15-859E: Advanced Algorithms **CMU, Spring 2015**
Lecture #29: Smoothed Complexity April 6th/8th, 2015
Lecturer: Anupam Gupta Scribe: Melanie Schmidt

1 Introduction

Smoothed analysis originates from a very influential paper of Spielman and Teng [ST01, ST04]. It provides an alternative to worst-case analysis. Assume that we want to analyze an algorithm's cost, e.g., the running time of an algorithm. Let $\text{cost}(A(I))$ be the cost that the algorithm has for input instance I. We will usually group the possible input instances according to their "size" (depending on the problem, this might be the number of jobs, vertices, variables and so on). Let $\mathcal{I}_n$ be the set of all inputs of 'size' n. For a worst case analysis, we are now interested in

$$\max_{I\in\mathcal{I}_n} \text{cost}(A(I)),$$

the maximum cost for any input of size n. Consider Figure 29.1 and imagine that all instances of size n are lined up on the x-axis. The blue line could be the running time of an algorithm that is fast for most inputs but very slow for a small number of instances. The worst case is the height of the tallest peak. For the green curve, which could for example be the running time of a dynamic programming algorithm, the worst case is a tight bound since all instances induce the same cost.

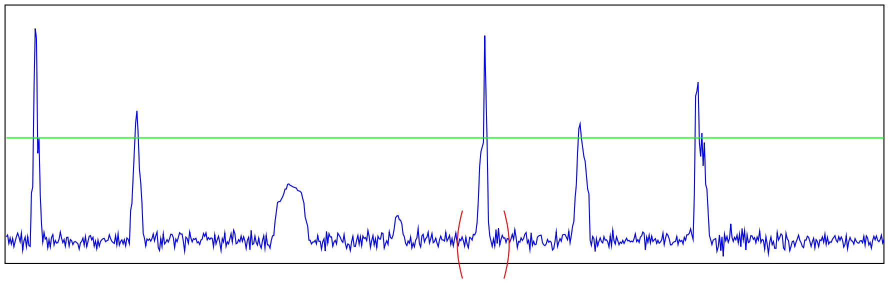
Figure 29.1: An example of a cost function with outliers.

Worst case analysis provides a bound that is true for all instances, but it might be very loose for some or most instances like it happens for the blue curve. When running the algorithm on real world data sets, the worst case instances might not occur, and there are different approaches to provide a more realistic analysis.

The idea behind smoothed analysis is to analyze how an algorithm acts on data that is jittered by a bit of random noise (notice that for real world data, we might have a bit of noise anyway). This is modeled by defining a notion of 'neighborhood' for instances and then analyzing the expected cost of an instance from the neighborhood of I instead of the running time of I.

The choice of the neighborhood is problem specific. We assume that the neighborhood of I is given in form of a probability distribution over all instances in $\mathcal{I}_n$. Thus, it is allowed that all instances in $\mathcal{I}_n$ are in the neighborhood but their influence will be different. The distribution depends on a parameter σ that says how much the input shall be jittered. We denote the neighborhood of I parameterized on σ by $\text{neighborhood}_\sigma(I)$. Then the smoothed complexity is given by

$$\max_{I\in\mathcal{I}_n} E_{I'\sim \text{neighborhood}_\sigma(I)}[\text{cost}(A(I'))]$$

In Figure 29.1, we indicate the neighborhood of one of the peak instances by the red brackets (imagine that the distribution is zero outside of the brackets – we will see one case of this in Section 3). The cost is smoothed within this area. For small σ, the bad instances get more isolated so that they dominate the expected value for their neighborhood, for larger σ, their influence decreases. We want the neighborhood to be big enough to smooth out the bad instances.

So far, we have mostly talked about the intuition behind smoothed analysis. The method has a lot of flexibility since the neighborhood can be defined individually for the analysis of each specific problem. We will see two examples in the following sections.

2 Smoothed complexity of the simplex algorithm

The simplex algorithm [Dan48, Dan51] is a method to solve linear programs. Given a vector $c \in \mathbb{R}^d$ and a matrix $\bar{A} \in \mathbb{R}^{n \times d}$, it finds values for d variables $x_1, \ldots, x_d$ stored in a d-dimensional vector $\bar{x} = (x_1, \ldots, x_d)^t$ that optimize

$$\begin{aligned} \max \quad & \bar{c}^t \bar{x} \\ & \bar{A}\bar{x} \leq \mathbb{1}. \end{aligned}$$

The algorithm iterates through solutions that correspond to vertices of the feasible solution polyhedron until it finds an optimal solution or detects that the linear program is unbounded. It always goes from a solution to one of at least the same value. In case of ties, a pivot rule decides to which solution the algorithm goes. Many pivot rules have been proposed. For most of them, it was shown that there are inputs where the simplex method iterates through an exponential number of vertices and thus has exponential running time. No pivot rule is proven to lead to a polynomial algorithm. However, the simplex method is widely used to solve linear programs.

In their seminal paper, Spielman and Teng show that the simplex algorithm has a polynomial smoothed complexity for a specific pivot rule, the *shadow vertex pivot rule*. More precisely, they the simplex method with this pivot rule provides a polynomial algorithm for the following problem[1]:

Input: vector $c \in \mathbb{R}^d$, matrix $\bar{A} \in \mathbb{R}^{n \times d}$

Problem: For a random matrix $\bar{G} \in \mathbb{R}^{n \times d}$ and a random vector $\bar{g} \in \mathbb{R}^n$ where all entries are chosen independently from a Gaussian distribution $\mathcal{N}(0, \max_i ||a_i||^2)$, solve the following LP:

$$\begin{aligned} \max \quad & \bar{c}^t \bar{x} \\ & (\bar{A} + \bar{G})\bar{x} \leq \mathbb{1} + \bar{g}. \end{aligned}$$

This is one specific neighborhood model. Notice that for any input $(\bar{A}, \bar{c}, \mathbb{1})$, all inputs $(\bar{A}+\bar{G}, \bar{c}, \mathbb{1}, \bar{g})$ are potential neighbors, but the probability decreases exponentially when we go 'further away' from the original input. The vector $\bar{c}$ is not changed, only $\bar{A}$ and $\mathbb{1}$ are jittered. The variance of the Gaussian distributions scales with the smoothness parameter σ. For $\sigma = 0$, the problem reduces to the standard linear programming problem and the analysis becomes a worst case analysis.

Notice that randomly jittering $\bar{A}$ and $\mathbb{1}$ means that the feasibility of the linear program can be switched (from feasible to infeasible or vice versa). Thus, jittering the instance and then solving the LP does not necessarily give any solution for the original LP. However, assuming that the input comes from an appropriate noisy source, the following theorem gives a polynomial running time bound.

[1]They consider even more general problems, where the right hand side value could be some b instead of $\mathbb{1}$.

Theorem 29.1 ([ST04]). *The 'smoothed' number of simplex steps excecuted by the simplex algorithm with the shadow vertex pivot rule is bounded by* $\mathrm{poly}(n, d, 1/\sigma)$ *for the smoothed analysis model described above.*

The original result bounded the number of steps by $\mathcal{O}((nd/\sigma)^{O(1)})$, but the exponents were somewhat large. Vershynin [Ver09] proved an improved bound of $\mathcal{O}(\log^7 n(d^9 + d^3/\sigma^4))$.

2.1 Shadow Vertex Pivot Rule

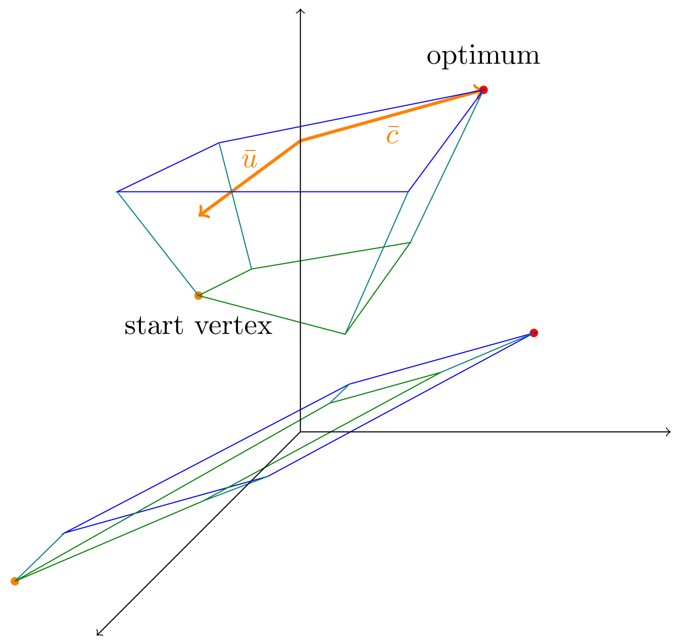

Figure 29.2: Illustration of the shadow vertex pivot rule.

We conclude this section with an informal description of the shadow vertex pivot rule. Consider Figure 29.2. The three-dimensional polytope stands for the polyhedron of all feasible solutions, which is in general d-dimensional. The vector $\bar{c}$ points in the direction in which we want to maximize (it is plotted with an offset to emphasize the optimal vertex). The shadow pivot rule considers a vector $\bar{u}$ which is orthogonal to $\bar{c}$ and satisfies that the start vertex is optimal for the LP

$$\max \bar{u}^T x, \bar{A}x \leq \mathbb{1}.$$

Assume that the original LP has an optimal solution with finite objective value. Then the polyhedron must be bounded in the direction of $\bar{c}$. It is also bounded in the direction of $\bar{u}$ since the start vertex is optimal for $\bar{u}$.

The idea now is to project the polyhedron onto the space spanned by $\bar{c}$ and $\bar{u}$ and then to follow the vertices that lie on the convex hull of the projection (moving into the direction of $\bar{c}$)[2]. Notice that the rightmost vertex on the $\bar{c}$-axis is the projection of an optimal vertex of the original polyhedron.

Since the polyhedron of all feasible solutions is not known, the projection cannot be done upfront. Instead, in each step, the algorithm projects the vectors to the neighbor vertices onto $\mathrm{span}\{\bar{c}, \bar{u}\}$ and identifies the neighbor which is the next on the convex hull.

[2]The two-dimensional object is called the *shadow* of the original polyhedron, hence the name of the pivot rule.

A first step towards proving the result is to show that the shadow has a small number of vertices if the polyhedron $(A+G)x \leq (\mathbb{1}+g)$ is projected onto two fixed vectors $\bar{u}$ and $\bar{c}$. The real result for simplex, however, is complicated by the fact that the vector $\bar{u}$ depends on the polyhedron and on the starting vertex, and so the polyhedron is projected to a subspace that is correlated to the polyhedron itself. Another complication: finding a starting vertex is as hard as solving an LP. Spielman and Teng handle these and other issues; see the original publication [ST04] or in [Lic13].

3 The Nemhauser-Ullman algorithm for the knapsack problem

Now let's turn to a problem for which we will give (almost) all the details of the smoothed analysis. The *knapsack problem* is given the size/weight $w_i \in \mathbb{R}_{\geq 0}$ and the profit $p_i \in \mathbb{R}_{\geq 0}$ of n objects. The problem is to find a subset $S \subset [n]$ that maximizes the profit $\sum_{i\in S} p_i$ while satisfying $\sum_{i\in S} w_i \leq B$ for a knapsack size of $B \in \mathbb{R}_{\geq 0}$. With $w := (w_1, \ldots, w_n)^t$ and $p := (p_1, \ldots, p_n)^t$, the problem can thus be described by

$$\begin{aligned} \max \quad & p^t x \\ & w^t x \leq G \\ & x \in \{0,1\}. \end{aligned}$$

The knapsack problem is weakly NP-hard. It can be solved by dynamic programming. Notice that perturbing the input changes its bit complexity and this might increase the running time of the dynamic programming algorithm instead of decreasing it. We see a smoothed analysis for a different algorithm below. The result is by Beier, Röglin and Vöcking [BV03, BRV07].

3.1 The smoothness model

One natural neighborhood is to smoothe each weight w_i uniformly over an interval of width σ centered at w_i. Figure 29.3 illustrates this. Within the interval, the densitiy function is uniform, outside of the interval, it is zero. The profits are not perturbed.

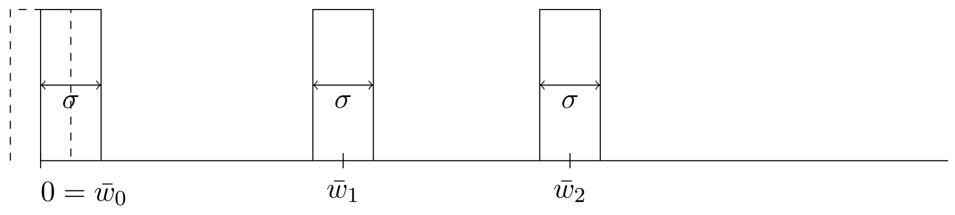

Figure 29.3: Distributions for three weights plotted into the same diagram.

Thus, an adversary chooses the exact profit p_i of each object and the mean $\bar{w}_i$ of each object, and then nature randomly perturbs all weights by adding a number uniformly chosen from $[-\sigma/2, \sigma/2]$. For weights that are smaller than $\sigma/2$, we allow that the box is shifted to the right, this only increases the mean value. One property of such density functions is that they take on value at most $1/\sigma$ for any possible weight. I.e., each w_i is given by a random draw with density function

$$f_i : [0, \infty] \to [0, 1/\sigma].$$

The following analysis allows any distribution on the weights, whose density function is upper bounded by $1/\sigma$; it does not use any other specific properties of the distribution.

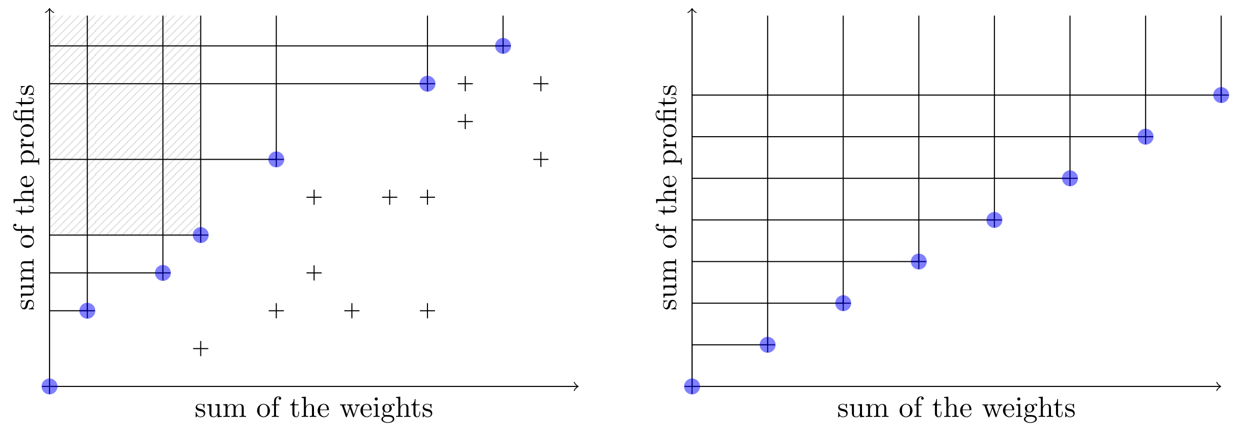

Figure 29.4: Left side: A point set and its Pareto curve. The blue points form the Pareto curve. They belong to the curve because the areas indicated by the black lines does not contain points. For the third point on the curve, this area is specifically highlighted. Right side: Worst Case.

In fact, for the purpose of this lecture, we make the simplifying assumption that we have

$$f_i : [0, 1] \to [0, 1/\sigma]$$

for all $i \in [n]$. This simplification can be avoided, see [BRV07]. We also assume that it never happens that two solutions get exactly the same weight. This is justified since this is an event with zero probability, so it is almost surely not the case.

3.2 The Nemhauser-Ullman algorithm

The Nemhauser-Ullman algorithm [NU69] for the knapsack problem computes the Pareto curve and returns the best solution from the curve. Recall that the Pareto curve for the knapsack problem is defined as follows.

Definition 29.2. Let S_1 and S_2 be two solutions for the knapsack problem. We say that S_2 is *dominated* by S_1 if $\sum_{i \in S_2} p_i \leq \sum_{i \in S_1} p_i$ and $\sum_{i \in S_2} w_i \geq \sum_{i \in S_1} w_i$. A solution S belongs to the *Pareto curve* or *Pareto front* iff it is not dominated by any other solution. It is then *Pareto optimal.*

The definition is visualized in Figure 29.4. The Pareto curve can be computed iteratively. Let $P(j)$ be the set of all Pareto optimal solutions when the instance is restricted to the first j items. Then $P(1) = \{\emptyset, \{1\}\}$ and

$$P(j + 1) \subseteq \big(P(j) \cup \{S \cup j + 1 \mid s \in P(j)\}\big).$$

If we keep the elements of $P(j)$ in sorted order with regard to the weights, then $P(j + 1)$ can be computed in linear time: Construct $P'(j + 1) = \{S \cup j + 1 \mid s \in P(j)\}$ from $P(j)$ by inserting $j + 1$ into all sets, notice that the order stays the same since the sum of the weights is increased by w_{j+1} for all sets. Points in $P(j)$ can only be dominated by sets in $P'(j + 1)$ and vice versa since both sets contain only solutions that do not dominate each other. By traversing $P(j)$ and $P'(j + 1)$ in parallel, we can identify all dominated sets and compute the union in $\mathcal{O}(j) \in \mathcal{O}(n)$ time.

Lemma 29.3. *The Nemhauser-Ullman algorithm can be implemented to have a running time of* $\mathcal{O}(n \cdot \max_{j \in [n]} |P(j)|)$.

Note that for this analysis we no longer care about the size of the knapsack B.

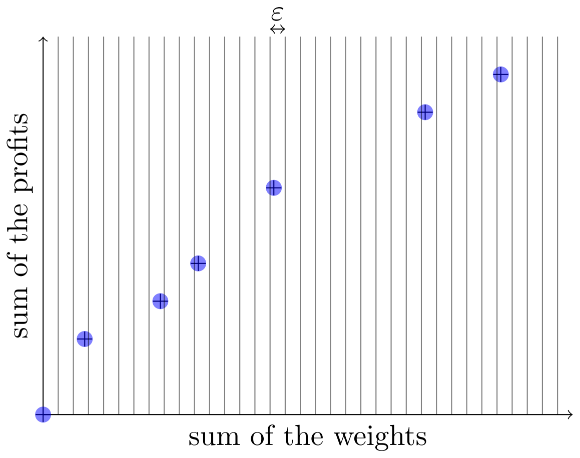

Figure 29.5: Dividing into stripes of width $\varepsilon = 1/k$.

3.3 Bounding Expected Size of Pareto Curve

In the worst-case it is possible that all 2^n solutions are Pareto optimal. (Exercise: how?) In this case, the Nemhauser-Ullman algorithm will take exponential time. However, we will see that the expected size of $P(j)$ is polynomial for all $j \in [n]$ if the instance is σ-smoothed.

Fix any $j \in [n]$, and let $P := P(j)$. The proof idea is to partition the interval $[0, \infty]$ of possible solution weights into stripes of decreasing width $\varepsilon = 1/k$ for an integer k and then to count the number of Pareto optimal solutions in each stripe. See Figure 29.5. There is always an $\varepsilon > 0$ that is small enough such that each stripe contains at most one point from P. Thus,

$$|P| = 1 + \lim_{k\to\infty} \sum_{\ell=0}^{\infty} \mathbb{1}\left(\exists p \in P \cap \left(\tfrac{\ell}{k}, \tfrac{\ell+1}{k}\right]\right)$$

where we count the empty set separately. We want to restrict the number of terms in the sum. Since the knapsack has size G, we can ignore all solutions that weight more than G. However, G might still be large. By our simplification, we know that all weights are at most 1. Thus, no solution can weight more than n and it is thus sufficient to consider the interval $[0, n]$. The kn stripes at $(0, 1/k], (1/k, 2/k], \ldots, (n(k-1)/k, nk/k]$ fit into this interval. We thus get that

$$\mathbf{E}[|P|] \leq 1 + \lim_{k\to\infty} \sum_{\ell=0}^{nk} \mathbf{Pr}\left(\exists p \in P \cap \left(\tfrac{\ell}{k}, \tfrac{\ell+1}{k}\right]\right) \tag{29.1}$$

Now we have to bound the probability that there is a point from P in the interval $(\frac{\ell}{k}, \frac{\ell+1}{k}]$. Fix a $t := \frac{\ell}{k}$ for $\ell \in \{0, \ldots, nk\}$. We define several random variables. First, let

$$x^* := \arg\max_{x \in P} \{p^t x \mid w^t x \leq t\}$$

be the best Pareto optimal solution which lies left of t. We call this solution the *winner*. Since the zero vector is always Pareto optimal there is always at least one point left of t, so x^* is well-defined. It always lies without the interval. The worst Pareto optimal solution right of t is called *least upper bound* or *loser*. It may not exist, so we define it thus:

$$\hat{x} := \begin{cases} \arg\max_{x\in P}\{p^t x \mid w^t x > t\} & \text{if } \{x \mid w^t x > t\} \neq \emptyset \\ \perp & \text{else} \end{cases}$$

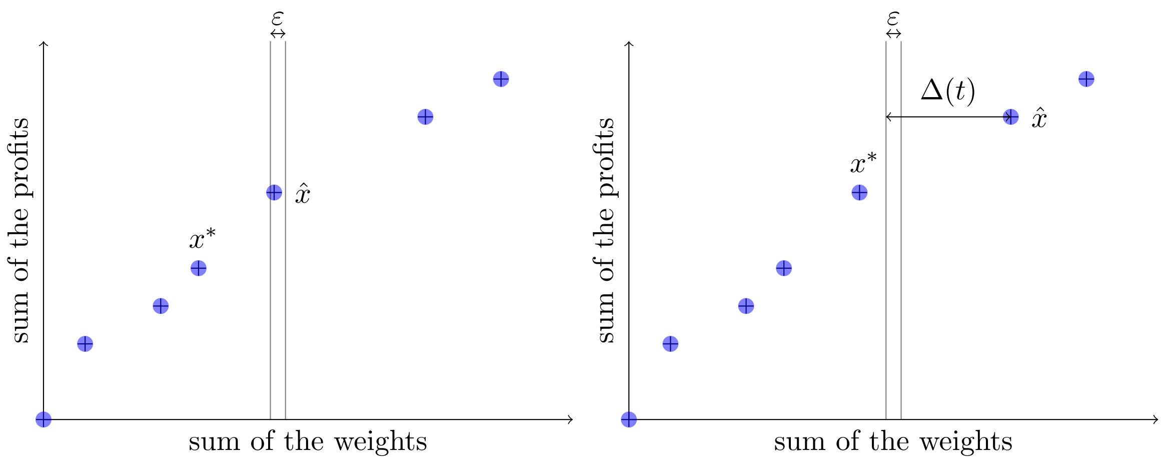

Figure 29.6: Illustration of the definition of x^* and $\hat{x}$. $\Delta(t)$ is only plotted in the right figure.

We define $\Delta(t)$ to be the distance between t and $\hat{x}$:

$$\Delta(t) := \begin{cases} w^t\hat{x} - t & \text{if } \hat{t} \neq \perp \\ \infty & \text{if } \hat{x} = \perp \end{cases}$$

(See Figure 29.6 for a visualization.) The crucial observation is:

$$P \cap \left(\tfrac{\ell}{k}, \tfrac{\ell+1}{k}\right] \neq \emptyset \qquad \iff \qquad \Delta(t) \leq \tfrac{1}{k}$$

The rest of the proof shows that $\mathbf{Pr}\left(\Delta(t) \in (0, \frac{1}{k}]\right)$ is small.

Claim 29.4. $\mathbf{Pr}(\Delta(t) \in (0, \varepsilon]) \leq \frac{n\cdot\varepsilon}{\sigma}$.

Proof. It is difficult to directly argue about $\Delta(t)$, so we use a set of auxiliary random variables which make the proof very clean. For any item $i \in [n]$ define the following:

$$x^{*,i} := \arg \max_{x \in P, x_i = 0} \{p^t x \mid w^t x \leq t\}$$

$$\hat{x}^i := \begin{cases} \arg \min_{x \in P, x_i = 1} \{w^t x \mid p^t x > t\} & \text{if } \{x \mid w^t x > t, x_i = 1\} \neq \emptyset \\ \perp & \text{else} \end{cases}$$

$$\Delta^i(t) := \begin{cases} w^t\hat{x}^i - t & \text{if } \hat{x}^i \neq \perp \\ \infty & \text{else} \end{cases}$$

Notice that if $\hat{x}$ exists, then it must contain at least one item that x^* does not contain. This i satisfies $\Delta(t) = \Delta^i(t)$. (Exercise: Check!) Hence,

Subclaim 1. Either $\Delta(t) = \infty$ or there exists an $i \in [n]$ such that $\Delta(t) = \Delta^i(t)$.

Now fix any item $i \in [n]$ and assume that the weights for all items except i are already chosen. Since $x^{*,i}$ is the best solution to the left of t that does not contain i, we know $x^{*,i}$ by knowing all weights except w_i. Now $\hat{x}^i$ arises from adding the ith item to a solution that does not contain i. The set of all solutions that contain i and whose profit is more than t is independent of any random draw and fixed. The least upper bound $\hat{x}^i$ is chosen among these as the one having the smallest weight, i.e. as the one that minimizes $\sum_{j \neq i} w_j x_j + w_i$. Now when all w_j for $j \neq i$ are fixed, then this sum only depends on w_i and is thus minimized by the same fixed solution, independent of w_i. Thus, $\hat{x}^i$ is actually fixed, but we do not know the value $w^t \hat{x}^i$ due to the random w_i. But we can compute that

$$\mathbf{Pr}(w^t \hat{x}^i \in t + (0, \varepsilon)) = \mathbf{Pr}(w' + w_i \in (t, t + \varepsilon)) = \mathbf{Pr}(w' + w_i - t \in (0, \varepsilon)) \leq \varepsilon / \sigma$$

where the inequality holds because we know that the density function is always bounded by $1/\sigma$. This gives us the following statement.

Subclaim 2. It holds for all $i \in [n]$ that $\mathbf{Pr}(\Delta^i(t) \in (0, \varepsilon]) \leq \varepsilon / \sigma$.

To finish the proof of Claim 29.4:

$$\begin{aligned}
\mathbf{Pr}(\Delta(t) \in (0, \varepsilon]) &\overset{\text{Subclaim 1}}{=} \mathbf{Pr}(\exists i : \Delta^i(t) \in (0, \varepsilon]) \\
&\leq \sum_{i=1}^{n} \mathbf{Pr}(\Delta^i(t) \in (0, \varepsilon]) \\
&\overset{\text{Subclaim 2}}{\leq} n \cdot \varepsilon / \sigma.
\end{aligned}$$

□

We now conclude with the main theorem.

Theorem 29.5. *For σ-smoothed instances, the expected size of P is bounded by n^2/σ for all $j \in [n]$. In particular, the Nemhauser-Ullman algorithm for the knapsack problem has a smoothed complexity of $\mathcal{O}(n^3/\sigma)$ for the smoothness model described in Subsection 3.1.*

Proof. By Inequality (29.1), Lemma 29.3 and Claim 29.4, we conclude that

$$\begin{aligned}
\mathbf{E}[|P|] &\leq 1 + \lim_{k \to \infty} \textstyle\sum_{\ell=0}^{nk} \mathbf{Pr}\left(\exists p \in P \cap \left(\frac{\ell}{k}, \frac{\ell+1}{k}\right]\right) \\
&= 1 + \lim_{k \to \infty} \textstyle\sum_{\ell=0}^{nk} \mathbf{Pr}\left(\Delta(\frac{\ell}{k}) \in \left(0, \frac{1}{k}\right]\right) \\
&\leq 1 + \lim_{k \to \infty} nk \cdot \frac{n/k}{\sigma} = n^2/\sigma.
\end{aligned}$$

□

3.4 More general result

The result can be extended beyond the knapsack problem. Let Π be a "combinatorial optimization" problem given as

$$\begin{aligned}
\max\ & p^t x \\
s.t.\ & Ax \leq b \\
& x \in \mathcal{S}
\end{aligned}$$

where $\mathcal{S} \subseteq \{0, 1\}^n$. Observe: the knapsack problem is one such problem. Beier and Vöcking prove the following theorem.

Theorem 29.6 ([BRV07])**.** *Problem Π has polynomial smoothed complexity if solving Π on unitarily encoded instances can be done in polynomial time.*

For the knapsack problem, the dynamic programming algorithm has a running time of $O(nB)$, where B is the size of the knapsack. If the instance is encoded in binary, then we need $O(n \log B)$ bits for the input and hence this algorithm is not polynomial-time; however if input is encoded in unary then we use $O(nB)$ to encode the instance, and the dynamic programming algorithm becomes polynomial-time.

4 Concluding remarks

We have seen two examples for a smoothed analysis and an important idea for doing such an analysis. In both cases, we have a convex figure (the shadow of a polyhedron, the Pareto curve of all feasible knapsack solutions) which has an exponential number of vertices in the worst case. Perturbing the instance now assures that many points fall 'in the interior' of the convex hull and the complexity of the structure decreases to be polynomial on expectation. The number of vertices was analyzed by subdividing the Euclidean space into pieces and analyzing them separately. This happens in the knapsack analysis, but also in the analysis of the simplex method, where one looks for the probability that a different edge of the shadow leads in and out of a certain area.

References

[BRV07] René Beier, Heiko Röglin, and Berthold Vöcking, *The smoothed number of pareto optimal solutions in bicriteria integer optimization*, Integer Programming and Combinatorial Optimization, 12th International IPCO Conference, Ithaca, NY, USA, June 25-27, 2007, Proceedings, 2007, pp. 53–67. 3, 3.1, 29.6

[BV03] René Beier and Berthold Vöcking, *Random knapsack in expected polynomial time*, Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), 2003, pp. 232 – 241. 3

[Dan48] George B. Dantzig, *Programming in a linear structure*, U.S. Air Force Comptroller, USAF (1948). 2

[Dan51] ———, *Maximization of a linear function of variables subject to linear inequalities*, pp. 339 – 347, 1951. 2

[Lic13] Martin W. Licht, *Smoothed analysis of linear programming*, Diplom thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2013, http://folk.uio.no/martinwl/linpro.pdf. 2.1

[NU69] George L. Nemhauser and Zev Ullmann, *Discrete dynamic programming and capital allocation*, Management Science **15** (1969), 494 – 505. 3.2

[ST01] Daniel A. Spielman and Shang-Hua Teng, *Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time*, Proceedings on 33rd Annual ACM Symposium on Theory of Computing (STOC), 2001, pp. 296 – 305. 1

[ST04] ———, *Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time*, Journal of the ACM **51** (2004), no. 3, 385 – 463. 1, 29.1, 2.1

[Ver09] Roman Vershynin, *Beyond hirsch conjecture: Walks on random polytopes and smoothed complexity of the simplex method*, SIAM Journal on Computing **39** (2009), no. 2, 646 – 678. 2