

# Smoothed Analysis: Analysis of Algorithms Beyond Worst Case

Bodo Manthey\*      Heiko Röglin†

Many algorithms perform very well in practice, but have a poor worst-case performance. The reason for this discrepancy is that worst-case analysis is often a way too pessimistic measure for the performance of an algorithm. In order to provide a more realistic performance measure that can explain the practical performance of algorithms, smoothed analysis has been introduced. It is a hybrid of the classical worst-case analysis and average-case analysis, where the performance on inputs is measured that are subject to random noise.

We give a gentle, not too formal introduction to smoothed analysis by means of two examples: the  $k$ -means method for clustering and the Nemhauser/Ullmann algorithm for the knapsack problem.

## 1 Analyzing Performance

The analysis of algorithms aims at providing measures for the performance of algorithms. It helps to compare algorithms and to understand their behavior, with the ultimate goal of facilitating the design of better algorithms.

### 1.1 Traditional Measures

**Worst-Case Analysis.** The most common approach is worst-case analysis, where the performance of an algorithm is measured by means of the most difficult instances. A good worst-case performance is the strongest possible guarantee for its performance: It means that the algorithm performs well on every possible instance, no matter where this instance comes from.

Unfortunately, there are quite a few algorithms for which one can prove that their worst-case performance is bad, but which nonetheless exhibit a remarkably good performance in practice. A famous example is the simplex method for solving linear optimization problems: Although its running time is exponential in the worst case, it often outperforms other algorithms that have a polynomial worst-case running time.

The reason for this discrepancy between worst-case and practical performance is that worst-case analysis may be dominated by pathological instances that do not resemble real-world instances. In this sense, worst-case analysis may be far too pessimistic in quantifying the performance of an algorithm.

---

\*University of Twente, Department of Applied Mathematics, [b.manthey@utwente.nl](mailto:b.manthey@utwente.nl)

†University of Bonn, Institute of Computer Science, [heiko@roeglin.org](mailto:heiko@roeglin.org)

**Average-Case Analysis.** To avoid the drawback of worst-case analysis, one frequently uses average-case analysis as an alternative measure. Here, the expected performance on a random input is used as performance measure. This leaves open the question of which probability distribution on the set of instances should be used. It would be ideal if we knew the distribution of instances that show up in practice. However, first, this distribution varies between different areas where the algorithm is applied to. Second, this distribution is usually not even known for a specific application. And, third, even if it were known, we cannot expect this distribution to have a clean mathematical representation. But a clean mathematical representation is what makes an average-case analysis doable in the first place.

Instead of using the (ideal but unknown) distribution of instances that show up in the application at hand, one uses probability distributions with concise descriptions such as the uniform distribution or independent Gaussian random variables. But such probability distributions share a crucial drawback: Instances from practice might have little in common with such random instances. The reason is that such random instances have very specific properties with overwhelming probability. In fact, this is exploited by many average-case analyses: They rely on a proof that the algorithm performs well if the instance has the specific properties that random instances have. Thus, statements about the average-case performance of an algorithm often do not say much about the performance on real-world instances.

## 1.2 Smoothed Analysis

To overcome the drawbacks of both average-case and worst-case analysis and in order to explain the speed of the simplex method, Daniel A. Spielman and Shang-Hua Teng introduced the notion of smoothed analysis [28]. Smoothed analysis is a hybrid of average-case and worst-case analysis: First, an adversary chooses an instance. Second, this instance is slightly randomly perturbed. The smoothed performance is the expected performance, where the expectation is taken over the random perturbation. The adversary, trying to make the algorithm look as bad as possible, chooses an instance that maximizes this expected performance.

The perturbation models random influences on the instances. These influences can, for instance, stem from measurements errors, numerical imprecision, or rounding errors (maybe from a preprocessing step). They also come into play naturally if the actual instance is a random sample from a larger set. This is the case, for instance, if a survey is taken by a random subset of a population. We might also think of the perturbation as modeling arbitrary influences on the input, which we cannot quantify exactly, but also have no reason to believe to be adversarial.

Let us now give a semi-formal definition of smoothed analysis. To do this, we recall how worst-case and average-case analysis are defined. Usually, we measure the performance of an algorithm as a function  $T$  of the size  $n$  of the input. Let  $\mathcal{S}_n$  denote the set of all instances of size  $n$  for a specific problem, and let  $t(X)$  be the running time of the algorithm on instance  $X$ . Then the worst-case running time for input size  $n$  is the maximum running time over all instances of this size:

$$T_{\text{worst}}(n) = \max_{X \in \mathcal{S}_n} t(X).$$

For average-case analysis, we need a probability distribution according to which the instances are drawn. Let  $\mathcal{P}_n$  be a probability distribution on the set  $\mathcal{S}_n$  of instances of size  $n$ . Then the average-case running time is the expected running time with respect to  $\mathcal{P}_n$ :

$$T_{\text{average}}(n) = \mathbb{E}_{X \sim \mathcal{P}_n} (t(X)).$$

In smoothed analysis, as mentioned above, an adversary chooses an instance  $X$ , and then this instance is subjected to random noise  $Y$ . The parameter  $\sigma$ , which is called the perturbation parameter, measures the strength of the noise. We can think of  $\sigma$  being the standard deviation of the perturbation. Overall, we get the following definition for the smoothed running time:

$$T_{\text{smoothed}}(n, \sigma) = \max_{X \in \mathcal{S}_n} \mathbb{E}_Y(t(X + \sigma Y)).$$

(For conciseness, we have assumed here that instances have the format of a numerical vector. Otherwise, the “+” in  $X + \sigma Y$  has to be interpreted in a problem-specific way.) If the perturbation parameter  $\sigma$  is extremely small, then  $X + \sigma Y$  is approximately  $X$ . Hence, smoothed analysis becomes worst-case analysis. If  $\sigma$  is extremely large, then the perturbation swamps out the original instance  $X$ . Then the instance  $X + \sigma Y$  is practically random instance, and we obtain an average-case analysis. By varying the strength  $\sigma$  of the perturbation, smoothed analysis interpolates between these two extreme cases. In order to get a realistic performance analysis, we are particularly interested in the case that  $\sigma$  is small. If the smoothed running time is small, then bad worst-case instances might exist, but one must be extremely unlucky to see such an instance in the presence of a small amount of random noise.

Smoothed analysis has been invented by Spielman and Teng in order to explain the performance of the simplex method for linear programming [28]. A linear program is an optimization problem of the form

$$\begin{aligned} & \text{minimize} && c \cdot x = \sum_{i=1}^d c_i x_i \\ & \text{subject to} && A \cdot x \leq b. \end{aligned}$$

Here,  $A$  is an  $n \times d$  matrix,  $b$  is an  $n$ -dimensional vector and  $c$  is a  $d$ -dimensional vector. The goal is to find a  $d$ -dimensional vector  $x$  that fulfills  $A \cdot x \leq b$  component-wise and minimizes the objective function  $c \cdot x$ . The simplex method is a popular method for solving linear programs, although its worst-case running time is exponential and there exist algorithms for solving linear programs with polynomial worst-case running time. The reason why nevertheless the simplex method is used is that it often outperforms the algorithms with polynomial worst-case running time in practice.

Spielman and Teng proved the following: For any matrix  $A$  and any vectors  $b$  and  $c$ , the expected running time of the simplex method on the linear program

$$\begin{aligned} & \text{minimize} && c \cdot x \\ & \text{subject to} && (A + G) \cdot x \leq (b + h) \end{aligned}$$

is bounded by a polynomial in  $n$ ,  $d$ , and  $1/\sigma$ . We denote this by  $\text{poly}(n, d, 1/\sigma)$ . Here,  $G$  and  $h$  are a matrix and a vector, respectively, consisting of independent Gaussian random variables with mean 0 and standard deviation  $\sigma \cdot \max_i(\|A_i, b_i\|)$ , where  $A_i$  denotes the  $i$ -th row of the matrix  $A$ . Thus, even if we perturb the instance only very slightly, i.e., if we choose  $\sigma = 1/\text{poly}(n, d)$ , the simplex method has polynomial expected running time for every  $A$ ,  $b$ , and  $c$  an adversary can come up with.

We stress that it is natural that the smoothed running time increases when  $\sigma$  becomes smaller: The smaller  $\sigma$ , the weaker the perturbation and the more powerful the adversary. Thus, for smaller  $\sigma$ , smoothed analysis gets closer to worst-case analysis.

Since its invention in 2001, smoothed analysis has been applied successfully to a variety of different algorithms and problems, ranging from continuous to discrete optimization. In addition to what we discuss in the subsequent sections, this includes linear programming [10,

25,27], online and approximation algorithms [4,9,26], searching and sorting [3,13,15,18], game theory [11,12], and local search [2,14].

Spielman and Teng have written a comprehensive survey of smoothed analysis in 2009 [29], where they discuss some of the aforementioned results. We strongly recommend their survey to the reader who wants to get an overview of the algorithms and problems to which smoothed analysis has already been applied successfully. In the present article, we present two results in more detail and provide some high-level ideas on how smoothed analysis can be applied.

First, we will consider the  $k$ -means method, which is a fundamental clustering algorithm (Section 2). Although the  $k$ -means method has an exponential worst-case running time, one of the main reasons for its popularity is its speed in practical applications. After that, we will consider the knapsack problem (Section 3). This problem, although one of the basic NP-hard optimization problems, can be solved very efficiently in practice. We conclude this survey with a brief discussion of smoothed analysis and its impact on algorithm design (Section 4).

## 2 Smoothed Analysis of $k$ -Means

### 2.1 The $k$ -Means Method

Clustering is ubiquitous in computer science, with applications ranging from computational biology over machine learning to image analysis. The  $k$ -means method is a very simple and implementation-friendly heuristic for clustering. It is used to partition a set  $\mathcal{X}$  of  $n$   $d$ -dimensional data points into  $k$  clusters. (The number  $k$  of clusters is fixed in advance.) In  $k$ -means clustering, our goal is not only to get a clustering of the data points, but also to get a *center*  $c_i$  for each cluster  $X_i$  of the clustering  $X_1, \dots, X_k$ . This center can be viewed as a representative of its cluster. We do not require the centers to be among the data points, but they can be arbitrary points.

The goal is to find a “good” clustering, where “good” means that the clustering should minimize the sum of the squared distances of data points to their respective centers. Thus, our goal is to minimize the objective function

$$\sum_{i=1}^k \sum_{x \in X_i} \|x - c_i\|^2.$$

Here,  $\|x - c_i\|$  denotes the Euclidean distance between  $x$  and  $c_i$ .

Given the cluster centers  $c_1, \dots, c_k \in \mathbb{R}^d$ , each point  $x \in \mathcal{X}$  should be assigned to the cluster  $X_i$  that is represented by its closest center  $c_i$ . On the other hand, given a clustering  $X_1, \dots, X_k$  of the data points, each center  $c_i$  should be chosen as  $\frac{1}{|X_i|} \cdot \sum_{x \in X_i} x$  in order to minimize the objective function (this is specific to using squared distances). This means that a center  $c_i$  should be chosen as the center of mass of its cluster  $X_i$ .

The *k-means method* (often called *k-means* for short or *Lloyd’s method* because it is based on ideas by Lloyd [17]) exploits this duality between clustering and centers: Given the centers, it is clear how the clustering should be chosen. And given the clustering, we know where the centers should be. We start with some clustering and then alternately optimize centers and clustering until this process stabilizes and we have reached a local optimum. Algorithm 1 states this more formally. Figure 1 provides an example run of the  $k$ -means method in two dimensions. Given the  $k$  centers, the Voronoi cell of a center  $c_i$  consists of all points closer to

$c_i$  than to any other center. The gray lines in Figure 1 are the borders between the Voronoi cells.

*Input:* set  $\mathcal{X}$  of  $n$  data points in  $\mathbb{R}^d$ .

*Output:* clustering  $X_1, \dots, X_k$ , centers  $c_1, \dots, c_k$ .

- 1: Choose initial cluster centers  $c_1, \dots, c_k$ .
- 2: For each point  $x \in \mathcal{X}$ : If  $c_i$  is the center closest to  $x$ , then assign  $x$  to  $X_i$ .
- 3: For each  $i \in \{1, \dots, k\}$ : Set  $c_i = \frac{1}{|X_i|} \cdot \sum_{x \in X_i} x$ .
- 4: If anything has changed during the last iteration, then go back to Step 2.

Algorithm 1: The  $k$ -means method.

The  $k$ -means method is one of the most popular clustering algorithms used in scientific and industrial applications. The main reason for its popularity is its speed. This, however, is in stark contrast to its performance in theory: The worst-case running time of  $k$ -means has recently been shown to be exponential in the number  $k$  of clusters [30]. The only known upper bound for its running time is  $\text{poly}(n^{kd})$  [16], but this bound is far from explaining the speed of  $k$ -means.

In order to explain the practical performance of  $k$ -means and to reconcile theory and practice, a smoothed analysis of  $k$ -means has been conducted. Here, an adversary specifies the data points, after which they are perturbed by independent Gaussian random variables with standard deviation  $\sigma$ . In a series of papers [1, 2, 19], it has been shown that the smoothed running time of the  $k$ -means method is bounded from above by a polynomial in the number  $n$  of data points and  $1/\sigma$ , i.e., it is bounded by  $\text{poly}(n, 1/\sigma)$ . The smoothed analysis has also been generalized to a wider class of distance measures called Bregman divergences [20]. For these distance measures, however, only a weaker bound is known as yet, and it is still open if the smoothed running time of  $k$ -means is bounded by  $\text{poly}(n, 1/\sigma)$  for Bregman divergences.

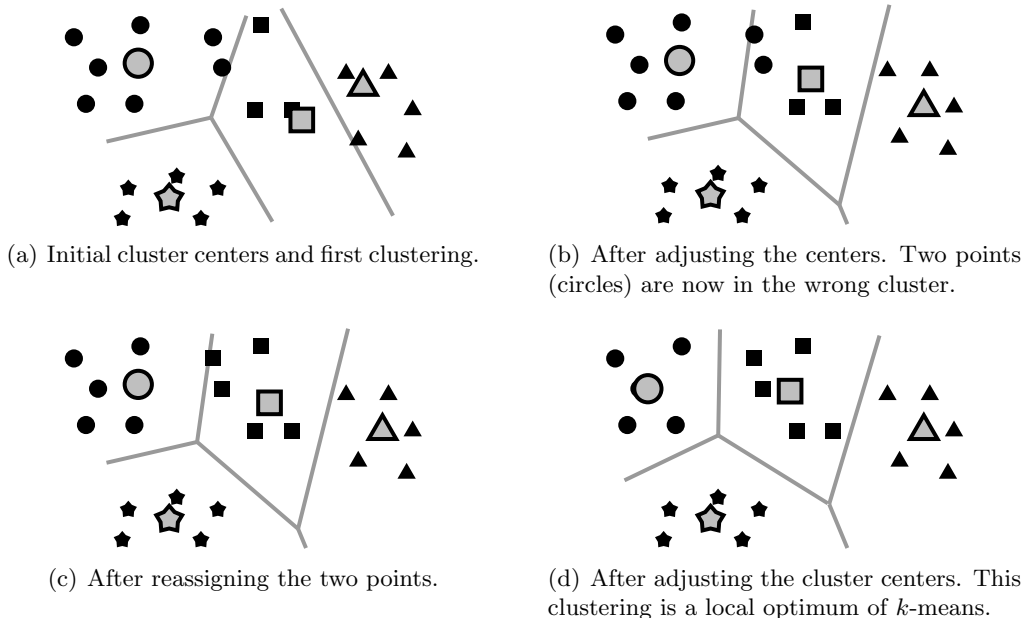


Figure 1: An example of  $k$ -means.

## 2.2 A Sketch of the Smoothed Analysis

In order to give the reader an idea of a smoothed analysis, we will give a (very) high-level sketch of the smoothed analysis of  $k$ -means. The idea is as follows: We use a potential function argument using the objective function, i.e., the sum of the squared distances of data points to their centers, as the potential. If we can show that the potential (1) is initially at most  $P$  and (2) decreases in every iteration by at least  $\Delta$ , then  $k$ -means can run for at most  $P/\Delta$  iterations. This is because the objective function cannot become negative. Thus, we have to bound  $P$  from above by  $\text{poly}(n, 1/\sigma)$  and  $\Delta$  from below by  $\text{poly}(1/n, \sigma)$ . Bounding  $P$  is neither difficult nor instructive. Thus, we concentrate on bounding  $\Delta$ .

The idea for bounding  $\Delta$  from below is to consider the two steps of  $k$ -means: reassigning data points and adjusting cluster centers. If a center is moved a certain distance, then this results in a potential drop proportional to the square of this distance. Thus, if any center moves a significant distance, then the potential decreases significantly. On the other hand, assume that all centers stay approximately at the same positions. In this case, we consider reassigning the points. Since all centers are roughly fixed, also the boundaries of the Voronoi cells are roughly fixed. Now consider the case that there is some point that is much closer to its new center than it is to its old center. This corresponds to this point being far away from the boundary between its old and new center. Then reassigning this point decreases the potential significantly. Thus, what remains to be done is to show the following: If the centers stay approximately at the same positions for a few iterations—thus, the potential does not drop significantly due to a movement of a center—, then (with high probability, where the probability is over the perturbation of the point set) there must be some point that is reassigned to a new cluster and far away from the corresponding boundary. This can indeed be formalized and proved.

While this is the basic idea behind the proof of the  $\text{poly}(n^k, 1/\sigma)$  bound for the smoothed running time of  $k$ -means [2], a finer division of iterations into different classes is needed to prove a bound of  $\text{poly}(n, 1/\sigma)$  [1]. But this is also roughly the intuition behind the proof of the latter.

## 3 Knapsack Problem

The *knapsack problem* is one of the classical NP-hard optimization problems. An instance of the problem consists of a capacity  $t \in \mathbb{R}_{\geq 0}$  and  $n$  items with profits  $p_1, \dots, p_n \in \mathbb{R}_{\geq 0}$  and weights  $w_1, \dots, w_n \in \mathbb{R}_{\geq 0}$ , respectively. The goal is to find a subset of the items that obeys the capacity constraint and maximizes the profit:

$$\begin{aligned} & \text{maximize} && p \cdot x = p_1x_1 + \dots + p_nx_n \\ & \text{subject to} && w \cdot x = w_1x_1 + \dots + w_nx_n \leq t \\ & && \text{and } x_i \in \{0, 1\} \text{ for all } i \in \{1, \dots, n\}. \end{aligned}$$

Knapsack-like problems often occur in applications, and numerous heuristics for solving them have been developed. These heuristics work very well on real-world instances of the knapsack problem. In the following, we describe the Nemhauser/Ullmann algorithm and analyze its smoothed running time.

### 3.1 Nemhauser/Ullmann Algorithm

We use the term *solution* to refer to a vector from  $\{0, 1\}^n$ , and we say that a solution  $x'$  *dominates* a solution  $x$  if  $p \cdot x \leq p \cdot x'$  and  $w \cdot x \geq w \cdot x'$ , with one of these inequalities being strict. Thus,  $x'$  is a strictly preferable solution than  $x$  because it is both lighter and more profitable. A solution  $x$  is called *Pareto-optimal* if no other solution  $x'$  dominates  $x$ . The Nemhauser/Ullmann algorithm is based on the observation that there always exists an optimal solution that is Pareto-optimal. Hence, if one knows the set  $\mathcal{P}$  of Pareto-optimal solutions, which we will also call the *Pareto set*, one can solve the knapsack problem by finding a solution with largest profit among all solutions  $x \in \mathcal{P}$  with  $w \cdot x \leq t$ .

Nemhauser and Ullmann [22] proposed a simple algorithm for enumerating the Pareto set of a knapsack instance based on *dynamic programming*: Let  $\mathcal{P}(i)$  denote the Pareto set of the instance restricted to the first  $i$  items. Then  $\mathcal{P}(0) = \{(0, \dots, 0)\}$  (the only solution in  $\mathcal{P}(0)$  is the trivial solution, where no item is taken) and  $\mathcal{P}(n) = \mathcal{P}$ . Using dynamic programming, we can compute  $\mathcal{P}(i + 1)$  from  $\mathcal{P}(i)$  in time  $O(|\mathcal{P}(i)|)$ . The Nemhauser/Ullmann algorithm simply computes iteratively the sets  $\mathcal{P}(0), \mathcal{P}(1), \dots, \mathcal{P}(n)$  (Figure 2 shows an example how to compute  $\mathcal{P}(i)$  from  $\mathcal{P}(i - 1)$ ).

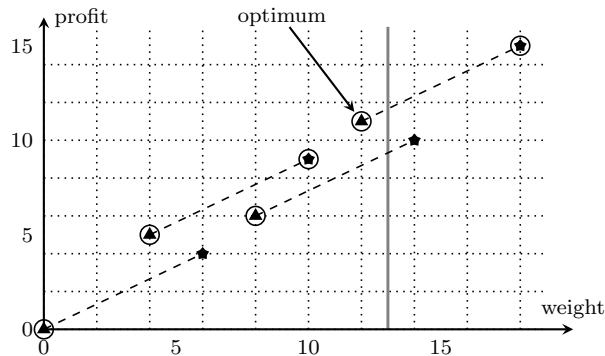


Figure 2: Example with items of weight 8, 4, and 6 and profits 6, 5, and 4, respectively, and  $t = 13$  (gray).  $\mathcal{P}(2)$  consists of the triangles. The stars are the entries of  $\mathcal{P}(2)$  plus the third item.  $\mathcal{P}(3)$  consists of the encircled items. The stars without circles are dominated by some other solution. The optimal solution has thus weight 12 and profit 11.

The total running time of the Nemhauser/Ullmann algorithm for computing the Pareto set  $\mathcal{P}$  and for solving a given knapsack instance optimally is  $O(\sum_{i=1}^n |\mathcal{P}(i)|)$ . This is polynomial on instances with a polynomial number of Pareto-optimal solutions. It is, however, easy to construct instances with exponentially many Pareto-optimal solutions. Thus, the Nemhauser/Ullmann algorithm is not polynomial in the worst case. But still, it works well in practice.

### 3.2 Smoothed Analysis

To explain the success of the Nemhauser/Ullmann algorithm, Beier and Vöcking [7] analyzed the number of Pareto-optimal solutions in perturbed instances of the knapsack problem. In their model, an adversary chooses arbitrary weights  $w_1, \dots, w_n$ , an arbitrary capacity  $t$ , and

a probability density  $f_i: [0, 1] \rightarrow [0, \phi]$  for each profit  $p_i$  according to which it is chosen independently of the other profits, for some parameter  $\phi \geq 1$ . (The restriction to weights and profits from the interval is without loss of generality since any instance can be normalized to fit in this interval.) This model is a generalization of the “traditional” model of smoothed analysis, where the adversary specifies the instance which is afterwards perturbed. In this model, the adversary directly specifies the probability distributions according to which the entries are drawn. The adversary can, 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  $\phi$ , the more concentrated the random variables can be and the closer the smoothed analysis is to a worst-case analysis. The extreme case is that the density functions were allowed to have peaks. In this case, the adversary can specify a worst-case instance precisely. On the other hand, for  $\phi = 1$ , the profits are drawn according to independent uniform distributions. Thus,  $\phi = 1$  corresponds to the average case. Hence, the parameter  $\phi$  takes over the role of  $1/\sigma$  for the Gaussian perturbations in the previous section. This model is more general than Gaussian perturbations as the adversary can not only determine the mean values of the random profits, but also the type of noise for each  $p_i$ .

Beier and Vöcking [7] have shown that the expected number of Pareto-optimal solutions is  $O(n^4\phi)$ , regardless of the choices of the adversary. This result has been improved to  $O(n^2\phi)$  by Beier et al. [5]. This implies that the smoothed running time of the Nemhauser/Ullmann algorithm to solve the knapsack problem is  $O(n^3\phi)$ , which can be viewed as an explanation why this algorithm works well on real-world data.

In fact, the results by Beier et al. [5, 7] are far more general. They not only apply to the knapsack problem, but to any problem with a linear objective function. In the general model, we only assume that there is an arbitrary set of solutions  $\mathcal{S} \subseteq \{0, 1\}^n$ , an arbitrary weight function  $w: \mathcal{S} \rightarrow \mathbb{R}$  that assigns a weight to every solution, and a linear objective function  $p \cdot x$ . If the coefficients  $p_1, \dots, p_n$  are drawn according to probability densities that are bounded by  $\phi$  as described above, the bound of  $O(n^2\phi)$  still applies, regardless of the choices of  $\mathcal{S}$ ,  $w$ , and the densities. The set  $\mathcal{S}$  could, for example, consist of all incidence vectors of paths from a source to a target in a given graph. In this way, one obtains that the smoothed number of Pareto-optimal paths in the bicriteria shortest-path problem is polynomial.

These results can be generalized even further to integer optimization problems where  $\mathcal{S} \subseteq \mathcal{D}^n$  and  $\mathcal{D}$  is a finite set of integers [24]. Furthermore, they can also be generalized to  $d$ -criteria problems with  $d$  linear objective functions with perturbed coefficients, for any constant  $d$  [21, 23]. For these problems the bound on the expected number of Pareto-optimal solutions becomes  $O(n^{2d}\phi^{d(d+1)/2})$ .

## 4 Outlook

Let us conclude with two remarks. First, the polynomial time bound for the  $k$ -means method is  $O(n^{34}k^{34}d^8 \log^4(n)/\sigma^6)$ , which is quite huge. Often in smoothed analysis, results are more qualitative than quantitative. This is because the combination of worst case and average case makes the analysis technically very challenging. However, Spielman and Teng’s bound for the simplex algorithm is  $O(n^{86}d^{55}/\sigma^{30})$ . This bound has been improved dramatically by Vershynin [31] to  $O((d^9 + d^3/\sigma^4) \log^7 n)$ . Thus, there is hope that, with new techniques, the smoothed analysis of  $k$ -means can be improved to yield better bounds.

Second, the insights gained from smoothed analysis sometimes lead to more efficient algo-

rithms. To conclude this survey, let us sketch one example. The most competitive algorithms for the knapsack problem are so-called *core algorithms*. These algorithms sort the items in descending order according to their profit-to-weight ratio. Then the items are inserted in this order until the first item is encountered that does not fit into the knapsack anymore because of the weight constraint. Let  $r$  be its profit-to-weight ratio. Now one can prove that items whose profit-to-weight ratio is way above  $r$  must be contained in every optimal solution while items whose ratio is way below  $r$  cannot be part of an optimal solution. Hence, only *critical* items whose ratio is close to  $r$  remain to be examined. Since there are usually only few critical items, it is often feasible to try all possible combinations of critical items to find the optimal solution.

Based on insights from their smoothed analysis, Beier and Vöcking have combined the concepts of core algorithms and Pareto-optimal solutions [6]. They have proved for certain types of random inputs that the expected running time of core algorithms is polynomial if one does not try all possible combinations of critical items but uses the Nemhauser/Ullmann algorithm instead. They have implemented this combined algorithm and demonstrated that it outperforms the best previous heuristics by orders of magnitude on various classes of random instances [8].

To summarize: Smoothed analysis is a powerful tool for explaining the performance of algorithms where conventional notions failed. And understanding why and when algorithms perform well facilitates the design of better algorithms.

## References

- [1] David Arthur, Bodo Manthey, and Heiko Röglin. Smoothed analysis of the  $k$ -means method. *Journal of the ACM*, 58(5), 2011.
- [2] David Arthur and Sergei Vassilvitskii. Worst-case and smoothed analysis of the ICP algorithm, with an application to the  $k$ -means method. *SIAM Journal on Computing*, 39(2):766–782, 2009.
- [3] Cyril Banderier, René Beier, and Kurt Mehlhorn. Smoothed analysis of three combinatorial problems. In *Proc. of the 28th Int. Symp. on Mathematical Foundations of Computer Science (MFCS)*, volume 2747 of *Lecture Notes in Computer Science*, pages 198–207. Springer, 2003.
- [4] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schäfer, and Tjark Vredeveld. Average case and smoothed competitive analysis of the multilevel feedback algorithm. *Mathematics of Operations Research*, 31(1):85–108, 2006.
- [5] René 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)*, volume 4513 of *Lecture Notes in Computer Science*, pages 53–67. Springer, 2007.
- [6] René Beier and Berthold Vöcking. Probabilistic analysis of knapsack core algorithms. In *Proc. of the 15th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 468–477. SIAM, 2004.
- [7] René Beier and Berthold Vöcking. Random knapsack in expected polynomial time. *Journal of Computer and System Sciences*, 69(3):306–329, 2004.

- [8] René Beier and Berthold Vöcking. An experimental study of random knapsack problems. *Algorithmica*, 45(1):121–136, 2006.
- [9] Markus Bläser, Bodo Manthey, and B. V. Raghavendra Rao. Smoothed analysis of partitioning algorithms for Euclidean functionals. In *Proc. of the 12th Workshop on Algorithms and Data Structures (WADS)*, Lecture Notes in Computer Science, pages 110–121. Springer, 2011.
- [10] Avrim L. Blum and John D. Dunagan. Smoothed analysis of the perceptron algorithm for linear programming. In *Proc. of the 13th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 905–914. SIAM, 2002.
- [11] Endre Boros, Khaled Elbassioni, Mahmoud Fouz, Vladimir Gurvich, Kazuhisa Makino, and Bodo Manthey. Stochastic mean payoff games: Smoothed analysis and approximation schemes. In *Proc. of the 38th Int. Coll. on Automata, Languages and Programming (ICALP), Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 147–158. Springer, 2011.
- [12] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3), 2009.
- [13] Valentina Damerow, Bodo Manthey, Friedhelm Meyer auf der Heide, Harald Räcke, Christian Scheideler, Christian Sohler, and Till Tantau. Smoothed analysis of left-to-right maxima with applications. *ACM Transactions on Algorithms*, to appear.
- [14] 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 Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1295–1304. SIAM, 2007.
- [15] Mahmoud Fouz, Manfred Kufleitner, Bodo Manthey, and Nima Zeini Jahromi. On smoothed analysis of quicksort and Hoare’s find. *Algorithmica*, to appear.
- [16] Mary Inaba, Naoki Katoh, and Hiroshi Imai. Variance-based  $k$ -clustering algorithms by Voronoi diagrams and randomization. *IEICE Trans. Information and Systems*, E83-D(6):1199–1206, 2000.
- [17] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.
- [18] Bodo Manthey and Rüdiger Reischuk. Smoothed analysis of binary search trees. *Theoretical Computer Science*, 378(3):292–315, 2007.
- [19] Bodo Manthey and Heiko Röglin. Improved smoothed analysis of  $k$ -means clustering. In *Proc. of the 20th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 461–470. SIAM, 2009.
- [20] Bodo Manthey and Heiko Röglin. Worst-case and smoothed analysis of  $k$ -means clustering with Bregman divergences. In *Proc. of the 20th Ann. Int. Symp. on Algorithms and Computation (ISAAC)*, volume 5878 of *Lecture Notes in Computer Science*, pages 1024–1033. Springer, 2009.

- [21] Ankur Moitra and Ryan O’Donnell. Pareto optimal solutions for smoothed analysts. In *Proc. of the 43rd Ann. ACM Symp. on Theory of Computing (STOC)*, pages 225–234. ACM, 2011.
- [22] George L. Nemhauser and Zev Ullmann. Discrete dynamic programming and capital allocation. *Management Science*, 15:494–505, 1969.
- [23] Heiko Röglin and Shang-Hua Teng. Smoothed analysis of multiobjective optimization. In *Proc. of the 50th Ann. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 681–690. IEEE, 2009.
- [24] Heiko Röglin and Berthold Vöcking. Smoothed analysis of integer programming. *Mathematical Programming*, 110(1):21–56, 2007.
- [25] Arvind Sankar, Daniel A. Spielman, and Shang-Hua Teng. Smoothed analysis of the condition numbers and growth factors of matrices. *SIAM J. Matrix Anal. Appl.*, 28(2):446–476, 2006.
- [26] Guido Schäfer and Naveen Sivadasan. Topology matters: Smoothed competitiveness of metrical task systems. *Theoretical Computer Science*, 241(1–3):216–246, 2005.
- [27] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of termination of linear programming algorithms. *Mathematical Programming*, 97(1–2):375–404, 2003.
- [28] 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.
- [29] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis: An attempt to explain the behavior of algorithms in practice. *Communications of the ACM*, 52(10):76–84, 2009.
- [30] Andrea Vattani.  $k$ -means requires exponentially many iterations even in the plane. *Discrete and Computational Geometry*, 45(4):596–616, 2011.
- [31] Roman Vershynin. Beyond Hirsch conjecture: Walks on random polytopes and smoothed complexity of the simplex method. *SIAM Journal on Computing*, 39(2):646–678, 2009.