = \{N(a) - a \mid a \in A\}$, where $N \colon A \to B$ denotes the assignment made during the second iteration. In step 4 of the second iteration, the translation x_2 is set to $\mathcal{C}(S)$ which yields a potential of

$$\Phi' = \sum_{s \in S} \|s - \mathcal{C}(S)\|^2 \ .$$

Hence, the potential drops by at least

$$\Phi - \Phi' = |A| \cdot \|\mathcal{C}(S) - x_1\|^2 = |A| \cdot \|x_2 - x_1\|^2 \ .$$

□

Lemma 4.8. *Let $A \subseteq \mathbb{R}^d$ and $B \subseteq \mathbb{R}^d$ be point sets, and suppose that B is (k, δ)-sparse. Every iteration of the ICP algorithm (except for the first one) in which the assignment $N\colon A \to B$ changes for at most k points results in a potential drop of at least $\delta^2/|A|$ or in the termination of the algorithm.*

Proof. Let N_1 denote the multiset $N(A) = \{N(a) \mid a \in A\}$ before the iteration of the ICP algorithm, and let N_2 denote the multiset $N(A)$ after the iteration. We are only interested in the points $b \in B$ that change their multiplicity during the iteration. Therefore, we define $N_0 = N_1 \cap N_2$ to be the multiset of the common elements. Furthermore, we set $B_1 = N_1 - N_2$ and $B_2 = N_2 - N_1$, which yields $N_1 = N_0 + B_1$ and $N_2 = N_0 + B_2$. If N changes for at most k points in A, then B_1 and B_2 are each of size at most k. If B_1 and B_2 are both empty, then the translation does not change during the iteration, and hence the ICP algorithm terminates after the next iteration.

Otherwise, B_1 and B_2 must be distinct size-k multisets of B. Since B is (k, δ)-sparse, it follows

$$\left\| \sum_{b \in B_1} b - \sum_{b \in B_2} b \right\| > \delta \ .$$

Let x_1 and x_2 denote the translations before and after the iteration, respectively. For $i \in \{1, 2\}$, the translation x_i can be written as

$$x_i = \frac{1}{|A|} \left(\sum_{b \in N_0} b + \sum_{b \in B_i} b - \sum_{a \in A} a \right) \ .$$

Hence,

$$\|x_1 - x_2\| = \frac{1}{|A|} \left\| \sum_{b \in B_1} b - \sum_{b \in B_2} b \right\| > \frac{\delta}{|A|} \ .$$

Due to Lemma 4.7, this implies that the potential decreases by at least $\delta^2/|A|$ during the iteration. □

Now it only remains to show that the point set B is likely to be (k, δ)-sparse.

Lemma 4.9. *Let the elements in B be independent Gaussian vectors in $\mathbb{R}^d$ with standard deviation σ. Then B is not (k, δ)-sparse with probability at most $|B|^{2k}(\delta/\sigma)^d$.*

Proof. We fix two distinct size-k multisets $B_1, B_2 \subseteq B$ and bound the probability that $\|y\| \leq \delta$ for $y = \sum_{b \in B_1} b - \sum_{b \in B_2} b$. Then a union bound over all possible choices of B_1 and B_2 yields the lemma.

We have $y = \sum_{b \in B} c_b b$ for some integer coefficients c_b. Since B_1 and B_2 are distinct, there exists some b for which $c_b \neq 0$. We assume that all points $b' \in B \setminus \{b\}$ are already fixed and use only the randomness of the point b. Under this assumption, we can write $\|y\| \leq \delta$ as $\|c_b b - y'\| \leq \delta$ for a fixed y' depending on the $b' \in B \setminus \{b\}$. Hence, the event $\|y\| \leq \delta$ occurs if and only if b falls into a ball of radius $\delta/|c_b|$ with center y'/c_b. Due to Lemma 2.3, we can upper bound the probability of this event by $(\delta/\sigma)^d$.

Since there are at most $|B|^{2k}$ choices for the sets B_1 and B_2, the lemma follows from a union bound. □

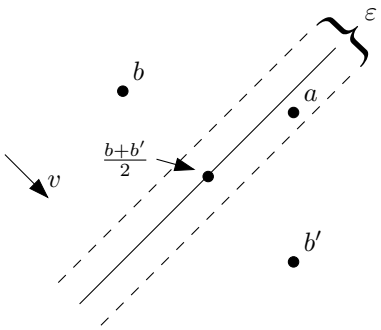


Figure 4.1: Illustration of Definition 4.10. The point a is ε-centered between b and b', and v is the unit vector in the direction $b' - b$.

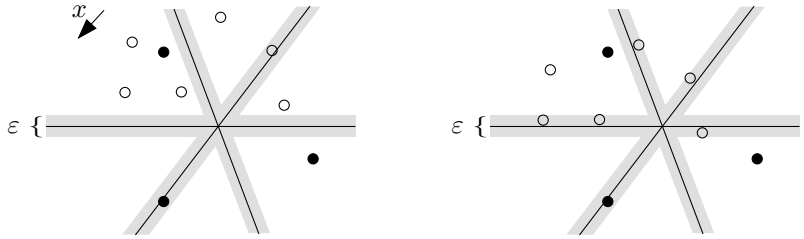


Figure 4.2: White dots are points in A, and black dots are points in B. (A, B) is $(5, \varepsilon)$-centerable but not $(6, \varepsilon)$-centerable.

4.3.2 Case 2: Large changes in N

Now we consider iterations of the ICP algorithm in which the assignment $N\colon A \to B$ changes for more than k points. The idea is that in these iterations at least one point decreases the potential significantly when its assignment is changed.

Definition 4.10. *Given points $a, b, b' \in \mathbb{R}^d$, we say that a is* ε-centered *between b and b' if a is within $\varepsilon/2$ of the hyperplane bisecting b and b'.*

Let v denote the unit vector in the direction $b' - b$, i. e., $v = (b' - b)/\|b' - b\|$, and let $\overline{b}$ denote the point $(b+b')/2$ (see Figure 4.1). Recall from linear algebra that $a \cdot v - \overline{b} \cdot v$ is the distance of the point a from the hyperplane with normal vector v that passes through the point $\overline{b}$. This is the hyperplane bisecting b and b', and hence we can phrase Definition 4.10 as follows: A point a is ε-centered between b and b' if and only if

$$a \cdot v \in \left(\frac{b + b'}{2}\right) \cdot v \pm \frac{\varepsilon}{2} \ . \tag{4.1}$$

Definition 4.11. *Let $A \subseteq \mathbb{R}^d$ and $B \subseteq \mathbb{R}^d$ be point sets. We say that (A, B) is* (k, ε)-centerable *if there exist distinct elements $a_1, \ldots, a_k \in A$, (not necessarily distinct) elements $b_1, \ldots, b_k \in B$ and $b'_1, \ldots, b'_k \in B$, and a translation $x \in \mathbb{R}^d$ such that $a_i + x$ is ε-centered between b_i and b'_i for all $i \in [k]$ (see Figure 4.2).*

If a point $a \in A$ changes its assignment $N(a)$ during one iteration of the ICP algorithm from a point $b \in B$ to a point $b' \in B$, and if $a + x$ is not ε-centered for the translation $x \in \mathbb{R}^d$ before or after the iteration, then the translation must have changed during the iteration by at least $\varepsilon/2$, yielding a potential drop of at least $|A| \cdot \varepsilon^2/4$ due to Lemma 4.7.

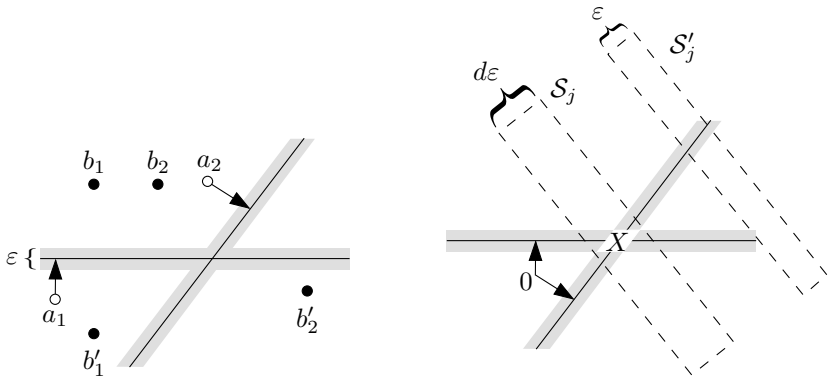


Figure 4.3: Definitions in the proof of Lemma 4.13.

Lemma 4.12. *Let $A \subseteq \mathbb{R}^d$ and $B \subseteq \mathbb{R}^d$ be point sets such that (A, B) is not (k, ε)-centerable. Every iteration of the ICP algorithm (except for the first one) in which the assignment $N\colon A \to B$ changes for at least k points results in a potential drop of at least $|A| \cdot \varepsilon^2/4$.*

Proof. Consider one iteration of the ICP algorithm in which at least k points change their assignment, and let $a_1, \dots, a_k \in A$ denote k distinct points from A whose assignment is changed. Let $b_1, \dots, b_k$ and $b_1', \dots, b_k'$ be the original and new points from B to which the a_i's are assigned. Since (A, B) is not (k, ε)-centerable, there exists an $i \in [k]$ such that $a_i + x$ is not ε-centered between b_i and b_i', where x denotes the translation after the iteration. That is, $a_i + x$ has a distance of more than $\varepsilon/2$ from the hyperplane $\mathcal{H}$ bisecting b_i and b_i'. Let x' denote the translation before the iteration. We know that $a_i + x'$ is on the same side of $\mathcal{H}$ as b_i and that $a_i + x$ is on the same side of $\mathcal{H}$ as b_i'. Combining these observations implies $\|x - x'\| > \varepsilon/2$. Hence, due to Lemma 4.7, the potential decreases during the iteration by at least $|A| \cdot \varepsilon^2/4$. □

Lemma 4.13. *Let the elements in A and B be independent Gaussian vectors in $\mathbb{R}^d$ with standard deviation σ, and assume $k \geq d$. The probability that (A, B) is (k, ε)-centerable is bounded from above by*

$$(|A||B|^2)^k \left(\frac{(d+1)\varepsilon}{\sigma}\right)^{k-d}.$$

Proof. We select (not necessarily distinct) elements $b_1, \dots, b_k \in B$, $b_1', \dots, b_k' \in B$, and distinct elements $a_1, \dots, a_k \in A$, estimate the probability that they satisfy the conditions given in Definition 4.11, and apply a union bound over all possible choices of these elements. The b_i's and b_i''s can be placed arbitrarily. We use only the randomness of the a_i's.

For $i \in [k]$, let v_i denote the unit vector in the direction $b_i' - b_i$. Without loss of generality, we assume that $V_0 = \{v_1, \dots, v_d\}$ satisfies the condition given in Lemma 2.6. We denote by $X \subseteq \mathbb{R}^d$ the set of translations $x \in \mathbb{R}^d$ such that $a_i + x$ is ε-centered between b_i and b_i' for all $i \in [d]$. Given $x_1, x_2 \in X$, it follows from Equation (4.1) that $|(x_2 - x_1) \cdot v_i| \leq \varepsilon$ for all $i \in [d]$.

Let $j \in \{d+1, \dots, k\}$ be chosen arbitrarily. From Lemma 2.6 it follows that v_j can be expressed as $v_j = \sum_{i=1}^{d} c_{ij} v_i$ with $|c_{ij}| \leq 1$, so it follows $|(x_2 - x_1) \cdot v_j| \leq d\varepsilon$. Therefore, X is contained in a slab $\mathcal{S}_j$ with height $d\varepsilon$ in the direction of v_j. The position of this slab is independent of the random vectors $y_{d+1}, \dots, y_k$.

On the other hand, $a_j + x$ is ε-centered between b_j and b_j' only if x is contained in a slab $\mathcal{S}_j'$ centered at $(b_j + b_j')/2 - a_j$ with height ε in the direction of v_j. Only if the slabs $\mathcal{S}_j$

and $\mathcal{S}'_j$ intersect for every $j \in \{d+1, \ldots, k\}$, a common translation $x \in \mathbb{R}^d$ can be found such that $a_i + x$ is ε-centered between b_i and b'_i for all $i \in [k]$ (see Figure 4.3). The slabs $\mathcal{S}_j$ and $\mathcal{S}'_j$ can only intersect if $a_j \cdot v_j$ lies in a certain fixed interval of length $(d+1)\varepsilon$. This happens with probability at most $(d+1)\varepsilon\sigma^{-1}$. Since the events that $\mathcal{S}_j$ and $\mathcal{S}'_j$ intersect are independent for $j = d+1, \ldots, k$, the probability that $\mathcal{S}_j$ and $\mathcal{S}'_j$ intersect for every $j \in \{d+1, \ldots, k\}$ can be bounded from above by $((d+1)\varepsilon/\sigma)^{k-d}$. Since there are at most $(|A||B|^2)^k$ possible choices for the a_i's, b_i's and b'_i's, this implies the lemma. □

4.3.3 Putting the pieces together

Now we are ready to prove Theorem 4.2. For the sake of simplicity, we do not attempt to minimize the degree of the polynomial.

Proof of Theorem 4.2. Ezra, Sharir, and Efrat [ESE06] prove that, in the worst case, the number of iterations of the ICP algorithm is bounded from above by $\kappa(|A| \cdot |B| \cdot d)^d$ for a sufficiently large constant κ. Let $p = \kappa^{-1}(|A| \cdot |B| \cdot d)^{-d}$, $k = 2d$, $\delta = \sigma|B|^{-4} \cdot p^{1/d}$, and $\varepsilon = \sigma((d+1) \cdot |A|^2 \cdot |B|^4)^{-1} \cdot p^{1/d}$.

In the following, we define three failure events. If one of these events occurs, our analysis fails and we can estimate the number of iterations only by its worst-case bound $\kappa(|A| \cdot |B| \cdot d)^d$.

- The first failure event is the event that B is not (k, δ)-sparse. Due to Lemma 4.9, this happens with probability at most

$$|B|^{2k} \left(\frac{\delta}{\sigma}\right)^d = |B|^{4d} \left(\frac{1}{|B|^4} \cdot p^{1/d}\right)^d = p \ .$$

- The second failure is the event that (A, B) is (k, ε)-centerable. Due to Lemma 4.13, this happens with probability at most

$$(|A||B|^2)^k \left(\frac{(d+1)\varepsilon}{\sigma}\right)^{k-d} = (|A||B|^2)^{2d} \left(\frac{1}{|A|^2|B|^4} \cdot p^{1/d}\right)^d = p \ .$$

- The third failure event is the event that there exists a point $a \in A$ with $\|a\| > D$, where $D = 1 + \sqrt{2d\sigma^2 \ln(|A|d\sigma/p)}$. This event can only occur if one Gaussian random vector from A deviates from its center by more than $D - 1$. Due to Corollary 2.5, the probability of this event is bounded from above by

$$|A| \cdot d\sigma \cdot \exp\left(-\frac{(D-1)^2}{2d\sigma^2}\right) = p \ .$$

If none of the three failure events occurs, the potential ϕ is bounded from above by $4|A|D^2$ after the first iteration, it decreases by at least

$$\frac{\delta^2}{|A|} = \frac{\sigma^2}{|A| \cdot |B|^8} \cdot p^{2/d}$$

in each iteration in which at most $2d$ points from A change their assignment due to Lemma 4.8, and it decreases by at least

$$\frac{|A| \cdot \varepsilon^2}{4} = \frac{\sigma^2}{4(d+1)^2|A|^3 \cdot |B|^8} \cdot p^{2/d}$$

in each iteration in which at least $2d$ points from A change their assignment due to Lemma 4.12. This implies that the number of iterations of the ICP algorithm is bounded by a polynomial $q(|A|, |B|, d, \sigma^{-1})$ if none of the failure events occurs.

Let T denote the number of iterations of the ICP algorithm and let $\mathcal{F}$ denote the union of the three failure events. Combining all arguments yields the following upper bound on the expected value of T:

$$\begin{aligned}\mathbf{E}[T] &= \mathbf{Pr}[\neg\mathcal{F}] \cdot \mathbf{E}[T \mid \neg\mathcal{F}] + \mathbf{Pr}[\mathcal{F}] \cdot \mathbf{E}[T \mid \mathcal{F}] \\ &\leq \mathbf{E}[T \mid \neg\mathcal{F}] + 3p \cdot \kappa(|A| \cdot |B| \cdot d)^d \\ &\leq q(|A|, |B|, d, \sigma^{-1}) + 3 \ .\end{aligned}$$

□

Bibliography

[AV06] David Arthur and Sergei Vassilvitskii. Worst-case and smoothed analysis of the ICP algorithm, with an application to the k-means method. In *Proc. of the 47th Ann. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 153–164, 2006.

[BM92] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

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

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

[ESE06] Esther Ezra, Micha Sharir, and Alon Efrat. On the ICP algorithm. In *Proc. of the 22nd ACM Symp. on Computational Geometry (SoCG)*, pages 95–104, 2006.

[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.

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

[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.