Smoothed Analysis: An Attempt to Explain the Behavior of Algorithms in Practice*

Daniel A. Spielman
Department of Computer Science
Yale University
spielman@cs.yale.edu

Shang-Hua Teng†
Department of Computer Science
Boston University
shanghua.teng@gmail.com

ABSTRACT

Many algorithms and heuristics work well on real data, despite having poor complexity under the standard worst-case measure. Smoothed analysis [36] is a step towards a theory that explains the behavior of algorithms in practice. It is based on the assumption that inputs to algorithms are subject to random perturbation and modification in their formation. A concrete example of such a smoothed analysis is a proof that the simplex algorithm for linear programming usually runs in polynomial time, when its input is subject to modeling or measurement noise.

1. MODELING REAL DATA

> "*My experiences also strongly confirmed my previous opinion that the best theory is inspired by practice and the best practice is inspired by theory.* "
>
> [Donald E. Knuth: "Theory and Practice", *Theoretical Computer Science*, 90 (1), 1–15, 1991.]

Algorithms are high-level descriptions of how computational tasks are performed. Engineers and experimentalists design and implement algorithms, and generally consider them a success if they work in practice. However, an algorithm that works well in one practical domain might perform poorly in another. Theorists also design and analyze algorithms, with the goal of providing provable guarantees about their performance. The traditional goal of theoretical computer science is to prove that an algorithm performs well in the worst case: if one can prove that an algorithm performs well in the worst case, then one can be confident that it will work well in every domain. However, there are many algorithms that work well in practice that do not work well in the worst case. Smoothed analysis provides a theoretical framework for explaining why some of these algorithms do work well in practice.

The performance of an algorithm is usually measured by its running time, expressed as a function of the input size of the problem it solves. The performance profiles of algorithms across the landscape of input instances can differ greatly and can be quite irregular. Some algorithms run in time linear in the input size on all instances, some take quadratic or higher order polynomial time, while some may take an exponential amount of time on some instances.

Traditionally, the complexity of an algorithm is measured by its *worst-case* performance. If a single input instance triggers an exponential run time, the algorithm is called an exponential-time algorithm. A polynomial-time algorithm is one that takes polynomial time on all instances. While polynomial time algorithms are usually viewed as being efficient, we clearly prefer those whose run time is a polynomial of low degree, especially those that run in nearly linear time.

It would be wonderful if every algorithm that ran quickly in practice was a polynomial-time algorithm. As this is not always the case, the worst-case framework is often the source of discrepancy between the theoretical evaluation of an algorithm and its practical performance.

It is commonly believed that practical inputs are usually more favorable than worst-case instances. For example, it is known that the special case of the Knapsack problem in which one must determine whether a set of n numbers can be divided into two groups of equal sum does not have a polynomial-time algorithm, unless **NP** is equal to **P**. Shortly before he passed away, Tim Russert of the NBC's "Meet the Press," commented that the 2008 election could end in a tie between the Democratic and the Republican candidates. In other words, he solved a 51 item Knapsack problem[1] by hand within a reasonable amount of time, and most

*This material is based upon work supported by the National Science Foundation under Grants No. CCR-0325630 and CCF-0707522. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
Because of CACM's strict constraints on bibliography, we have to cut down the citations in this writing. We will post a version of the article with more complete bibliograph on our webpage.

†Affliation after the summer of 2009: Department of Computer Science, University of Southern California.

[1]In presidential elections in the United States, each of the 50 states and the District of Columbia is allocated a number of electors. All but the states of Maine and Nebraska use winner-take-all system, with the candidate winning the majority votes in each state being awarded all of that states electors. The winner of the election is the candidate who is awarded the most electors. Due to the exceptional behavior of Maine and Nebraska, the problem of whether the general election could end with a tie is not a perfect Knapsack

likely without using the pseudo-polynomial-time dynamic-programming algorithm for Knapsack!

In our field, the simplex algorithm is the classic example of an algorithm that is known to perform well in practice but has poor worst-case complexity. The simplex algorithm solves a linear program, for example, of the form,

$$\max \quad \mathbf{c}^T\mathbf{x} \qquad \text{subject to} \quad \mathbf{A}\mathbf{x} \leq \mathbf{b}, \tag{1}$$

where $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{b}$ is an m-place vector, and $\mathbf{c}$ is an n-place vector. In the worst case, the simplex algorithm takes exponential time [25]. Developing rigorous mathematical theories that explain the observed performance of practical algorithms and heuristics has become an increasingly important task in Theoretical Computer Science. However, modeling observed data and practical problem instances is a challenging task as insightfully pointed out in the 1999 "Challenges for Theory of Computing" Report for an NSF-Sponsored Workshop on Research in Theoretical Computer Science[2].

> "While theoretical work on models of computation and methods for analyzing algorithms has had enormous payoff, we are not done. In many situations, simple algorithms do well. Take for example the Simplex algorithm for linear programming, or the success of simulated annealing of certain supposedly intractable problems. We don't understand why! It is apparent that worst-case analysis does not provide useful insights on the performance of algorithms and heuristics and our models of computation need to be further developed and refined. Theoreticians are investing increasingly in careful experimental work leading to identification of important new questions in algorithms area. Developing means for predicting the performance of algorithms and heuristics on real data and on real computers is a grand challenge in algorithms".

Needless to say, there are a multitude of algorithms beyond simplex and simulated annealing whose performance in practice is not well-explained by worst-case analysis. We hope that theoretical explanations will be found for the success in practice of many of these algorithms, and that these theories will catalyze better algorithm design.

2. THE BEHAVIOR OF ALGORITHMS

When A is an algorithm for solving problem P we let $T_A[\mathbf{x}]$ denote the running time of algorithm A on an input instance $\mathbf{x}$. If the input domain Ω has only one input instance $\mathbf{x}$, then we can use the instance-based measure $T_{A_1}[\mathbf{x}]$ and $T_{A_2}[\mathbf{x}]$ to decide which of the two algorithms A_1 and A_2 more efficiently solves P. If Ω has two instances $\mathbf{x}$ and $\mathbf{y}$, then the instance-based measure of an algorithm A defines a two dimensional vector $(T_A[\mathbf{x}], T_A[\mathbf{y}])$. It could be the case that $T_{A_1}[\mathbf{x}] < T_{A_2}[\mathbf{x}]$ but $T_{A_1}[\mathbf{y}] > T_{A_2}[\mathbf{y}]$. Then, strictly speaking, these two algorithms are not comparable. Usually, the input domain is much more complex, both in theory and in practice. The *instance-based complexity measure* $T_A[\cdot]$ defines an $|\Omega|$ dimensional vector when Ω is finite. In general, it can be viewed as a function from Ω to $\mathbb{R}^1_+$. But, it is unwieldy. To compare two algorithms, we require a more concise complexity measure.

An input domain Ω is usually viewed as the union of a family of subdomains $\{\Omega_1, \ldots, \Omega_n, \ldots\}$, where Ω_n represents all instances in Ω of size n. For example, in sorting, Ω_n is the set of all tuples of n elements; in graph algorithms, Ω_n is the set of all graphs with n vertices; and in computational geometry, we often have $\Omega_n \in \mathbb{R}^n$. In order to succinctly express the performance of an algorithm A, for each Ω_n one defines scalar $T_A(n)$ that summarizes the instance-based complexity measure, $T_A[\cdot]$, of A over Ω_n. One often further simplifies this expression by using big-O or big-Θ notation to express $T_A(n)$ asymptotically.

2.1 Traditional Analyses

It is understandable that different approaches to summarizing the performance of an algorithm over Ω_n can lead to very different evaluations of that algorithm. In Theoretical Computer Science, the most commonly used measures are the worst-case measure and the average-case measures.

The *worst-case measure* is defined as

$$\mathsf{WC}_A(n) = \max_{\mathbf{x} \in \Omega_n} T_A[\mathbf{x}].$$

The *average-case measures* have more parameters. In each average-case measure, one first determines a distribution of inputs and then measures the expected performance of an algorithm assuming inputs are drawn from this distribution. Supposing $\mathcal{S}$ provides a distribution over each Ω_n, the average-case measure according to $\mathcal{S}$ is

$$\mathsf{Ave}^{\mathcal{S}}_A(n) = \mathop{\mathbf{E}}_{\mathbf{x} \in_{\mathcal{S}} \Omega_n} [\ T_A[\mathbf{x}]\ ],$$

where we use $\mathbf{x} \in_{\mathcal{S}} \Omega_n$ to indicate that $\mathbf{x}$ is randomly chosen from Ω_n according to distribution $\mathcal{S}$.

2.2 Critique of Traditional Analyses

Low worst-case complexity is the gold standard for an algorithm. When low, the worst-case complexity provides an absolute guarantee on the performance of an algorithm no matter which input it is given. Algorithms with good worst-case performance have been developed for a great number of problems.

However, there are many problems that need to be solved in practice for which we do not know algorithms with good worst-case performance. Instead, scientists and engineers typically use heuristic algorithms to solve these problems. Many of these algorithms work well in practice, in spite of having a poor, sometimes exponential, worst-case running time. Practitioners justify the use of these heuristics by observing that worst-case instances are usually not "typical" and rarely occur in practice. The worst-case analysis can be too pessimistic. This theory-practice gap is not limited to heuristics with exponential complexity. Many polynomial time algorithms, such as interior-point methods for linear programming and the conjugate gradient algorithm for solving linear equations, are often much faster than their worst-case bounds would suggest. In addition, heuristics are often used to speed up the practical performance of implementations that are based on algorithms with polynomial worst-case complexity. These heuristics might in fact worsen the worst-case performance, or make the worst-case complexity difficult to analyze.

problem. But one can still efficiently formulate it as one.

[2]Available at http://sigact.acm.org/

Average-case analysis was introduced to overcome this difficulty. In average-case analysis, one measures the expected running time of an algorithm on some distribution of inputs. While one would ideally choose the distribution of inputs that occur in practice, this is difficult as it is rare that one can determine or cleanly express these distributions, and the distributions can vary greatly between one application and another. Instead, average-case analyses have employed distributions with concise mathematical descriptions, such as Gaussian random vectors, uniform $\{0,1\}$ vectors, and Erdös-Rényi random graphs.

The drawback of using such distributions is that the inputs actually encountered in practice may bear very little resemblance to the inputs that are likely to be generated by such distributions. For example, one can see what a random image looks like by disconnecting most TV sets from their antennas, at which point they display "static". These random images do not resemble actual television shows. More abstractly, Erdös-Rényi random graph models are often used in average-case analyses of graph algorithms. The Erdös-Rényi distribution $G(n,p)$, produces a random graph by including every possible edge in the graph independently with probability p. While the average degree of a graph chosen from $G(n, 6/(n-1))$, is approximately six, such a graph will be very different from a the graph of a triangulation of points in two dimensions, which will also have average degree approximately six.

In fact, random objects such as random graphs and random matrices have special properties with exponentially high probability, and these special properties might dominate the average-case analysis. Edelman [14] writes of random matrices:

> What is a mistake is to psychologically link a random matrix with the intuitive notion of a "typical" matrix or the vague concept of "any old matrix."
>
> In contrast, we argue that "random matrices" are *very* special matrices.

2.3 Smoothed Analysis: A Step towards Modeling Real Data

Because of the intrinsic difficulty in defining practical distributions, we consider an alternative approach to modeling real data. The basic idea is to identify typical properties of practical data, define an input model that captures these properties, and then rigorously analyze the performance of algorithms assuming their inputs have these properties.

Smoothed analysis is a step in this direction. It is motivated by the observation that practical data are often subject to some small degree of random noise. For example,

- in industrial optimization and economic prediction, the input parameters could be obtained by physical measurements, and measurements usually have some of low magnitude uncertainty;
- in the social sciences, data often come from surveys in which subjects provide integer scores in a small range (say between 1 and 5) and select their score with some arbitrariness;
- even in applications where inputs are discrete, there might be randomness in the formation of inputs. For instance, the network structure of the Internet may very well be governed by some "blueprints" of the government and industrial giants, but it is still "perturbed" by the involvements of smaller Internet Service Providers.

In these examples, the inputs usually are neither completely random nor completely arbitrary. At a high level, each input is generated from a two-stage model: In the first stage, an instance is generated and in the second stage, the instance from the first stage is slightly perturbed. The perturbed instance is the input to the algorithm.

In smoothed analysis, we assume that an input to an algorithm is subject to a slight random perturbation. The *smoothed measure* of an algorithm on an input instance is its expected performance over the perturbations of that instance. We define the *smoothed complexity* of an algorithm to be the maximum smoothed measure over input instances.

For concreteness, consider the case $\Omega_n = \mathbb{R}^n$, which is a common input domain in computational geometry, scientific computing, and optimization. For these continuous inputs and applications, the family of Gaussian distributions provides a natural models of noise or perturbation.

Recall that a univariate Gaussian distribution with mean 0 and standard deviation σ has density

$$\frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2}.$$

The standard deviation measures the magnitude of the perturbation. A *Gaussian random vector* of variance σ^2 centered at the origin in $\Omega_n = \mathbb{R}^n$ is a vector in which each entry is an independent Gaussian random variable of standard deviation σ and mean 0. For a vector $\bar{\mathbf{x}} \in \mathbb{R}^n$, a σ-Gaussian perturbation of $\bar{\mathbf{x}}$ is a random vector $\mathbf{x} = \bar{\mathbf{x}} + \mathbf{g}$, where $\mathbf{g}$ is a Gaussian random vector of variance σ^2. The standard deviation of the perturbation we apply should be related to the norm of the vector it perturbs. For the purposes of this paper, we relate the two by restricting the unperturbed inputs to lie in $[-1,1]^n$. Other reasonable approaches are taken elsewhere.

Definition 1 (Smoothed Complexity). *Suppose A is an algorithm with $\Omega_n = \mathbb{R}^n$. Then, the smoothed complexity of A with σ-Gaussian perturbations is given by*

$$\mathsf{Smoothed}_A^\sigma(n) = \max_{\bar{\mathbf{x}} \in [-1,1]^n} \mathop{\mathbf{E}}_{\mathbf{g}} [\, T_A(\bar{\mathbf{x}} + \mathbf{g}) \,],$$

where $\mathbf{g}$ is a Gaussian random vector of variance σ^2.

In this definition, the "original" input $\bar{\mathbf{x}}$ is perturbed to obtain the input $\bar{\mathbf{x}} + \mathbf{g}$, which is then fed to the algorithm. For each original input, this measures the expected running time of algorithm A on random perturbations of that input. The maximum out front tells us to measure the smoothed analysis by the expectation under the worst possible original input.

The smoothed complexity of an algorithm measures the performance of the algorithm both in terms of the input size n and in terms of the magnitude σ of the perturbation. By varying σ between zero and infinity, one can use smoothed analysis to interpolate between worst-case and average-case analysis. When $\sigma = 0$, one recovers the ordinary worst-case analysis. As σ grows large, the random perturbation $\mathbf{g}$ dominates the original $\bar{\mathbf{x}}$, and one obtains an average-case analysis. We are most interested in the situtation in which σ is small relative to $\|\bar{\mathbf{x}}\|$, in which case $\bar{\mathbf{x}} + \mathbf{g}$ may be

interpreted as a slight perturbation of $\bar{\mathbf{x}}$. The dependence on the magnitude σ is essential and much of the work in smoothed analysis demonstrates that noises often make a problem easier to solve.

DEFINITION 2. *A has* **polynomial smoothed complexity** *if there exist positive constants n_0, σ_0, c, k_1 and k_2 such that for all $n \geq n_0$ and $0 \leq \sigma \leq \sigma_0$,*

$$\mathsf{Smoothed}_A^{\sigma}(n) \leq c \cdot \sigma^{-k_2} \cdot n^{k_1}, \tag{2}$$

From Markov's inequality, we know that if an algorithm A has smoothed complexity $T(n, \sigma)$, then

$$\max_{\bar{\mathbf{x}} \in [-1,1]^n} \Pr_{\mathbf{g}} \left[T_A(\bar{\mathbf{x}} + \mathbf{g}) \leq \delta^{-1} T(n, \sigma) \right] \geq 1 - \delta. \tag{3}$$

Thus, if A has polynomial smoothed complexity, then for any $\bar{\mathbf{x}}$, with probability at least $(1 - \delta)$, A can solve a random perturbation of $\bar{\mathbf{x}}$ in time polynomial in n, $1/\sigma$, and $1/\delta$. However, the probabilistic upper bound given in (3) does not necessarily imply that the smoothed complexity of A is $O(T(n, \sigma))$. Blum and Dunagan [6] and subsequently Beier and Vöcking [5] introduced a relaxation of polynomial smoothed complexity.

DEFINITION 3. *A has* **probably polynomial smoothed complexity** *if there exist constants n_0, σ_0, c, and α, such that for all $n \geq n_0$ and $0 \leq \sigma \leq \sigma_0$,*

$$\max_{\bar{\mathbf{x}} \in [-1,1]^n} \mathop{\mathbf{E}}_{\mathbf{g}} \left[\; T_A \left(\bar{\mathbf{x}} + \mathbf{g} \right)^{\alpha} \; \right] \leq c \cdot \sigma^{-1} \cdot n. \tag{4}$$

They show that some algorithms have probably polynomial smoothed complexity, in spite of the fact that their smoothed complexity according to Definition 1 is unbounded.

3. EXAMPLES OF SMOOTHED ANALYSIS

In this section, we give a few examples of smoothed analysis. We organize them in five categories: mathematical programming, machine learning, numerical analysis, discrete mathematics, and combinatorial optimization. For each example, we will give the definition of the problem, state the worst-case complexity, explain the perturbation model, and state the smoothed complexity under the perturbation model.

3.1 Mathematical Programming

The typical problem in Mathematical programming is the optimization of an objective function subject to a set of constraints. Because of its importance to economics, management science, industry and military planning, many optimization algorithms and heuristics have been developed, implemented and applied to practical problems. Thus, this field provides a great collection of algorithms for smoothed analysis.

Linear Programming

Linear programming is the most fundamental optimization problem. A typical linear program is given in Eqn. (1). The most commonly used linear programming algorithms are the simplex algorithm [12] and the interior-point algorithms.

The simplex algorithm, first developed by Dantzig in 1951 [12], is a family of iterative algorithms. Most of them are two-phase algorithms: Phase I determines whether a given linear program is infeasible, unbounded in the objective direction, or feasible with a bounded solution, in which case, a vertex $\mathbf{v}_0$ of the feasible region is also computed. Phase II is iterative: in the i^{th} iteration, the algorithm finds a neighboring vertex $\mathbf{v}_i$ of $\mathbf{v}_{i-1}$ with better objective value or terminates by returning $\mathbf{v}_{i-1}$ when no such neighboring vertex exists. The simplex algorithms differ in their pivot rules, which determine which vertex $\mathbf{v}_i$ to choose when there are multiple choices. Several pivoting rules have been proposed However, almost all existing pivot rules are known to have exponential worst-case complexity [25].

Spielman and Teng [36] considered the smoothed complexity of the simplex algorithm with the shadow-vertex pivot rule, developed by Gass and Saaty [18]. They used Gaussian perturbations to model noise in the input data and proved that the smoothed complexity of this algorithm is polynomial. Vershynin [38] improved their result to obtain a smoothed complexity of

$$O\left(\max\left(n^5 \log^2 m, n^9 \log^4 n, n^3 \sigma^{-4}\right)\right).$$

See [13, 6] for smoothed analyses of other linear programming algorithms such as the interior-point algorithms and the perceptron algorithm.

Quasi-Concave Minimization

Another fundamental optimization problem is quasi-concave minimization. Recall that a function $f : \mathbb{R}^n \to \mathbb{R}$ is quasi-concave if all of its upper level sets $L_\gamma = \{x | f(x) \geq \gamma\}$ are convex. In quasi-concave minimization, one is asked to find the minimum of a quasi-concave function subject to a set of linear constraints. Even when restricted to concave quadratic functions over the hypercube, concave minimization is NP-hard.

In applicationbs such as stochastic and multi-objective optimization, one often deals with data from low-dimensional subspaces. In other words, one needs to solve a quasi-concave minimization problem with a low-rank quasi-concave function [23]. Recall that a function $f : \mathbb{R}^n \to \mathbb{R}$ has rank k if it can be written in the form

$$f(x) = g(a_1^T x, a_2^T x, \ldots, a_k^T x),$$

for a function $g : \mathbb{R}^k \to \mathbb{R}$ and linearly independent vectors $a_1, a_2, \ldots, a_k$.

Kelner and Nikolova [23] proved that, under some mild assumptions on the feasible convex region, if k is a constant then the smoothed complexity of quasi-concave minimization is polynomial when f is perturbed by noise. Key to their analysis is a smoothed bound on the size of the k-dimensional shadow of the high-dimensional polytope that defines the feasible convex region. There result is a non-trivial extension of the analysis of 2-dimensional shadows of [36, 24].

3.2 Machine Learning

Machine Learning provides many natural problems for smoothed analysis. The field has many heursitics that work in practice, but not in the worst case, and the data for most machine learning problems is inherently noisy.

K-means

One of the fundamental problems in Machine Learning is that of k-means clustering: the partitioning of a set of d-dimensional vectors $Q = \{q_1, \ldots, q_n\}$ into k clusters $\{Q_1, \ldots, Q_k\}$

so that the intra-cluster variance

$$V = \sum_{i=1}^{k} \sum_{q_j \in Q_i} \|q_j - \mu(Q_i)\|^2,$$

is minimized, where $\mu(Q_i) = \left(\sum_{q_j \in Q_i} q_j\right) / |Q_i|$ is the centroid of Q_i.

One of the most widely used clustering algorithms is Lloyd's algorithm [27]. It first chooses an arbitrary set of k centers and then uses the Voronoi diagram of these centers to partition Q into k clusters. It then repeats the following process until it stabilizes: use the centroids of the current clusters as the new centers, and then re-partition Q accordingly.

Two important questions about Lloyd's algorithm are how many iterations its takes to converge, and how close to an optimal solution does it find. Arthur and Vassilvitskii proved that in the worst-case, Lloyd's algorithm requires $2^{\Omega(\sqrt{n})}$ iterations to converge [2]. Fix: Focusing on the iteration complexity, Arthur, Manthey, and Röglin [10] recently settled an early conjecture of Arthur and Vassilvitskii by showing that Llyod's algorithm has polynomial smoothed complexity.

Perceptrons, Margins and Support Vector Machines

Blum and Dunagan's analysis of the perceptron algorithm [6] for linear programming implicitly contains results of interest in Machine learning. The ordinary perceptron algorithm solves a fundamental problem in Machine Learning: given a collection of points $x_1, \ldots, x_n \in \mathbb{R}^d$ and labels $b_1, \ldots, b_n \in \{\pm 1\}^n$, find a hyperplane separating the positively labeled examples from the negatively labeled ones, or determine that no such plane exists. Under a smoothed model in which the points $x_1, \ldots, x_n$ are subject to a σ-Gaussian perturbation, Blum and Dunagan show that the perceptron algorithm has probably polynomial smoothed complexity, with exponent $\alpha = 1$. Their proof follows from a demonstration that if the positive points can be separated from the negative points, then they can probably be separated by a large margin. It is known that the perceptron algorithm converges quickly in this case. Moreover, this margin is exactly what is maximized by Support Vector Machines.

PAC Learning

In a recent paper, as another application of smoothed analysis in machine learning, Kalai and Teng [21] proved that all decision trees are PAC-learnable from most product distributions. Probably approximately correct learning (PAC learning) is a framework in machine learning introduced by Valiant. In PAC-learning, the learner receives polynomial number of samples and constructs in polynomial a classifier which can predicate future sample data with a given probability of correctness.

3.3 Numerical Analysis

One of the focii of Numerical Analysis is the determiniation of how much precision is required by numerical methods. For example, consider the most fundamental problem in computational science— that of solving systems of linear equations. Because of the round-off errors in computation, it is crucial to know how many bits of precision a linear solver should maintain so that its solution is meaningful.

For example, Wilkinson [40] demonstrated a family of linear systems[3] of n variables and $\{0, -1, 1\}$ coefficients for which Gaussian elimination with partial pivoting — the algorithm implemented by Matlab — requires n-bits of precision.

Precision Requirements of Gaussian Elimination

However, fortunately, in practice one almost always obtains accurate answers using much less precision. In fact, high-precision solvers are rarely used or needed. For example, Matlab uses 64 bits.

Building on the smoothed analysis of condition numbers (to discussed below), Sankar, Spielman, and Teng [34, 33] proved that it is sufficient to use $O(\log^2(n/\sigma))$ bits of precision to run Gaussian elimination with partial pivoting when the matrices of the linear systems are subject to σ-Gaussian perturbations.

The Condition Number

The smoothed analysis of the condition number of a matrix is a key step toward understanding the numerical precision required in practice. For a square matrix $\mathbf{A}$, its condition number $\kappa(\mathbf{A})$ is given by $\kappa(\mathbf{A}) = \|\mathbf{A}\|_2 \left\|\mathbf{A}^{-1}\right\|_2$ where $\|\mathbf{A}\|_2 = \max_{\mathbf{x}} \|\mathbf{A}\mathbf{x}\|_2 / \|\mathbf{x}\|_2$. The condition number of $\mathbf{A}$ measures how much the solution to a system $\mathbf{A}\mathbf{x} = \mathbf{b}$ changes as one makes slight changes to $\mathbf{A}$ and $\mathbf{b}$: If one solves the linear system using fewer than $\log(\kappa(\mathbf{A}))$ bits of precision, then one is likely to obtain a result far from a solution.

The quantity $1/\left\|\mathbf{A}^{-1}\right\|_2 = \min_{\mathbf{x}} \|\mathbf{A}\mathbf{x}\|_2 / \|\mathbf{x}\|_2$ is known as the smallest singular value of $\mathbf{A}$. Sankar, Spielman, and Teng [34] proved the following statement: For any squared matrix $\bar{\mathbf{A}}$ in $\mathbb{R}^{n \times n}$ satisfying $\left\|\bar{\mathbf{A}}\right\|_2 \leq \sqrt{n}$, and for any $x > 1$,

$$\Pr_{\mathbf{A}}\left[\left\|\mathbf{A}^{-1}\right\|_2 \geq x\right] \leq 2.35 \frac{\sqrt{n}}{x\sigma},$$

where $\mathbf{A}$ is a σ-Gaussian perturbation of $\bar{\mathbf{A}}$. Consequently, together with an improved bound of Wschebor [41], one can show that

$$\Pr\left[\kappa(\mathbf{A}) \geq x\right] \leq O\left(\frac{n \log n}{x\sigma}\right).$$

See [9, 13] for smoothed analysis of the condition numbers of other problems.

3.4 Discrete Mathematics

For problems in discrete mathematics, it is more natural to use Boolean perturbations: Let $\bar{\mathbf{x}} = (\bar{x}_1, \ldots, \bar{x}_n) \in \{0,1\}^n$ or $\{-1,1\}^n$: the σ-Boolean perturbation of $\bar{\mathbf{x}}$ is a random string $\mathbf{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ or $\{-1,1\}^n$, where $x_i = \bar{x}_i$ with probability $1 - \sigma$ and $x_i \neq \bar{x}_i$ with probability σ. That is, each bit is flipped independently with probability σ.

Believing that σ-perturbations of Boolean matrices should behave like Gaussian perturbations of Real matrices, Spielman and Teng [35] made the following conjecture:

> For any n by n matrix $\bar{\mathbf{A}}$ of ± 1's. Let $\mathbf{A}$ be a σ-Boolean perturbation of $\bar{\mathbf{A}}$. Then
>
> $$\Pr_{\mathbf{A}}\left[\left\|\mathbf{A}^{-1}\right\|_2 \geq x\right] \leq O\left(\frac{\sqrt{n}}{x\sigma}\right)$$

[3] See the second line of the Matlab code at the end of Section 4 for an example.

In particular, let $\mathbf{A}$ be an n by n matrix of independently and uniformly chosen ± 1 entries. Then

$$\Pr_{\mathbf{A}}\left[\left\|\mathbf{A}^{-1}\right\|_2 \geq x\right] \leq \frac{\sqrt{n}}{x} + \alpha^n.$$

This conjectured is recently proved by Vu and Tao [39] and Rudelson and Vershynin [32].

In graph theory, σ-Boolean perturbations of a graph can be viewed as a smoothed extension of the classic Erdös-Rényi random graph model. The Erdös-Rényi model, denoted by $G(n,p)$, is a random graph in which every possible edge occurs independently with probability p.Let $\bar{G} = (V,E)$ be a graph over vertices $V = \{1,...,n\}$. Then, the σ-perturbation of $\bar{G}$, which we denote by $G_{\bar{G}}(n,\sigma)$, is a distribution of random graphs. Clearly for $p \in [0,1]$, $G(n,p) = G_{\emptyset}(n,p)$, i.e., the p-Boolean perturbation of the empty graph. One can define a smoothed extension of other random graph models. For example, for any m and $G = (V,E)$, Bohman, Frieze and Martin [8] define $G(\bar{G},m)$ to be the distribution of the random graphs $(V, E \cup T)$ where T is a set of m edges chosen uniformly at random from the complement of E, i.e., chosen from $\bar{E} = \{(i,j) \notin E\}$.

A popular subject of study in the traditional Erdös-Rényi model is the phenomenon of phase transition: for many properties such as being connected or being Hamiltonian, there is a critical p below which a graph is unlikely to have each property and above which it probably does have the property. Related phase transitions have also been found in the smoothed Erdös-Rényi models $G_{\bar{G}}(n,\sigma)$ [26, 17].

Smoothed analysis based on Boolean perturbations can be applied to other discrete problems. For example, Feige [16] used the following smoothed model for 3CNF formulas. First, an adversary picks an arbitrary formula with n variables and m clauses. Then, the formula is perturbed at random by flipping the polarity of each occurrence of each variable independently with probability σ. Feige gave a randomized polynomial time refutation algorithm for this problem.

3.5 Combinatorial Optimization

Beier and Vöcking [5] and Röglin and Vöcking [31] considered the smoothed complexity of integer linear programming. They studied programs of the form

$$\max \quad \mathbf{c}^T\mathbf{x} \qquad \text{subject to} \quad \mathbf{A}\mathbf{x} \leq \mathbf{b} \text{ and } \mathbf{x} \in \mathcal{D}^n, \qquad (5)$$

where $\mathbf{A}$ is an $m \times n$ Real matrix, $\mathbf{b} \in \mathbb{R}^m$, and $\mathcal{D} \subset \mathbb{Z}$.

Recall that ZPP denotes the class of decision problems solvable by a randomized algorithm that always returns the correct answer, and whose expected running time (on every input) is polynomial. Beier, Röglin and Vöcking [5, 31] proved the following statement: For any constant c, let Π be a class of integer linear programs of form (5) with $|D| = O(n^c)$. Then, Π has an algorithm of probably smoothed polynomial complexity if and only if $\Pi_u \in$ ZPP, where Π_u is the "unary" representation of Π. Consequently, the 0/1-knapsack problem, the constrained shortest path problem, the constrained minimum spanning tree problem, and the constrained minimum weighted matching problem can be solved in smoothed polynomial time in the sense according to Definition 3.

Remark: Usually by saying Π has a *pseudo-polynomial time* algorithm, one means $\Pi_u \in$ P. So $\Pi_u \in$ ZPP means that Π is solvable by a randomized *pseudo-polynomial time* algorithm. We say a problem Π is strongly NP-hard if Π_u is NP-hard. For example, 0/1-integer programming with a fixed number of constraints is in pseudo-polynomial time, while general 0/1-integer programming is strongly NP-hard.

Smoothed analysis has been applied to several other optimization problems such as local search and TSP [15], scheduling [4], sorting [3], motion planing [11], superstring approximation [28], multi-objective optimization [31], embedding [1], and multidimensional packing [22].

4. DISCUSSION

4.1 Other Performance Measures

Although we normally evaluate the performance of an algorithm by its running time, other performance parameters are often important. These performance parameters include the amount of space required, the number of bits of precision required to achieve a given output accuracy, the number of cache misses, the error probability of a decision algorithm, the number of random bits needed in a randomized algorithm, the number of calls to a particular subroutine, and the number of examples needed in a learning algorithm. The quality of an approximation algorithm could be its approximation ratio; the quality of an online algorithm could be its competitive ratio; and the parameter of a game could be its price of anarchy or the rate of convergence of its best-response dynamics. We anticipate future results on the smoothed analysis of these performance measures.

4.2 Pre-cursors to Smoothed Complexity

Several previous probabilistic models have also combined features of worst-case and average-case analyses.

Haimovich [19] considered the following probabilistic analysis: Given a linear program $\mathcal{L} = (\mathbf{A}, \mathbf{b}, \mathbf{c})$ as in Eqn. (1), they defined the *expected complexity* of $\mathcal{L}$ to be the expected complexity of the simplex algorithm when the inequality sign of each constraint is uniformly flipped. They proved that the expected complexity of the worst possible $\mathcal{L}$ is polynomial.

Blum and Spencer [7] studied the design of polynomial-time algorithms for the semi-random model, which combines the features of the semi-random source with the random graph model that has a "planted solution". This model can be illustrated with the k-Coloring Problem: An adversary plants a solution by partitioning the set V of n vertices into k subsets $V_1, \ldots, V_k$. Let

$$F = \{(u,v) | u \text{ and } v \text{ are in different subsets}\}$$

be the set of potential inter-subset edges. A graph is then constructed by the following semi-random process that perturbs the decisions of the adversary: In a sequential order, the adversary decides whether to include each edge in F in the graph, and then a semi-random process reverses the decision with probability σ. Note that every graph generated by this semi-random process has the planted coloring: $c(v) = i$ for all $v \in V_i$, as both the adversary and the semi-random process preserve this solution by only considering edges from F.

As with the smoothed model, one can work with the semi-random model by varying σ from 0 to 1 to interpolate between worst-case and average-case complexity for k-coloring. In fact, the semi-random model is related with the following perturbation model that *partially preserves* a particular solution: Let $\bar{G} = (V, \bar{E})$ be a k-colorable graph. Let $c : V \rightarrow$

$\{1, ..., k\}$ be a k-coloring of $\bar{G}$ and let $V_i = \{v \mid c(v) = i\}$. The model then returns a graph $G = (V, E)$ that is a σ-Boolean perturbation of $\bar{G}$ subject to c also being a valid k-coloring of G. This perturbation model is equivalent to the semi-random model with an *oblivious* adversary, who simply chooses a set $\bar{E} \subseteq F$, and sends the decisions that only include edges in $\bar{E}$ (and hence exclude edges in $F - \bar{E}$) through the semi-random process.

4.3 Algorithm Design and Analysis for Special Families of Inputs

Probabilistic approaches are not the only means of characterize practical inputs. Much work has been spent on designing and analyzing inputs that satisfy certain deterministic but practical input conditions. We mention a few examples that excite us.

In parallel scientific computing, one may often assume that the input graph is a well-shaped finite element mesh. In VLSI layout, one often only considers graphs that are planar or nearly planar. In geometric modeling, one may assume that there is an upper bound on the ratio among the distances between points. In web analysis, one may assume that the input graph satisfies some powerlaw degree distribution or some small-world properties. When analyzing hash functions, one may assume that the data being hashed has some non-negligible entropy [30].

4.4 Limits of Smoothed Analysis

The goal of smoothed analysis is to explain why some algorithms have much better performance in practice than predicated by the traditional worst-case analysis. However, for many problems, there may be better explanations.

For example, the worst-case complexity and the smoothed complexity of the problem of computing a market equilibrium are essentially the same [20]. So far, no polynomial-time pricing algorithm is known for general markets. On the other hand, pricing seems to be a practically solvable problem, as Kamal Jain put it "If a Turing machine can't compute then an economic system can't compute either."

A key step to understanding the behaviors of algorithms in practice is the construction of analyzable models that are able to capture some essential aspects of practical input instances. For practical inputs, there may often be multiple parameters that govern the process of their formation.

One way to strengthen the smoothed analysis framework is to improve the model of the formation of input instances. For example, if the input instances to an algorithm A come from the output of another algorithm B, then algorithm B, together with a model of B's input instances, provide a description of A's inputs. For example, in finite-element calculations, the inputs to the linear solver A are stiffness matrices which are produced by a meshing algorithm B. The meshing algorithm B, which could be a randomized algorithm, generates a stiffness matrix from a geometric domain Ω and a partial differential equation F. So, the distribution of the stiffness matrices input to algorithm A is determined by the distribution $\mathcal{D}$ of the geometric domains Ω and the set F of partial differential equations, and the randomness in algorithm B. If, for example, $\bar{\Omega}$ is the design of an advanced rocket from a set $\mathcal{R}$ of "blueprints" and F is from a set $\mathcal{F}$ of PDEs describing physical parameters such as pressure, speed, and temperature, and Ω is generated by a perturbation model $\mathcal{P}$ of the blueprints, then one may further measure the performance of A by the smoothed value of the quantity above:

$$\max_{F \in \mathcal{F}, \bar{\Omega} \in \mathcal{R}} \mathop{\mathbf{E}}_{\Omega \leftarrow \mathcal{P}(\bar{\Omega})} \left[\mathop{\mathbf{E}}_{X \leftarrow B(\Omega, F)} [\, Q(A, X) \,] \right].$$

In the above formulae, $\Omega \leftarrow \mathcal{P}(\bar{\Omega})$ denotes that Ω is obtained from the perturbation of $\bar{\Omega}$ and $X \leftarrow B(\Omega, F)$ denotes that X is the output of the randomized algorithm B.

4.5 Algorithm Design based on Perturbations and Smoothed Analysis

Finally, we hope insights gained from smoothed analysis will lead to new ideas in algorithm design. On a theoretical front, Kelner and Spielman [24] exploited ideas from the smoothed analysis of the simplex method to design a (weakly) polynomial-time simplex method that functions by systematically perturbing its input program. On a more practical level, we suggest that it might be possible to solve some problems more efficiently by perturbing their inputs. For example, some algorithms in computational geometry implement variable-precision arithmetic to correctly handle exceptions that arise from geometric degeneracy [29]. However, degeneracies and near-degeneracies occur with exceedingly small probability under perturbations of inputs. To prevent perturbations from changing answers, one could employ quad-precision arithmetic, placing the perturbations into the least-significant half of the digits.

Our smoothed analysis of Gaussian Elimination suggests a more stable solver for linear systems: When given a linear system $\mathbf{Ax} = \mathbf{b}$, we first use the standard Gaussian Elimination with partial pivoting algorithm to solve $\mathbf{Ax} = \mathbf{b}$. Suppose $\mathbf{x}^*$ is the solution computed. If $\|\mathbf{b} - \mathbf{Ax}^*\|$ is small enough, then we simply return $\mathbf{x}^*$. Otherwise, we can determine a parameter ϵ and generate a new linear system $(\mathbf{A} + \epsilon\mathbf{G})\mathbf{y} = \mathbf{b}$, where $\mathbf{G}$ is a Gaussian matrix with mean $\mathbf{0}$ and variance 1. Instead of solving $\mathbf{Ax} = \mathbf{b}$, we solve a perturbed linear system $(\mathbf{A} + \epsilon\mathbf{G})\mathbf{y} = \mathbf{b}$. It follows from standard analysis that if ϵ is sufficiently smaller than $\kappa(\mathbf{A})$, then the solution to the perturbed linear system is a good approximation to the original one. One could use practical experience or binary search to set ϵ.

The new algorithm has the property that its success depends only on the machine precision and the condition number of $\mathbf{A}$, while the original algorithm may fail due to large growth factors. For example, the following is a segment of Matlab code that first solves a linear system whose matrix is the 70×70 matrix Wilkinson designed to trip up partial pivoting, using the Matlab linear solver. We then perturb the system, and apply the Matlab solver again.

```
>> % Using the Matlab Solver
>> n = 70; A = 2*eye(n)-tril(ones(n)); A(:,n)=1;
>> b = randn(70,1); x = A\b;
>> norm(A*x-b)
>> 2.762797463910437e+004
>> % FAILED because of large growth factor
>> %Using the new solver
>> Ap = A + randn(n)/10^9; y = Ap\b;
>> norm(Ap*y-b)
>> 6.343500222435404e-015
>> norm(A*y-b)
>> 4.434147778553908e-008
```

Note that while the Matlab linear solver fails to find a good solution to the linear system, our new perturbation-based algorithm finds a good solution. While there are standard algorithms for solving linear equations that do not have the poor worst-case performance of partial pivoting, they are rarely used as they are less efficient.

For more examples of algorithm design inspired by smoothed analysis and perturbation theory, see [37].

5. ACKNOWLEDGMENTS

We would like to thank Alan Edelman for suggesting the name "Smoothed Analysis" and thank Heiko Röglin and Don Knuth for helpful comments on this writing.

6. REFERENCES

[1] A. Andoni and R. Krauthgamer. The smoothed complexity of edit distance. In *Proceedings of ICALP*, volume 5125 of *Lecture Notes in Computer Science*, pages 357–369. Springer, 2008.

[2] D. Arthur and S. Vassilvitskii. How slow is the k-mean method? In *SOCG' 06, the 22nd Annual ACM Symposium on Computational Geometry*, pages 144–153, 2006.

[3] C. Banderier, R. Beier, and K. Mehlhorn. Smoothed analysis of three combinatorial problems. In *the 28th International Symposium on Mathematical Foundations of Computer Science*, pages 198–207, 2003.

[4] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, G. Schäfer, , and T. Vredeveld. Average case and smoothed competitive analysis of the multi-level feedback algorithm. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, page 462, 2003.

[5] R. Beier and B. Vöcking. Typical properties of winners and losers in discrete optimization. In *STOC' 04: the 36th annual ACM symposium on Theory of computing*, pages 343–352, 2004.

[6] A. Blum and J. Dunagan. Smoothed analysis of the perceptron algorithm for linear programming. In *SODA '02*, pages 905–914, 2002.

[7] A. Blum and J. Spencer. Coloring random and semi-random k-colorable graphs. *J. Algorithms*, 19(2):204–234, 1995.

[8] T. Bohman, A. Frieze, and R. Martin. How many random edges make a dense graph hamiltonian? *Random Struct. Algorithms*, 22(1):33–42, 2003.

[9] P. Bürgissera, F. Cucker, and M. Lotz. Smoothed analysis of complex conic condition numbers. *J. de Mathématiques Pures et Appliqués*, 86(4):293–309, 2006.

[10] B. Manthey D. Arthur and H. Röglin. k-means has polynomial smoothed complexity. In *to appear*, 2009.

[11] V. Damerow, F. Meyer auf der Heide, H. Räcke, Christian Scheideler, and C. Sohler. Smoothed motion complexity. In *Proc. 11th Annual European Symposium on Algorithms*, pages 161–171, 2003.

[12] G. B. Dantzig. Maximization of linear function of variables subject to linear inequalities. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347. 1951.

[13] J. Dunagan, D. A. Spielman, and S.-H. Teng. Smoothed analysis of Renegar's condition number for linear programming. Available at `http://arxiv.org/abs/cs/0302011v2`, 2003.

[14] A. Edelman. Eigenvalue roulette and random test matrices. In Marc S. Moonen, Gene H. Golub, and Bart L. R. De Moor, editors, *Linear Algebra for Large Scale and Real-Time Applications*, NATO ASI Series, pages 365–368. 1992.

[15] M. Englert, H. Röglin, and B. Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the TSP: extended abstract. In *SODA' 07: the 18th annual ACM-SIAM symposium on Discrete algorithms*, pages 1295–1304, 2007.

[16] U. Feige. Refuting smoothed 3CNF formulas. In *the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 407–417, 2007.

[17] A. Flaxman and A. M. Frieze. The diameter of randomly perturbed digraphs and some applications.. In *APPROX-RANDOM*, pages 345–356, 2004.

[18] S. Gass and T. Saaty. The computational algorithm for the parametric objective function. *Naval Research Logistics Quarterly*, 2:39–45, 1955.

[19] M. Haimovich. The simplex algorithm is very good ! : On the expected number of pivot steps and related properties of random linear programs. Technical report, Columbia University, April 1983.

[20] L.-S. Huang and S.-H. Teng. On the approximation and smoothed complexity of Leontief market equilibria. In *Frontiers of Algorithms Workshop*, pages 96–107, 2007.

[21] A. T. Kalai and S.-H. Teng. Decison trees are pac-learnable from most product distributions: A smoothed analysis. MSR-NE, submitted, 2008.

[22] D. Karger and K. Onak. Polynomial approximation schemes for smoothed and random instances of multidimensional packing problems. In *SODA'07: the 18th annual ACM-SIAM symposium on Discrete algorithms*, pages 1207–1216, 2007.

[23] J. A. Kelner and E. Nikolova. On the hardness and smoothed complexity of quasi-concave minimization. In *the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 472–482, 2007.

[24] J. A. Kelner and D. A. Spielman. A randomized polynomial-time simplex algorithm for linear programming. In *the 38th annual ACM symposium on Theory of computing*, pages 51–60, 2006.

[25] V. Klee and G. J. Minty. How good is the simplex algorithm? In Shisha, O., editor, *Inequalities – III*, pages 159–175. Academic Press, 1972.

[26] M. Krivelevich, B. Sudakov, and P. Tetali. On smoothed analysis in dense graphs and formulas. *Random Structures and Algorithms*, 29:180–193, 2005.

[27] S. Lloyd. Least squares quantization in pcm. *IEEE Trans. on Information Theory*, 28(2):129–136, 1982.

[28] B. Ma. Why greed works for shortest common superstring problem. In *Combinatorial Pattern Matching, LNCS, Springer*.

[29] K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, New York, 1999.

[30] M. Mitzenmacher and S. Vadhan. Why simple hash functions work: exploiting the entropy in a data stream. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 746–755, 2008.

[31] H. Röglin and B. Vöcking. Smoothed analysis of integer programming. In *Michael Junger and Volker Kaibel, editors, Proc. of the 11th Int.Conf. on Integer Programming and Combinatorial Optimization, volume 3509 of Lecture Notes in Computer Science, Springer*, pages 276 – 290, 2005.

[32] M. Rudelson and R. Vershynin. The littlewood-offord problem and invertibility of random matrices. *Advances in Mathematics*, 218:600–633, June 2008.

[33] A. Sankar. Smoothed analysis of Gaussian elimination. Ph.D. Thesis, MIT, 2004.

[34] A. Sankar, D. A. Spielman, and S.-H. Teng. Smoothed analysis of the condition numbers and growth factors of matrices. *SIAM Journal on Matrix Analysis and Applications*, 28(2):446–476, 2006.

[35] D. A. Spielman and S.-H. Teng. Smoothed analysis of algorithms. In *Proceedings of the International Congress of Mathematicians*, pages 597–606, 2002.

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

[37] S.-H. Teng. Algorithm design and analysis with perburbations. In *Fourth International Congress of Chinese Mathematicans*, 2007.

[38] R. Vershynin. Beyond Hirsch conjecture: Walks on random polytopes and smoothed complexity of the simplex method. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 133–142, 2006.

[39] V. H. Vu and T. Tao. The condition number of a randomly perturbed matrix. In *STOC '07: the 39th annual ACM symposium on Theory of computing*, pages 248–255, 2007.

[40] J. H. Wilkinson. Error analysis of direct methods of matrix inversion. *J. ACM*, 8:261–330, 1961.

[41] M. Wschebor. Smoothed analysis of $\kappa(\mathbf{a})$. *J. of Complexity*, 20(1):97–107, February 2004.