CS264: Beyond Worst-Case Analysis
Lecture #17: Smoothed Analysis of Local Search*

Tim Roughgarden†

March 7, 2017

1 Context

This lecture and the next are about *smoothed analysis*, which is perhaps the most well-studied "hybrid" analysis framework, blending average-case and worst-case analysis. (Recall our previous examples: semi-random graph models and the random-order model for online algorithms.)

In smoothed analysis, an adversary first picks an input, and nature subsequently adds a "small" perturbation to it. This concept makes sense for computational problems that involve numbers (to make sense of small perturbations). Thus the setup is quite similar to the random-order model from last lecture, with the random re-ordering of the input replaced by a random perturbation.

Like other worst-case/average-case hybrids, smoothed analysis has the benefits of potentially avoiding worst-case inputs (especially if they are "brittle"), and avoiding overfitting a solution to a specific distributional assumption. Something particularly attractive about smoothed analysis is that it has a natural interpretation as a model of "problem formation." To explain, consider a truism that seems downbeat at first: *inaccuracies are inevitable in a problem formulation*. For example, when solving a linear programming problem, there is arguably the "ground truth" version, which is what you would solve if you could perfectly estimate all of the relevant quantities (costs, inventories, future demands, etc.) and then the (probably inaccurate) version that you wind up solving. The latter linear program can be viewed as a perturbed version of the former one. It might be hard to justify any assumptions about the real linear program, but whatever it is, it might be uncontroversial to assume that you instead solve a perturbed version of it. Smoothed analysis is a direct translation of this idea.

†Department of Computer Science, Stanford University, 474 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: tim@cs.stanford.edu.

2 Basic Definitions

Consider a cost measure $\text{COST}(A, z)$ for algorithms A and inputs z. In almost all of the killer applications of smoothed analysis, COST is the running time of A on z.[1] For a fixed input size n and measure σ of perturbation magnitude, the standard definition of the *smoothed complexity of* A is

$$\sup_{\text{inputs } z \text{ of size } n} \left\{ \mathbf{E}_{r(\sigma)}[\text{COST}(A, z + r(\sigma))] \right\}, \tag{1}$$

where "$z + r(\sigma)$" denotes the input obtained from z under the perturbation $r(\sigma)$. For example, z could be an element of $[-1, 1]^n$, and $r(\sigma)$ a spherical Gaussian with directional standard deviation σ. Observe that as the perturbation size σ goes to 0, this measure tends to worst-case analysis. As the perturbation size goes to infinity, the perturbation dwarfs the original input and the measure tends to average-case analysis (under the perturbation distribution). Thus smoothed complexity is an interpolation between the two extremes of worst- and average-case analysis, parameterized by the perturbation size σ.

For the rest of this week, we assume that COST is the running time. We say that an algorithm has *polynomial smoothed complexity* if its smoothed complexity (1) is a polynomial function of both n and $1/\sigma$. In this case, the algorithm runs in expected polynomial running time as long as the perturbations have magnitude at least inverse polynomial in the input size.

3 Smoothed Analysis of the Simplex Method

Recall the problem of linear programming, of going as far as possible in some direction (the direction of the objective function) while staying inside a prescribed polyhedron (i.e., intersection of halfspaces). Recall from Lecture #1 the *simplex algorithm* for linear programming, which effectively does greedy local search along the "edges" of the feasible region, hopping from one vertex to another. Note that the number of vertices of the feasible region can be exponential in the number of dimensions (i.e., variables), as in e.g. the hypercube. The simplex method has exponential worst-case running time (as shown by the Klee-Minty "squashed hypercube" [9]), but it is almost always very fast (typically near-linear) in practice. The first "killer application" of smoothed analysis is the result that the simplex algorithm has polynomial smoothed complexity [13]. This result is too technical to cover in class (even after the simplifications in [6, 15]), but we'll mention a couple of the higher-order bits of the result.

An implementation of the simplex method requires a choice of a pivot rule that dictates which polytope vertex to go to next, when there are multiple adjacent vertices with better objective function values. The result of Spielman and Teng [13] concerns the "shadow pivot rule." This and related pivot rules were considered in the average-case analyses of the simplex method in the 1980s (e.g. [1, 4, 12, 14]). The idea is to project the high-dimensional feasible region onto a plane (the "shadow"), where running the simplex algorithm is easy. Every

[1] For approximation measures, like the approximation or competitive ratio of an algorithm, worst-case inputs tend to remain (nearly) worst-case inputs even after a small perturbation.

vertex in the projection is a vertex of the original polytope, though some of the former's vertices will be sucked into the interior of the shadow. It can be arranged so that the optimal solution appears as a vertex in the shadow (basically, one uses the objective function as one of the vectors defining the two-dimensional projection).[2]

If the simplex method is easy in two dimensions (just go around the polygon as long as the objective function keeps improving), why aren't we done? The issue is that the number of vertices of the shadow can be exponential in the size of the original (high-dimensional) linear program that we started with.[3] So it's easy enough to go around the shadow, but this could conceivably involve an exponential number of steps.

The key result in [13], then, is that the expected number of vertices of the shadow (over the randomness in the perturbations) is polynomial in the input size and $1/\sigma$. The perturbation model is: independently for each entry of the constraint matrix and right-hand side of the linear program, a Gaussian (i.e., normal) random variable is added, with mean 0 and standard deviation σ.

The proof plan in many smoothed analyses, including that in [13] and the one we consider in the next section, follows a two-step approach:

1. First, one identifies a sufficient condition on the data under which the given algorithm has good performance.

2. Second, one shows that the sufficient condition is likely to hold for perturbed data.

Generally the sufficient condition and the consequent algorithm performance are parameterized, and this parameter can be interpreted as a "condition number" of the input.[4] Thus the essence of a smoothed analysis is showing that a perturbed instance is likely to have a "good condition number."

To explain the condition number used in [13], note that with each vertex v of the shadow we can associate an angle of $\pi - \delta_v$ radians, for some $\delta_v > 0$. If you go around the boundary of the shadow, finishing up where you started, the sum of the δ_v's is 2π (why?). Thus if $\delta_v \geq \gamma$ for every v, the number of vertices of the shadow is at most $2\pi/\gamma$. Thus if γ is at least inverse polynomial, this proves a polynomial running time bound on the algorithm. Much of the hard work in [13] is to show that the "condition number" γ is very likely (over the perturbations) to be at least inverse polynomial in the input size and $\frac{1}{\sigma}$.

There are still a number of open questions concerning the smoothed analysis of the simplex method.

[2]For many linear programs, it's trivial or easy to figure out a vertex that can act as a starting point (e.g., in many cases, the all-zero vector). For linear programs with no obvious initial feasible point, one must also run a "Phase I" version of the simplex method to find one. Basically, you allow for inequalities to be violated as some large penalty, and then use the simplex method to minimize the penalty incurred (which will be zero at a point feasible for the original linear program). In any case, the other direction defining the projection should be one optimized by the initial feasible solution (so that it is also a vertex on the shadow).

[3]Whereas the intersection of m explicitly described halfplanes has at most m vertices.

[4]In numerical analysis, the condition number of a problem has a very precise meaning, which is roughly the worst-case fluctuation in the output as a function of perturbations to the input. Here we use the term much more loosely.

1. Perhaps the biggest open question concerns sparsity-preserving perturbations. An alternative explanation for the empirical performance of simplex is to conjecture that "real-world" linear programs have special structure not exhibited by worst-case instances. (Cf., our discussion of nonnegative matrix factorization in Lecture #15.) "Special structure" in linear programs often concerns the patterns of non-zeroes in the constraint matrix. For example, in many network flow problems, there are only a constant number of non-zeroes in each column. But the perturbations in [13] perturb every single entry of the constraint matrix (and right-hand side), thereby eroding its special structure. One would hope that it's good enough to just perturb the non-zero entries of a linear program, but it is a major challenge to prove this.

2. Empirically, the simplex method typically converges in a near-linear number of iterations. The best upper bound on the smoothed complexity of the simplex method known is a significantly bigger polynomial [15]. It would be nice to narrow this gap.

3. Only the shadow pivot rule has been analyzed, and this is not one of the most commonly used pivot rules in practice. It would be nice to establish polynomial smoothed complexity for the most popular pivot rules.

4. It would be good to have simpler proofs of the polynomial smoothed complexity of the simplex method, even if the polynomial in the bound is quite big.

4 Smoothed Analysis of Local Search for the TSP

4.1 The 2-OPT Heuristic

Recall that a local search algorithm for an optimization problem maintains a feasible solution, and iteratively improves that solution via "local moves" as long as is possible, terminating with a locally optimal solution. Local search heuristics are common in practice, in many different application domains. For many such heuristics, the worst-case running time is exponential while the empirical performance on "real-world data" is quite fast (typically a sub-quadratic number of iterations). Discrepancies such as this cry out for a theoretical explanation, and provide an opportunity for smoothed analysis.[5]

We illustrate the smoothed analysis of local search heuristics with a relatively simple example: the 2-OPT heuristic for the Traveling Salesman Problem (TSP) in the plane with the ℓ_1 metric. The input is n points in the square $[0,1] \times [0,1]$. The goal is to define a tour

[5] Another issue with a local search heuristic, that we ignore in this lecture, is that it terminates at a locally optimal solution that could be much worse than a globally optimal one. Here, the gap between theory and practice is not as embarrassing — on real data, local search algorithms can produce pretty lousy solutions. Generally one invokes a local search algorithm many times (either with random starting points or a more clever preprocessing step) and returns the best of all locally optimal solutions found.

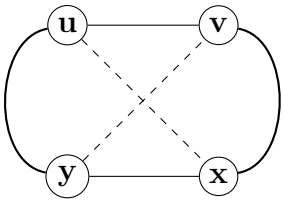

Figure 1: After removing edges $(\mathbf{u}, \mathbf{v})$ and $(\mathbf{x}, \mathbf{y})$ in this tour, the only way to reconnect components is to add edges $(\mathbf{u}, \mathbf{x})$ and $(\mathbf{x}, \mathbf{y})$.

(i.e., a simple n-cycle) $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to minimize

$$\sum_{i=1}^{n} \|\mathbf{x}_{i+1} - \mathbf{x}_i\|_1, \tag{2}$$

where $\mathbf{x}_{n+1}$ is understood to be $\mathbf{x}_1$ and $\|\mathbf{x}\|$ denote the ℓ_1 norm of $\mathbf{x}$, $|x^{(1)}| + |x^{(2)}|$. This problem is NP-hard. It does admit a polynomial-time approximation scheme [3], but it is not particularly practical, and local search is a common way to attack the problem in practice.

We focus on the *2-OPT* local search heuristic. Here, the allowable local moves are swaps: given a tour, one removes two edges, leaving two connected components, and then reconnects the components in the unique way that yields a new tour (Figure 1). Such a swap is *improving* if it yields a tour with strictly smaller objective function value (2). Since there are only $O(n^2)$ possible swaps from a given tour, each iteration of the algorithm can be implemented in polynomial time. In the worst case, however, this local search algorithm might require an exponential number of iterations before finding a locally optimal solution [7].[6] In contrast, the algorithm has polynomial smoothed complexity.[7]

Theorem 4.1 ([7]) *The smoothed complexity of the 2-OPT heuristic for the TSP in the plane with the ℓ_1 metric is polynomial. In particular, the expected number of iterations is $O(\sigma^{-1} n^6 \log n)$.*

Theorem 4.1 is underspecified until we describe the perturbation model, which happily accommodates a range of distributions. As usual, the adversary goes first and chooses an arbitrary set of n points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in $[0,1] \times [0,1]$, which are then randomly perturbed by nature. It's convenient to combine these two steps into one equivalent step, where an adversary picks a density function f_i from which each point $\mathbf{x}_i$ is drawn. We assume the distributions are "not too spiky," meaning that $f_i(\mathbf{x}) \leq \frac{1}{\sigma}$ for all i and $\mathbf{x} \in [0,1] \times [0,1]$. In particular, this forces the support of f_i to have area at least σ (since it has to integrate to 1). For simplicity, we also assume that every point lies in the square $[0,1] \times [0,1]$ with

[6] Similar results had been known for graphs (rather than points in the plane) for a long time [10, 5].

[7] To formally nail down the algorithm, we need to commit to a tie-breaking rule, which decides which improving move to take when there are many (cf., pivot rules for the simplex method). As we'll see from its proof, Theorem 4.1 holds no matter what tie-breaking rule is used.

probability 1. A canonical example of a possible f_i is the uniform distribution over a subset of the square with area σ.

4.2 The High-Level Plan

For Theorem 4.1, our sufficient condition is that every swap that ever gets used by the local search algorithm results in a significantly improved tour. Precisely, consider a local move that begins with some tour, removes the edges $(\mathbf{u}, \mathbf{v}), (\mathbf{x}, \mathbf{y})$, and replaces them with the edges $(\mathbf{u}, \mathbf{x}), (\mathbf{v}, \mathbf{y})$ (as in Figure 1). Note that the decrease in the TSP objective under this swap is

$$\|\mathbf{u} - \mathbf{v}\|_1 + \|\mathbf{x} - \mathbf{y}\|_1 - \|\mathbf{u} - \mathbf{x}\|_1 - \|\mathbf{v} - \mathbf{y}\|_1,$$

that is,

$$|u_1 - v_1| + |u_2 - v_2| + |x_1 - y_1| + |x_2 - y_2| - |u_1 - y_1| - |u_2 - y_2| - |v_1 - x_1| - |v_2 - x_2|. \quad (3)$$

Importantly, this decrease is independent of what the rest of the tour is. We call the swap *ϵ-bad* if (3) is strictly between 0 and ϵ. We'll prove Theorem 4.1 by lower bounding the probability that there are no bad ϵ-swaps (as a function of ϵ) and upper bounding the number of iterations when this condition is satisfied. Thus the parameter ϵ is playing the role of a "condition number" of the input.

4.3 Proof of Theorem 4.1

The key lemma is an upper bound on the probability that there is an ϵ-bad swap.

Lemma 4.2 *For every perturbed instance and $\epsilon > 0$, the probability (over the perturbation) that there is ϵ-bad swap is $O(\epsilon n^4/\sigma)$.*

Let's first see why Lemma 4.2 implies Theorem 4.1.

Proof of Theorem 4.1: Since all points lie in the square $[0, 1] \times [0, 1]$, the ℓ_1 distance between any pair of points is at most 2, and so every tour has length (2) at most $2n$ (and at least 0). Thus, if there are no ϵ-bad swaps, then the local search algorithm must terminate within $\frac{2n}{\epsilon}$ iterations. Also, because every iteration strictly improves the quality of the current tour, the worst-case number of iterations of the algorithm is at most the number of different tours, which is bounded above by $n!$. Using these observations and Lemma 4.2 (with $\frac{2n}{M}$ playing

the role of ϵ), we have

$$\begin{aligned}
\mathbf{E}[\#\text{ of iterations}] &= \sum_{M=1}^{n!} \mathbf{Pr}[\#\text{ of iterations} \geq M] \\
&\leq \sum_{M=1}^{n!} \mathbf{Pr}\left[\text{there is a } \frac{2n}{M}\text{-bad swap}\right] \\
&\leq \sum_{M=1}^{n!} O\left(\frac{n^5}{M\sigma}\right) \\
&= O\left(\sigma^{-1} n^6 \log n\right),
\end{aligned}$$

where the final step uses the estimate $\ln(n!) = O(n \log n)$ for the harmonic series $1 + \frac{1}{2} + \cdots + \frac{1}{n!}$. ■

Proof of Lemma 4.2: Ranging over every improving move that could ever occur (an exponential number of them), there are only $O(n^4)$ fundamentally distinct swaps ($O(1)$ per choice of $\mathbf{u}, \mathbf{v}, \mathbf{x}, \mathbf{y}$). We can therefore complete the proof by showing a bound of $O(\epsilon/\sigma)$ on the probability that a fixed swap is ϵ-bad, and then applying the Union Bound.

Consider a swap, say of $(\mathbf{u}, \mathbf{v}), (\mathbf{x}, \mathbf{y})$ with $(\mathbf{u}, \mathbf{x}), (\mathbf{v}, \mathbf{y})$. Suppose we fix the relative order of u_1, v_1, x_1, y_1 and of u_2, v_2, x_2, y_2. Then, the expression in (3) is linear in the 8 variables, with each variable having a coefficient of -2, 0, or 2 (since each variable appears in two terms, in each case with a ± 1 coefficient). Ranging over all pairs of relative orders, there are $(4!)^2$ such linear combinations. The point is: *the expression* (3) *lies in* $(0, \epsilon)$ *only if one of these* $(4!)^2$ *linear combinations lies in* $(0, \epsilon)$ (since with probability 1, it is equal to one of the linear combinations).

Finally, we claim that for each fixed linear combination derived from (3), the probability that is lies in $(0, \epsilon)$ is at most $\epsilon/2\sigma$. (This completes the proof, after taking a union bound over the $O(1)$ linear combinations.) We can assume that at least one of the coefficients has a coefficient of ± 2. (If they all have zero coefficients, then the linear combination is certainly not in $(0, \epsilon)$.) Condition on the values of all of the variables save for one with a nonzero coefficient, say u_1 (the other cases are the same). The value of the linear combination is then some fixed number z (the contributions of the other 7 variables) plus either $2u_1$ or $-2u_1$. This means the range of values for u_1 that cause the linear combination to lie in $(0, \epsilon)$ is an interval of length at most $\epsilon/2$. Since the density is bounded above by $1/\sigma$ everywhere, the probability that u_1 lies in this interval is at most $\epsilon/2\sigma$, as claimed. ■

There is still a big gap between the polynomial bound in Theorem 4.1 and the observed empirical performance of the 2-OPT algorithm (which is quadratic or better). It is possible to save a factor of n^2 in the bound in Theorem 4.1 with significant extra work. The idea is to define a refined condition number that concerns *pairs* of swaps rather than single swaps (see [7]). Getting still better bounds appears to be technically challenging. It is also possible to extend this analysis to handle any constant number of dimensions (see Homework #9) and other norms (see [7]).

5 Open Questions

While there have been some successes in the smoothed analysis of local search algorithms, there is still much we do not know. It is plausible that for every "natural" local search problem (in some sense), the canonical local search algorithm has polynomial smoothed complexity.

One concrete challenge is the Maximum Cut problem with the "flip" neighborhood. Recall from Lecture #8 that in the Maximum Cut problem, the input is an undirected graph $G = (V, E)$ in which every edge e has a nonnegative weight w_e. By scaling, we can assume that the edge weights lie in $[0, 1]$. Feasible solutions correspond to cuts, meaning partitions (S, T) of V into two sets. Local moves correspond to moving a single vertex from one side of the cut to the other. In the worst case, local search can require an exponential number of iterations, no matter how ties are broken [11]. But as with other local search problems, the bad examples are brittle and are not robust to small perturbations. This problem would seem to be right in the wheelhouse of smoothed complexity. And indeed, the problem is known to have smoothed polynomial complexity if the maximum degree of G is $O(\log |V|)$ (Homework #9) or if G is the complete graph [2]. It is also known to have quasi-polynomial (meaning $n^{O(\log n)}$) smoothed complexity for every graph [8]. But we still don't know if local search has polynomial smoothed complexity for this problem in general.

References

[1] I. Adler and N. Megiddo. A simplex algorithm whose average number of steps is bounded between two quadratic functions of the smaller dimension. *Journal of the ACM*, 32(4):871–895, 1985.

[2] O. Angel, S. Bubeck, Y. Peres, and F. Wai. Local max-cut in smoothed polynomial time. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC)*, pages 429–437, 2017.

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

[4] K. H. Borgwardt. *The Simplex Method: a probabilistic analysis*. Springer-Verlag, 1980.

[5] B. Chandra, H. Karloff, and C. Tovey. New results on the old k-opt algorithm for the traveling salesman problem. *SIAM Journal on Computing*, 28(6):1998–2029, 1999.

[6] A. Deshpande and D. A. Spielman. Improved smoothed analysis of the shadow vertex simplex method. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 349–356, 2005.

[7] M. Englert, H. Röglin, and B. Vöcking. Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP. *Algorithmica*, 68(1):190–264, 2014.

[8] M. Etscheid and H. Röglin. Smoothed analysis of local search for the maximum-cut problem. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 882–889, 2014.

[9] V. Klee and G. J. Minty. How Good is the Simplex Algorithm? In O. Shisha, editor, *Inequalities III*, pages 159–175. Academic Press Inc., New York, 1972.

[10] G. S. Lueker. Unpublished manuscript. Princeton University, 1975.

[11] A. A. Schäffer and M. Yannakakis. Simple local search problems that are hard to solve. *SIAM Journal on Computing*, 20(1):56–87, 1991.

[12] S. Smale. On the average number of steps in the simplex method of linear programming. *Mathematical Programming*, 27:241–242, 1983.

[13] D. A. Spielman and S.-H. Teng. Smoothed analysis: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004.

[14] M. Todd. Polynomial expected behavior of a pivoting algorithm for linear complementarity and linear programming problems. *Mathematical Programming*, 35:173–192, 1986.

[15] R. Vershynin. Beyond hirsch conjecture: Walks on random polytopes and smoothed complexity of the simplex method. *SIAM Journal on Computing*, 39(2):646–678, 2009.