Beyond Smoothed Analysis: Analyzing the Simplex Method By-the-Book

Eleon Bach
Technical University Munich
Germany

Alexander E. Black
Bowdoin College
USA

Sophie Huiberts
CNRS
LIMOS
University Clermont Auvergne
France

Sean Kafer
Illinois State University
USA

Abstract

Narrowing the gap between theory and practice is a longstanding goal of the algorithm analysis community. To further progress our understanding of how algorithms work in practice, we propose a new algorithm analysis framework that we call *by-the-book analysis*. In contrast to earlier frameworks, by-the-book analysis not only models an algorithm's input data, but also the algorithm itself. Results from by-the-book analysis are meant to correspond well with established knowledge of an algorithm's practical behavior, as they are meant to be grounded in observations from implementations, input modeling best practices, and measurements on practical benchmark instances. We apply our framework to the simplex method, an algorithm which is beloved for its excellent performance in practice and notorious for its high running time under worst-case analysis. The simplex method similarly showcased the previous state of the art framework *smoothed analysis* (Spielman and Teng, STOC'01). We explain how our framework overcomes several weaknesses of smoothed analysis and we prove that under input scaling assumptions, feasibility tolerances and other design principles used by simplex method implementations, the simplex method indeed attains a polynomial running time. Our results provide analytical justification for these features which are common to all high-quality simplex method implementations.

CCS Concepts

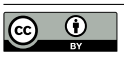
• **Theory of computation** → **Mathematical optimization**; • **Mathematics of computing** → **Solvers**.

Keywords

By-The-Book Analysis, Simplex Method, Smoothed Analysis

ACM Reference Format:
Eleon Bach, Alexander E. Black, Sophie Huiberts, and Sean Kafer. 2026. Beyond Smoothed Analysis: Analyzing the Simplex Method By-the-Book. In *Proceedings of the 58th Annual ACM Symposium on Theory of Computing (STOC '26), June 22–26, 2026, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3798129.3800742

Supported by ANR JCJC grant ANR-24-CE48-2762.



STOC '26, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-2536-4/2026/06
https://doi.org/10.1145/3798129.3800742

1 Introduction

There is a longstanding need for frameworks explaining the performance of algorithms whose users—whether they are engineers, operations researchers or mathematicians—know that they perform very well for their applications, but for which traditional worst-case analysis is too pessimistic. The simplex method for linear programming problems is an example of such an algorithm: it was a driving force in the development of electronic computers [67] as it was observed to perform efficiently; requiring a linear number of pivot steps in practice [5, 26, 35, 66, 75, 81]. A long line of research attempts to explain this behavior theoretically by a suitable algorithm analysis framework. The project of explaining the performance of the simplex method is not only interesting from a mathematical point of view, but also for practical users who are constantly facing new areas of application for linear programs which they might fear to lead to a substantially worse performance.

The simplex algorithm serves well as a showcase for the developments in the algorithm analysis community. Following the traditional line of algorithm analysis research, in 1972, the first exponential worst-case analysis lower bounds were shown for the number of pivot steps the simplex method may take with certain pivot rules, and many analogous results followed [3, 6, 11–13, 30, 31, 38, 39, 45, 46, 50, 56, 57, 61, 68, 73]. This discovery opened up the search for algorithm analysis frameworks which could more properly capture the behavior of the simplex algorithm and algorithms which showed a similar behavior.

For algorithms like these, average-case analysis is the theoretician's first attempt to provide a better understanding of the running time. In the case of the simplex method, this framework of analysis saw much activity during the 1970s and 1980s, and as a result, we know tight polynomial upper and lower bounds in the average case setting [1, 2, 16–19, 19, 20, 48, 69, 85]. However, linear programs seen in the average case setting greatly differ on a structural level from linear programs seen in practice. As such, average case analysis offered an incomplete explanation. Despite the development of algorithms for linear programming which run in polynomial time under a worst-case analysis, the simplex method is still an essential part of all linear programming software. As such, there remains a need to understand its performance in practice.

In their seminal work [82], Spielman and Teng proposed the smoothed analysis framework as an explanation as to "why the simplex method usually takes polynomial time." Their framework was subsequently applied to many more algorithms (see [78] for an overview). In Section 1.1, we review smoothed analysis of the simplex method. In Section 1.2, we discuss limitations of the smoothed analysis framework and argue why smoothed analysis is an incomplete explanation of the simplex method's efficient practical performance. In Section 1.3, we describe our new algorithm analysis framework and how our framework is overcoming these limitations.

1.1 Smoothed Analysis and the Simplex Method

We first recall the basic principles of the simplex method. The simplex method is best thought of as a class of algorithms. A simplex method first determines a basic feasible solution using a *Phase I* procedure and then iteratively moves to new basic feasible solutions until an optimal solution has been found. The individual moves are called *pivot steps*, and the number of pivot steps is a proxy for the running time. The choice for which basic feasible solution to move to is governed by a *pivot rule*. Some popular examples of pivot rules are the most negative reduced cost rule [27], the steepest edge rule and its approximations [37, 44, 51], and the shadow vertex rule, also known as the parametric rule [18, 40]. It is this last pivot rule that is the foundational tool for most probabilistic analyses, including all results in smoothed analysis.

In the smoothed analysis of linear programming, we assume that an adversary specifies an LP

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad & \bar{A}\mathbf{x} \leq \bar{\mathbf{b}}, \end{aligned}$$

and that subsequently a perturbation $\hat{A} \in \mathbb{R}^{n \times d}, \hat{\mathbf{b}} \in \mathbb{R}^n$ is sampled. We assume that the entries of $(\hat{A}, \hat{\mathbf{b}})$ are independently sampled from the Gaussian distribution with mean 0 and standard deviation $\sigma > 0$, and we assume that the rows of the extended matrix $(\bar{A}, \bar{\mathbf{b}})$ each have Euclidean norm at most 1. Under these assumptions, smoothed analysis aims to develop simplex methods for solving

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad & (\bar{A} + \hat{A})\mathbf{x} \leq \bar{\mathbf{b}} + \hat{\mathbf{b}}, \end{aligned}$$

for which the expected running time can be bounded by a polynomial function in σ^{-1}, d and n. After a line of work [24, 29, 55, 79, 82, 87], Bach and Huiberts [7] found a simplex method that requires no more than

$$O(\sigma^{-1/2} d^{11/4} \log(n)^{7/4})$$

pivot steps. They also described a lower bound which applies to all pivot rules, stating that

$$\Omega(\sigma^{-1/2} d^{1/2} \ln(1/\sigma)^{-1/4})$$

pivot steps are necessary when $n = (4/\sigma)^d$.

Despite these upper and lower bounds agreeing on the exponent of $\sigma^{-1/2}$, smoothed analysis can hardly be said to be a complete explanation of the simplex method's running time. The LPs in smoothed analysis are unlike LPs in practice in a number of important ways, of which we describe three in the next section. These three differences undermine the three leading interpretations given for the smoothed analysis of the simplex method.

1.2 Limitations of Smoothed Analysis

First, most linear programs seen in practice are very *sparse*; less than 0.1% of their entries are non-zero (see, e.g., [43, 49, 67]). The same holds true for known worst-case inputs. This stands in stark contrast with smoothed analysis, where 100% of the entries are non-zero since they are randomly perturbed according to a continuous probability distribution. As such, in any smoothed probability model, the set of sparse linear programs has measure 0. Since both practical linear programs and theoretical worst-case inputs are sparse, it remains unclear whether smoothed analysis provides a viable synthetic model for describing the "brittleness" of worst-case inputs and whether smoothed analysis indeed helps to explain why the simplex method is usually fast on inputs observed in practice. In their foundational work introducing smoothed analysis [82], Spielman and Teng highlighted the lack of sparsity as a weakness of their model and stated finding some analysis accounting for sparsity as a future direction.

Second, when LP constraint data is noisy or is rounded to low precision, this affects LP solver behavior negatively. For example, a nominally singular set of constraints might become a basis (potentially an ill-conditioned one) due to improper rounding, and this can lead to performance degradation. For this reason it is recommended to specify all constraints with maximum numerical precision (see, e.g., Gurobi's Guidelines for Numerical Issues [47]). In contrast, smoothed analysis produces stronger performance guarantees if *more* noise is added to the constraint matrix, thereby seemingly predicting that lower numerical accuracy results in performance improvement. Due to this disagreement between practical advice and theoretical prediction, we argue that smoothed analysis is not appropriate for accurately modeling the effects of round-off error or floating-point imprecision.

Third, a small perturbation to the constraint matrix can lead to a large change in the feasible region. This fact has long been observed on practical LPs in the robust optimization literature (see, e.g., [9]). We must assume that even these LPs which are sensitive to small perturbations were formulated appropriately by the modeler. We interpret this to imply that the constraint data is highly accurate, in the sense that the constraint matrix entries cannot be independently perturbed by as little as $\frac{1}{10000}$. In this manner, we argue that smoothed analysis is not an appropriate model for measurement error or other forms of limited precision input.

The variable nature of σ leaves smoothed analysis agnostic as to what it models. Although different interpretations have been given in the literature, they largely fall within one of three categories: smoothed analysis models the "brittleness" or "isolation" of worst-case instances, smoothed analysis models the effects of numerical imprecision in floating-point arithmetic, or smoothed analysis models the effects of measurement error or other arbitrary influences. As we have seen above, each of these three interpretations has notable weaknesses.

1.3 Introducing By-The-Book Analysis

In this paper we propose a new algorithm analysis framework that we call by-the-book analysis. Unlike smoothed analysis, by-the-book analysis is explicitly empirical by being intentional about

what effect it models. As part of this, we open up the possibility of modeling not only the input data but also aspects of an algorithm's implementation. It is well-known that theoretical descriptions of algorithms differ from their software implementations. These differences are often thought to be incidental and to be theoretically uninteresting, even though they have repeatedly proven to be beneficial in practice. We posit that these differences, if they are found in state of the art implementations, can be essential to a full theoretical understanding. In doing so, we answer the call of [52] for an algorithmic theory capturing "the all-important tricks that are engineered into commercial codes".

With by-the-book analysis, we strive to prove performance guarantees while using assumptions that are maximally grounded in computational experience. For this reason, our research process has occurred in three phases as depicted in Figure 1. We will first describe what we propose as the general framework of by-the-book analysis, and then we will exemplify this framework by outlining the specifics of our own by-the-book analysis of the simplex method.

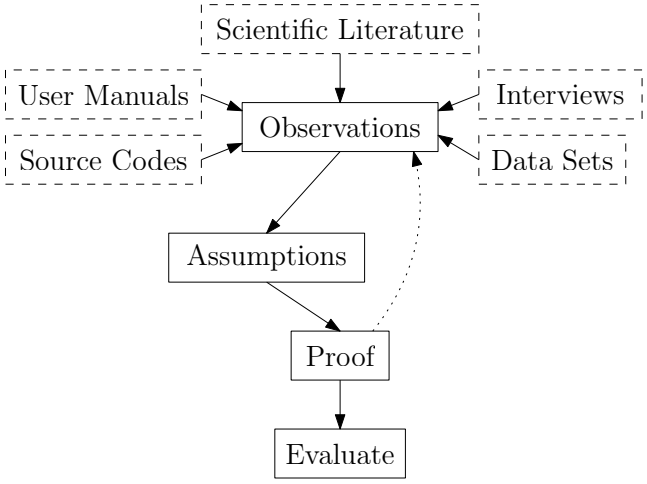


Figure 1: Schematic depiction of by-the-book analysis

In general, by-the-book analysis proceeds in three phases. In the first phase, one establishes an understanding of true and relevant features of real-world implementations of an algorithm and features of real-world inputs. This process can involve reading of the scientific literature, open-source code, user manuals, etc. in order to establish how the algorithm is implemented in practice, how users of the algorithm are instructed to formulate their inputs, and what features are common among real problem instances.

The second phase is a more subjective process which converts parts of this understanding to a mathematical model of the algorithm on which some form of traditional theoretical analysis can be performed. Based on the observations made in the first phase, one chooses an algorithm description and a collection of assumptions satisfied by the algorithm and/or its inputs. The chosen algorithm description may deviate from traditional theoretical descriptions of the algorithm if that deviation is warranted by observations. One might assume, for example, that certain relevant parameters of the input are bounded. Although one must inevitably make assumptions that make the upcoming mathematical analysis tractable, one should, to the greatest extent possible, choose a mathematical model and mathematical assumptions that comport with the observations made in the first phase. This is what gives a by-the-book analysis its empirical basis.

The third phase is a traditional mathematical analysis of the chosen model under the chosen assumptions. In this phase, by-the-book analysis is agnostic to the mathematical techniques used provided that any further assumptions inherent to those techniques are likewise grounded (to the greatest extent possible) in the observations of the first phase. For example, in this phase the mathematical analysis could resemble a worst-case, average-case, smoothed analysis, etc. Ultimately, what makes something a by-the-book analysis is not the choice of mathematical tools used, but that the model, assumptions, and tools are grounded in observation.

Note that the order of the phases is important. In particular, the determination of mathematical assumptions occurs *before* the production of theoretical running time bounds. It is contrary to by-the-book analysis to produce a theoretical bound parameterizing by whichever parameters turn out to be mathematically fruitful and to only then search for explanations as to why these parameters may be bounded in practice. By *basing* the theoretical approach on observations, we argue that this methodology is better positioned to produce accurate explanations. This is the ultimate objective of performing a by-the-book analysis. In this spirit, we emphasize again that what makes something a by-the-book analysis is not which type of mathematical tools or proof techniques are used, but whether or not the research methodology follows the general framework outlined above. To call something a by-the-book analysis (or not) is not a mathematical distinction, but a methodological one; it is not a question of *what* mathematical assumptions are made, but of *how* they are produced in the first place.

We will now briefly outline how we applied the above procedure to the simplex method in the present analysis. In the initial phase we studied the scientific literature, source-code, and user manuals of simplex implementations, and we interviewed developers working on multiple different LP software packages. A primary role in this phase was played by the titular textbooks [67, 74] which describe how to implement a simplex method. After observing the prominent role that numerical tolerances are given in both books (on which we will elaborate later), we verified their importance by way of inspecting both the source code of open-source LP solvers and the user manuals of closed-source LP solvers. Because the simplex method is a popular and well-documented algorithm, we were able to draw on a diverse array of sources and identify points of broad agreement. The outcomes of this phase are described in detail in Section 2.

In the second phase, we balanced verisimilitude with our observations against the tractability of the upcoming mathematical analysis. We aimed for a traditional, theoretical performance guarantee which, when all parameters are filled in with plausible values, gives the strongest possible conclusion in terms of how well this bound is reflecting an algorithm's behavior in practice (subject, of course, to adherence to our observations in the first phase). For

this analysis, we chose a model of the simplex method that perturbs the right-hand side vector with random exponentially distributed noise (whose magnitude is grounded in our observations) and which solves the input LP up to optimality and feasibility tolerances (whose magnitudes are grounded in our observations) using the shadow vertex pivot rule with an auxiliary objective function chosen uniformly at random. To maintain mathematical tractability, we assumed the algorithm to be implemented using exact real valued arithmetic and Dantzig's ratio test, as well as the shadow vertex pivot rule. Relevant instance-specific parameters are assumed to be bounded in magnitude (again, grounded in observation). Detailed descriptions can be found in the full version of this paper.

In the third and final phase, we used our mathematical assumptions on the data and the algorithm to prove running time bounds using a mathematical analysis similar to those found in smoothed analysis, using the randomness in our perturbations and our auxiliary objective function. During this phase of the work, we realized that using the *mean width* (a type of mixed volume of a convex body, see, e.g., [80]) of the feasible set would refine our argument. This became our mathematical assumption standing in for the scaling of the LP, replacing an earlier assumption of bounded Euclidean diameter of the feasible set. We believe refining and revising the set of assumptions can be a natural part of the by-the-book process, especially when the resulting assumptions are easier to measure or observe. For that reason, we went back to the first phase: we performed measurements on the MIPLIB 2017 data set [43] to find typical values for the mean width. The mean width is easily approximated up to high accuracy due to Milman's concentration of measure phenomenon (see, e.g., [8]).

As is evident from the specificity of some choices detailed above, different by-the-book analyses for the same algorithm are possible. They can vary in how the algorithm and the data are modeled, i.e., which assumptions and parameters are chosen, and they can vary in the types of mathematical techniques used to prove running time bounds. We can distinguish at least two paths for improvements on a particular by-the-book analysis. The first occurs in the third phase and is one that we are used to in theoretical computer science: for the same set of mathematical assumptions, stronger mathematical analysis techniques can be developed, leading to stronger theorems. The second is to formulate *different* mathematical assumptions to model the same or different observations performed in the first phase.

One is unlikely to produce a singular, holistic explanation for the performance of an algorithm, parameterizing by all possible factors and perfectly distilling the impact of these parameters into a bound. Theoretical analysis requires simplified models on which theoretical analyses can be performed. The goal is not to produce a perfect model of reality, but instead to produce a mathematical model which is better informed by, and more reflective of, what we can observe. As such, a by-the-book analysis can be improved in the first phase by doing more or better observations, and second phases by better modelling. Some axes on which such improvement can be evaluated include, but are not limited to, evaluability and verisimilitude. It is preferable to express bounds in terms of parameters which can be effectively observed. It is also preferable to use mathematical assumptions which more closely resemble computational practice, such as to have a theoretical description of an algorithm which comports with the way in which that algorithm is implemented in code.

To exemplify this last point, we wield it against our present analysis in Section 5. We evaluate how well the chosen model, assumptions, and mathematical techniques comport with our observations, and evaluate how well our research process adhered to the framework of by-the-book analysis. For the sake of good scientific practice, such a final evaluation phase is crucial. We recommend the evaluation process to be not only conducted as an isolated final last step. Instead, we recommend to evaluate the observations and decisions made as a steady part of the process throughout the first three phases.

1.4 Observations of Practical LP

Having described the framework of by-the-book analysis in general, we can now present a by-the-book analysis of the simplex method. We begin here with an overview of our findings about the properties held by LP in practice. As described in Section 1.3, these findings will inform our mathematical assumptions. As noted already, two main contributing factors to our findings are the software implementations of the simplex method and the observed characteristics of (well-formulated) inputs. A given problem can be modeled in various ways, and a given LP has a wide array of equivalent formulations. When LPs are solved in practice, they are typically modeled by domain experts who are constructing their formulations according to best practice recommendations explicitly detailed in software user manuals.

We review a number of software packages and describe our observations. For the open-source solvers Glop and HiGHS, we report our observations from reading their respective source codes. For the closed-source solvers Gurobi and MOSEK, we read their user manuals in detail and spoke extensively with developers. We ran an experiment on the MIPLIB 2017 benchmark set to support our assumption that feasible sets of practical LPs have small mean width. The features of solvers and data described here are mostly limited to those which are directly relevant to our models of scaling, tolerances, and bound/cost perturbations. We first describe the general principles of what we observe, and in Section 2 we describe what observations we have with regard to specific software packages. There are many other features we do not remark upon, although we believe that some can be potentially of interest to theoretical computer scientists and may lead to improved by-the-book analyses of the simplex method.

Condition number. In an IEEE 754 double precision floating point number, there is 1 bit for the sign, 11 bits for the exponent, and 53 bits for the significand (of which 52 are explicitly stored). This means that any such number can have a relative precision of at most $2^{-53} \approx 10^{-16}$. Any arithmetic performed with numbers whose relative magnitudes differ by many orders of magnitude will yield imprecise results. This effect is particularly visible when solving a system of linear equations. Here, the backward error up to which it can be solved increases when the linear system has a high *condition number*. The condition number κ of a matrix A is defined as $\kappa := \|A\|\|A^{-1}\|$. The relation to the backward error is the following. When solving the linear system $Ax = b$, a condition number $\kappa = 10^k$ indicates that one may loose up to k digits of accuracy in x from

the accuracy in b. Typically, in linear programming software, the condition number of linear systems is assumed to be no greater than 10^{10}, see Section 2.

Tolerances. With 53 bits of accuracy and a condition number of order 10^{10}, solving a linear system $A_B\mathbf{x} = \mathbf{b}_B$ can be done up to backward error around 10^{-6}. That is, the software is able to find $\hat{\mathbf{x}}$ such that $\|A_B^{-1}\mathbf{b}_B - \hat{\mathbf{x}}\| \leq 10^{-6} \cdot \|A_B^{-1}\mathbf{b}_B\|$. Because of this inevitable inaccuracy, every LP solver based on floating-point arithmetic includes a number of different tolerances and thresholds. Here we discuss only the user-facing *absolute feasibility tolerances*. There are two of these, a primal feasibility tolerance and a dual feasibility tolerance, also called the *optimality tolerance*. These two numbers, both commonly of order 10^{-6}, indicate how large the violations of primal or dual constraints are allowed to be for the solver to still report a solution as being 'feasible and optimal'. In contrast to the relative error described in the preceding paragraph, solver tolerances are absolute.

In this work we model the primal feasibility and complementary slackness conditions as follows. For more information on the deduction of this statement of complementary slackness, see Section 3. There, we solve linear programs of the form

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{0} \leq \mathbf{x} \leq \mathbf{u}, \end{aligned}$$

where we assume that every row of A has norm 1. The solver may return (as optimal and feasible up to tolerances) any primal-dual solution pair $\mathbf{x}^* \in \mathbb{R}^d, \mathbf{y}^* \in \mathbb{R}^n_{\geq 0}$ which satisfies primal feasibility up to primal feasibility tolerance feasTol > 0 and complementary slackness up to dual feasibility tolerance optTol > 0. Formally, they satisfy

$$\begin{aligned} A\mathbf{x}^* &\leq \mathbf{b} + \text{feasTol} \cdot \mathbf{1}_n \\ -\text{feasTol} \cdot \mathbf{1}_d \leq \mathbf{x}^* &\leq \mathbf{u} + \text{feasTol} \cdot \mathbf{1}_d, \end{aligned} \tag{1}$$

and

$$\begin{aligned} &\text{if } \mathbf{y}^*_i > 0 \text{ then } (A\mathbf{x}^*)_i \geq \mathbf{b}_i \text{ for all } i \in [n], \\ &\text{if } \mathbf{c}_j > (A^\top \mathbf{y}^*)_j + \text{optTol then } \mathbf{x}^*_j \geq \mathbf{u}_j \text{ for all } j \in [d], \\ &\text{if } \mathbf{c}_j < (A^\top \mathbf{y}^*)_j - \text{optTol then } \mathbf{x}^*_j \leq 0 \text{ for all } j \in [d]. \end{aligned} \tag{2}$$

In our analysis we will take $\mathbf{y}^* \geq 0$, but solvers may assume a bit more flexibility and take $\mathbf{y}^* \geq -\text{optTol}$.

Scaling. A well-formed LP should be scaled appropriately: the input data, the output solution, and intermediate values should be not too large and not too small. This is important to the functioning of real-world software.

The primary way in which this scaling requirement has been explained is through the effects of floating-point arithmetic. A poorly scaled matrix is likely to have a high condition number. The numerical concerns described above are an important reason for why proper scaling of the input is explicitly recommended.

The tolerances provide a second important anchor point for scaling. When some of the optimal values are small relative to the tolerances, this can negatively affect the quality of the solution for its intended purpose. When some of the solution values are too large relative to the tolerance, the solver may be unable to satisfy all feasibility and optimality conditions within the small (absolute) tolerance.

It is best practice to choose the measurement units for the rows and columns such that the non-zero values of variables across basic feasible solutions lie in some common, limited range of magnitudes. This practice speaks directly to the geometry of the feasible set. Most LP solvers also include automatic scaling algorithms in case the user's scaling is deemed insufficient.

Overall, the role of scaling for the performance of the simplex method is poorly understood [67, p. 110].

Perturbations. When feasibility tolerances are permitted, solvers can be made faster by intentionally changing the right-hand side $\mathbf{b}$, bounds $\mathbf{0}, \mathbf{u}$, or the objective $\mathbf{c}$. A number of mechanisms make use of this allowance. We focus on one specific such mechanism, which is random perturbation before starting the simplex method. Generally these perturbations are done in such a manner as to make the feasible region larger. The perturbations are of a similar order of magnitude to the feasibility tolerance. The primary use of a priori perturbation in practice is in perturbing the objective vector $\mathbf{c}$ before starting the dual simplex method. During both the primal and dual simplex method, more perturbations and shifts can be performed at later iterations if needed. One consequence of bound perturbations is that degeneracy is avoided and consequently the simplex method cannot cycle. We furthermore analyze their implications on the running time of the simplex method.

As noted by [53, p. 61], perturbations are ubiquitous among LP solvers but "despite its wide application, the discussion of why perturbation works is relatively rare." He highlights that in the survey by Bixby [10] it is noted that more "aggressive" perturbations are experimentally viewed as more effective and that, in personal communication, it became clear there was no theoretical understanding as to why. Our results provide the first theoretical explanation of this phenomenon.

1.5 Related Work

The simplex method has been studied in the worst case complexity framework under various additional assumptions. One line of work has assumed that the constraint matrix $A \in \mathbb{Z}^{n \times d}$ is integer and has every square submatrix have a determinant which is bounded from above in absolute value. Under this assumption, upper bounds on the running time of the simplex method have been proven [15, 21, 23, 32, 33]. The semi-random shadow vertex pivot rule of [23] strongly informed our own. Approximating the maximum subdeterminant is known to be NP-hard [83]. Similar results are available for a closely related condition measure, the circuit imbalance [34]. A study estimating the circuit imbalance on the instances in MIPLIB 2017 [63] found it to require much higher numerical precision than is available with IEEE 754 double precision floating point arithmetic.

A parallel line of work investigates the worst-case performance of the simplex method when the slack values and reduced costs are bounded away from zero [59, 60, 84]. These papers extend an earlier result about the policy iteration algorithm for Markov Decision Processes with bounded discount rate [88], and are similar to a calculation done by Dantzig [28]. The condition number underlying this line of work is NP-hard to compute as shown in [65]. The same paper also estimated this condition number on the LP instances

in the NETLIB test problem set [41], though the measurement approach they used severely limits what conclusions can be drawn based on these estimates. The use of maximal and minimal non-zero slacks was a direct inspiration for our analysis, although we have the minimal non-zero slack be probabilistically bounded using perturbations instead of deterministically by assuming structure in the input data.

In our by-the-book analysis, we use a simplex method with bound perturbations to obtain provable running time guarantees. Bound perturbations are often incorporated in codes with the stated purpose of preventing issues of degeneracy and stalling [62, 67]. Other methods to prevent stalling are possible, including the use of expanding tolerances or minimum step sizes [42]. In a theoretical regime where bounds may not be changed, other approaches are available [14, 64, 72].

A technique akin to, but developed independently of, bound perturbations was previously used in [58] to find a weakly polynomial-time algorithm for linear programming. Although aspects of what they do are similar to what we do here, their arguments adapted to our setting would produce exponentially worse running time dependencies on the parameters we consider.

2 Observations

In the following we discuss our findings in the several open source codes and user manuals. The order of the solvers is alphabetical.

Glop. As part of Google's software suite OR-Tools [77], Glop is an open-source LP solver.

For any finite values in the input LP, Glop by default[1] allows numbers with absolute values as low as 10^{-30} and as high as 10^{30}. However, in order to increase sparsity, by default any floating-point numbers with absolute value below 10^{-14} are set to zero[2].

Glop has configurable primal and dual feasibility tolerances, which have a value of 10^{-6} by default[3]. In the dual simplex method, Glop can perturb costs before starting. When this is done, the default perturbation on the i'th element of the cost vector $\mathbf{c}$ is given by[4]

$$r_i \cdot (10^{-5}\mathbf{c}_i + 10^{-7}\|\mathbf{c}\|_\infty),$$

where r_i is sampled uniformly from the interval $[1, 2]$. This cost perturbation is turned off by default[5].

Gurobi. The Gurobi user manual recommends that any right-hand side coefficients are scaled such that their absolute value is 10^4 or less [47, Section 7.3.4]. The same recommendation is given with regards to the objective coefficients and variable bounds. Similarly, they recommend that every LP has an optimal value less than 10^4. The latest version of Gurobi includes 3 different automatic scaling modes but the documentation does not specify what algorithms are used or when they are used.

In Gurobi, any constraint coefficient whose absolute value is smaller than 10^{-13} is treated as zero [47, Section 7.3.6]. The user manual recommends that the user scales the input problem such that all non-zero matrix coefficients have their absolute value between 10^{-3} and 10^6, and that they span no more than 6 orders of magnitude. The default big-M value (essentially, the finite number that stands in for 'infinity' in ILP formulations) is 10^6, further indicating that coefficients larger than this are undesirable. The primal feasibility tolerance defaults to 10^{-6} but is configurable to any value between 10^{-2} and 10^{-9}. The same holds for the optimality tolerance. The primal tolerance is described as follows [47, Section 7.3]: "$a \cdot x \leq b$ will be considered to hold if $(a * x) - b \leq$ FeasibilityTol." Here, '$\cdot$' is the exact inner product, whereas '$*$' is the computed product (subject to errors).

The Gurobi user manual further indicates that a condition number of 10^{12} for a basic submatrix is considered large [47, Section 7.4]. The Gurobi solver warns the user of large bounds when a variable has a bound with absolute value that is 10^{10} or larger. It warns the user of large objective coefficients when there is a coefficient with absolute value 10^{10} or larger.

Conversation with the developers of Gurobi confirmed that perturbations are actively used in all their simplex codes. Initial perturbations have magnitude proportional to the feasibility tolerance.

HiGHS. In the open-source MIP solver HiGHS [54], the simplex method incorporates two scaling methods, one using equilibration and one based on maximum value[6]. This automatic scaling is not performed for the interior-point method. Notably, if all non-zero elements of the constraint matrix are between 0.2 and 5.0 then no automatic scaling will take place[7].

HiGHS has configurable parameters for the smallest non-zero and largest finite entries that may be present in the constraint matrix[8]. By default, any matrix entry whose absolute value is smaller than 10^{-9} is treated as 0, and any matrix entry whose absolute value is larger than 10^{15} is treated as infinity.

There are configurable primal and dual feasibility tolerances which equal 10^{-7} by default and which can be tuned to be as low as 10^{-10}. In the primal simplex method, when a bound is found to be infeasible, it is shifted to make it feasible by at least the feasibility tolerance, plus a uniformly random quantity between 0 and the feasibility tolerance[9]. In the dual simplex method, the solver perturbs the cost vector already during initialization[10]. These perturbations are uniformly distributed in an interval.

The PhD thesis of Qi Huangfu [53] documents the development of the HiGHS simplex method solver. In his work, he suggests that feasibility and optimality tolerances are typically on the order of 10^{-6} or 10^{-7}. For HiGHS he describes[11] an explicit choice of value for cost perturbation of

$$(10^{-5}|\mathbf{c}_j| + 10^{-7}\|\mathbf{c}\|_\infty + 10 \cdot \text{optTol})(r + 1),$$

[1] https://github.com/google/or-tools/blob/9b770/ortools/glop/parameters.proto#L481
[2] https://github.com/google/or-tools/blob/9b770/ortools/glop/parameters.proto#L183
[3] https://github.com/google/or-tools/blob/9b770/ortools/glop/parameters.proto#L251
[4] https://github.com/google/or-tools/blob/9b770/ortools/glop/reduced_costs.cc#L255
[5] https://github.com/google/or-tools/blob/9b770/ortools/glop/parameters.proto#L415

[6] https://ergo-code.github.io/HiGHS/dev/options/definitions/#simplex_scale_strategy
[7] https://github.com/ERGO-Code/HiGHS/blob/c67f06df422ce1faff4089033513874b8/src/lp_data/HighsLpUtils.cpp#L861
[8] https://ergo-code.github.io/HiGHS/dev/options/definitions/#small_matrix_value
[9] https://github.com/ERGO-Code/HiGHS/blob/c67f06df422ce1faff4089033513874b8/highs/simplex/HEkkPrimal.cpp#L2793
[10] https://github.com/ERGO-Code/HiGHS/blob/c67f06df422ce1faff4089033513874b8/highs/simplex/HEkkDual.cpp#L126
[11] See also https://github.com/ERGO-Code/HiGHS/blob/c67f06df422ce1faff4089/highs/simplex/HEkk.cpp#L2488

where optTol > 0 is the dual feasibility tolerance and r is a uniformly random number in $(0, 1]$. In particular, in his work, the cost perturbation is the sum of a term proportionate to the objective and an absolute perturbation. We model these terms as being the same by assuming our LP is sufficiently well-scaled.

MOSEK. The MOSEK instruction manual [71] mentions that problems containing large or small coefficients, outside the range $[10^{-7}, 10^{9}]$, are often hard to solve[12]. Although it includes a method to automatically scale variables, no specifics are reported about how this scaling is performed or under what circumstances. The solver will print an error[13] to warn the user if any matrix element is larger than 10^{10}, if a cost entry is larger than 10^{8} or if a bound is larger than 10^{8}. If an upper and a lower bound for a variable differ by less than 10^{-8}, then MOSEK will consider the two numbers to be equal[14].

The absolute tolerance[15] for the primal and dual infeasibility both default to 10^{-6}. A technical report [4] mentions that, when troubleshooting infeasible problems, the most important constraints are those whose relative values in the Farkas certificate are at least 10^{-8}.

The MOSEK Modeling Cookbook [70, Section 7.3] considers "a model to be badly scaled if variables are measured in very different scales, or constraints or bounds are measured on very different scales". In practice, it is standard to see violations of inequality constraints of order 10^{-8}.

Interviews with the developers of MOSEK confirm that perturbations are actively used in all their simplex codes, and that adding larger perturbations are an effective tool when the simplex method stalls. Initial perturbations have magnitude proportional to the feasibility tolerance.

MIPLIB 2017. Our main technical result Theorem 3 is stated in terms of the *mean width* of the feasible set of the linear program (after bounds are perturbed). The mean width of a polyhedron P is defined as the expected value of

$$\begin{aligned} &\max\ \boldsymbol{\theta}^\top(\mathbf{x} - \mathbf{x}') \\ &\text{subject to } \mathbf{x}, \mathbf{x}' \in P, \end{aligned}$$

where $\boldsymbol{\theta} \in \mathbb{S}^{d-1}$ is uniformly random distributed on the unit sphere.

In order to get an indication of what the mean width of practical linear programs is, we performed an experiment using the MIPLIB 2017 benchmark set [43]. The experiment is rudimentary in design, foregoing any considerations of automatic scaling or presolve.

For every instance, we performed 100 trials as follows. We sampled an objective vector $\mathbf{z}$ with entries sampled independently at random from a Gaussian distribution with mean 0 and standard deviation 1. Taking the LP relaxation provided by `Gurobi 13.0.1`, we maximized and minimized the objective $\mathbf{z}^\top x$. The difference between the values was divided by $\|\mathbf{z}\|$ to obtain the width in direction $\frac{\mathbf{z}}{\|\mathbf{z}\|}$. Besides `neos-5114902-kasavu`, every individual minimization and maximization completed within the deterministic work limit of 300. Due to Milman's concentration of measure phenomenon, we can expect to get accurate estimates using a small number of samples $\mathbf{z}$.

Any instance finding an unbounded or infeasible status code (`INFEASIBLE`, `INF_OR_UNBD` or `UNBOUNDED`) was dropped. This left 186 instances for which mean widths were estimated. All sample means obtained were between 10^{-1} and 10^{9}, with 46 instances having sample means in $[10^{-1}, 10^{1}]$, 65 having sample means in $[10^{1}, 10^{2}]$, 54 in $[10^{2}, 10^{4}]$, 10 in $[10^{4}, 10^{6}]$ and the remaining two instances with sample means between 10^{6} and 10^{9}. The code for this experiment can be found at https://github.com/sophiehuiberts/miplib-meanwidth, and was run on an Intel i5-14600KF CPU.

Not all MIPLIB instances are properly scaled by the model formulators. Nevertheless, we see that many instances have sample mean widths that are small. We take these results as preliminary evidence that the mean width of the feasible set is bounded from above for well-scaled linear programs. More work is needed to verify whether indeed the better scaled instances tend to have lower mean widths, and how mean widths are affected by automatic scaling and presolve reductions.

3 A Simplex Method with Bound Perturbations

Our analysis makes use of a particular simplex method called the shadow vertex simplex method originally introduced by Gass and Saaty in [40] in the context of parametric linear programming. A textbook introduction that proves the standard properties we rely on may be found in [25].

We assume that we are given a linear program of the form

$$\begin{aligned} &\text{maximize } \mathbf{c}^\top \mathbf{x} \\ &\text{subject to } A\mathbf{x} \le \mathbf{b} \\ &\qquad \mathbf{0} \le \mathbf{x} \le \mathbf{u} \end{aligned} \tag{input-LP}$$

with $A \in \mathbb{R}^{n \times d}$. We assume that every row of A has Euclidean norm 1. Any LP can easily be transformed to this format. We will solve this LP up to tolerances feasTol > 0 and optTol > 0.

To start, we perturb our bound vectors $\mathbf{0}, \mathbf{u}$ and the right-hand side vector $\mathbf{b}$. The perturbation distribution is defined as follows:

Definition 1. *Let $\eta, \gamma \in \mathbb{R}_{>0}$, and let $\mathbf{v} \in \mathbb{R}^k$. Then define for each $i \in [k]$,*

$$f_i(t) = \frac{1}{2\eta} e^{-|t - \mathbf{v}_i - \gamma\eta|/\eta}.$$

Define a random vector $\hat{\mathbf{v}}$ to be $(\mathbf{v}, \eta, \gamma)$-exponentially distributed if its entries $\hat{\mathbf{v}}_i$ are independently distributed with each entry's probability density function given by f_i.

We sample the bounds and right-hand sides $(-\hat{\mathbf{0}}, \hat{\mathbf{u}}, \hat{\mathbf{b}}) \in \mathbb{R}^{n+2d}$ to be $\left((\mathbf{0}, \mathbf{u}, \mathbf{b}), \frac{\text{feasTol}}{4\ln(n+2d)}, 2\ln(n+2d)\right)$-exponentially distributed. Strong tail bounds will ensure that these perturbed bounds are suitable:

Lemma 2. *Let $\eta, \gamma \in \mathbb{R}_{>0}$, and let $\mathbf{v} \in \mathbb{R}^k$. Let $\hat{\mathbf{v}}$ be a $(\mathbf{v}, \eta, \gamma)$-exponentially distributed random vector. Then*

$$\Pr\left[\hat{\mathbf{v}}_i \in [\mathbf{v}_i, \mathbf{v}_i + 2\gamma\eta] \text{ for all } i \in [k]\right] \ge 1 - ke^{-\gamma}.$$

[12] https://docs.mosek.com/latest/pythonapi/presolver.html#index-5
[13] https://docs.mosek.com/latest/pythonapi/parameters.html#mosek.dparam.data_tol_aij_large
[14] https://docs.mosek.com/latest/pythonapi/parameters.html#mosek.dparam.data_tol_x
[15] https://docs.mosek.com/latest/pythonapi/parameters.html#mosek.dparam

It directly follows from Lemma 2 that with probability greater than $1 - \frac{1}{n+2d}$ we have

$$\begin{aligned} -\text{feasTol} \cdot \mathbf{1}_d \leq \hat{\mathbf{0}} \leq \mathbf{0} \\ \mathbf{u} \leq \hat{\mathbf{u}} \leq \mathbf{u} + \text{feasTol} \cdot \mathbf{1}_d \\ \mathbf{b} \leq \hat{\mathbf{b}} \leq \mathbf{b} + \text{feasTol} \cdot \mathbf{1}_n. \end{aligned} \tag{3}$$

In order to ensure that the above inequalities are guaranteed to hold for our algorithm's output, we reject any vector $(\hat{\mathbf{0}}, \hat{\mathbf{u}}, \hat{\mathbf{b}})$ which violates these conditions and run the entire algorithm anew. This rejection sampling increases the expected running time by a factor $\frac{n+2d}{n+2d-1} \leq 2$.

We will use a simplex method to approximately maximize the perturbed linear program

$$\begin{aligned} &\text{maximize } \mathbf{c}^\top \mathbf{x} \\ &\text{subject to } A\mathbf{x} \leq \mathbf{b} \\ &\qquad \mathbf{0} \leq \mathbf{x} \leq \mathbf{u}. \end{aligned} \tag{perturbed-LP}$$

We do this by following a sequence of simplex paths using the shadow vertex pivot rule. For every path, there is one fixed objective unit vector, either $\mathbf{c}$ or an auxiliary objective vector, and one random objective vector $\mathbf{z} \in \mathbb{R}^d$ with probability density function $\mathbf{x} \mapsto \exp(-\|\mathbf{x}\|)$.

Theorem 3. *Assume $\varepsilon > 0$. Let $A \in \mathbb{R}^{n \times d}$, $\mathbf{c} \in \mathbb{S}^{d-1}$, let $\hat{\mathbf{b}} \in \mathbb{R}^n$ be $(\mathbf{b}, \eta, \gamma)$-exponentially distributed, let $\mathbf{z} \in \mathbb{R}^d$ have probability density function $\mathbf{x} \mapsto \exp(-\|\mathbf{x}\|)$. If the perturbed LP*

$$\max\{\mathbf{c}^\top \mathbf{x} : A\mathbf{x} \leq \hat{\mathbf{b}}\}$$

has expected mean width M and maximum absolute objective value N, then the shadow path from a vertex maximizing $\mathbf{z}$ to a vertex maximizing $\mathbf{c} + \varepsilon\mathbf{z}$ has expected length at most

$$O\left(d^{1.5} \ln(n+d) \sqrt{\frac{M}{\eta} \ln\left(\frac{dN \ln(n)}{\eta \cdot \varepsilon}\right)}\right).$$

Choosing $\eta = \text{feasTol}/4\ln(n+2d)$ and $\varepsilon = \text{optTol}/d\kappa$, where κ is an upper bound on the maximum condition number of any basis, we obtain the running time of Phase II of our simplex method.

For Phase I of the simplex method we use the semi-random shadow vertex rule and the `sequential` algorithm described in [22]. Here, the inequality constraints $A\mathbf{x} \leq \hat{\mathbf{b}}$ are added to the LP one by one. This Phase I approach loses a factor n in the running time, yielding a total running time of

$$O\left(nd^{1.5} \ln(n+d) \sqrt{\frac{M}{\text{feasTol}} \ln\left(\frac{d\kappa N \ln(n)}{\text{feasTol} \cdot \text{optTol}}\right)}\right).$$

pivot steps.

After approximately maximizing the perturbed Phase II LP, the resulting basis will be output. This basis will satisfy (1) and (2).

4 Proof Overview

The proof of our main technical result Theorem 3 proceeds in three phases. First, we describe the effect of the right-hand side perturbations. We show that, conditional on a basis being feasible, the corresponding basic solution will have all its non-zero slack values on the inequalities be large. Specifically, the smallest non-zero slack will be at least $\frac{\eta}{1240d\ln(n)}$ with probability at least 0.9. As in the previous paragraph, η is a measure of the magnitude of the perturbations. The lower bound on the slacks is derived using a Chernoff bound. This argument is derived from a similar one used in smoothed analyses of the simplex method [7, 55].

Second, we show a dual analog to the above, based on the shadow vertex simplex path from a random auxiliary objective $\mathbf{z}$ to a largely fixed objective $\varepsilon^{-1}\mathbf{c} + \mathbf{z}$. We show that, conditional on a basis lying on this shadow path, with probability at least 0.9 there exists an intermediate objective $t\mathbf{c} + \mathbf{z}$ with $t \in [0, \varepsilon^{-1}]$ for which this basis is optimized and for which the reduced costs on the tight constraints are large. Note that the existence of $t \in [0, \varepsilon^{-1}]$ for which the basis is optimal for $t\mathbf{c} + \mathbf{z}$ is guaranteed by virtue of being on the shadow path. The novelty is that we can guarantee a choice of t yielding large reduced costs on tight constraints. In order to simplify this part of the analysis (found in the full version of this paper), we assume that the random objective $\mathbf{z}$ is sampled from an L-log-Lipschitz probability distribution. This will not affect the final conclusion.

Finally, we show our upper bound on the length of the semi-random shadow vertex path. Taking everything together, we show that a large fraction of the feasible bases on this path must have both (in the above explained senses) large non-zero slacks and admit an intermediate objective for which all d reduced costs are large. Afterwards, we argue that whenever two subsequent bases on this path both have both properties, then the pivot step leading from the first to the second of them must make large progress. This progress is measured through four different potential functions. Two of these potential functions measure the progress of the intermediate objective vector $t\mathbf{c} + \mathbf{z}$ on the line segment from $\mathbf{z}$ to $\varepsilon^{-1}\mathbf{c} + \mathbf{z}$. The third and fourth potentials encode progress in the objective value and auxiliary objective value respectively. A majority of pivot steps will consume a certain amount of at least one of these four potential functions. The total amount of potential available is bounded, which leads to our upper bound on the expected number of pivot steps.

5 Discussion

In this section we evaluate how well the present analysis comports with further observation and how well our overall methodological structure ultimately comported with the general by-the-book analysis framework. Of course, in any mathematical analysis compromises are inevitable in order to accommodate the analyst's ability to produce proofs. Since the impetus for performing a by-the-book analysis (and for the by-the-book analysis framework itself) is to produce theoretical results which have greater capacity to explain what is observed in practice, we propose that it is essential to a by-the-book analysis to evaluate how successful the analysis was in comporting with its observations. This is in part to provide a realistic assessment of the explanatory power of our results, but for this work, in which we propose the by-the-book analysis framework, it is also to differentiate the strengths and weaknesses of the by-the-book framework as a general algorithm anylysis framework from the strengths and weaknesses of the particular by-the-book analysis we performed.

5.1 Input Assumptions

Our bound is written as polynomial in:

$$d, n, \eta, M, \ln(N), \ln(\kappa), \text{optTol, and feasTol.}$$

The standard input size measures for any linear program are d and n, the number of variables and number of inequalities respectively. For weakly polynomial run-time bounds, $\ln(\kappa)$ and $\ln(N)$ are also standard to include as part of the measure of the input-size as they are bounded by the bit-complexity of the linear program. The remaining parameters η, optTol, and feasTol are all directly related to our modeling assumptions. We contend that, as supported by our research and computational experiments, it is reasonable to assume the chosen bounds on these numbers in a real-world computational setting. Likewise, our assumption that the rows of the input constraint matrix have 1-norm equal to 1 is reasonable and unobtrusive. This is due to the fact that common LPs are hyper sparse, often containing only a constant number of non-zero elements in every inequality constraint. The mean width M was already discussed in detail in Section 2.

5.2 Model Assumptions

Our analysis has strong requirements on the pivot rule and ratio test, needing variants that are considered inefficient in practice and needing both to be performed exactly, without any loss of numerical precision.

We require that bounds are perturbed once and never shifted again. In particular, this requires a numerically exact implementation of Dantzig's ratio test, ruling out the possibility of using Harris' ratio test. Since any state of the art implementation of the simplex method benefits from Harris' ratio test [67, p. 179], this is a notable limitation of the current analysis. We use a priori perturbations of the bounds and right-hand side. In our observations, we found that HiGHS does a priori perturbations not in the primal simplex method, but only to the cost vector in the dual simplex method. A more realistic model of the simplex method would allow for pivot steps which are slightly infeasible, and would allow to add and subtract perturbations throughout the running time.

When perturbing, our analysis requires a particular choice of probability density function for the individual perturbations. This is different from current implementations, which sample uniformly from an interval. We do not currently know how our probability distribution compares to the uniform distribution in practice. As such, we can not evaluate whether this is an unrealistic modeling choice or a possible improvement over the state-of-the-art.

For the pivot rule, we require that the semi-random shadow vertex path is followed. For our analysis, the randomness in the perturbations and pivot rule should furthermore be independent of each other and of the input data. This rules out a wide range of sophisticated pricing strategies, including steepest edge pivoting, partial pricing and multiple pricing (see [74, p. 112-114] and [67, p. 186-188]), as well as specific pivoting rules used in Phase I of the simplex method [74, p. 116-117]. In the dual, it rules out the bound-flipping ratio test.

The shadow vertex rule has been criticized for the difficulty of following it with appropriate pivot tolerances in a numerical setting [74, p130]. As such, it does not see much use.

There are many other techniques in active use among high-quality simplex implementations that we did not bring into consideration for this paper.

5.3 Methodology

Although we broadly succeeded in following the proposed methodology of by-the-book analysis, we did lightly deviate from this structure during the third phase. We originally planned to let an assumption of bounded geometric diameter act as a model and proxy for the input being well-scaled. It was only after beginning mathematical analysis in the third phase that we discovered the mean width may be a better parameter for this purpose. In one sense this is in conflict with the principle that our assumptions should be informed solely by a priori observations, and not retroactively informed by the proofs.

On the other hand, we believe that it is natural and even inevitable to discover over the course of the analysis that early choices of assumptions for modeling one's observations can be improved by better ones. In this case, we did not so much introduce a new assumption as refine the parameter we were using to model an existing assumption (in this case, the assumption that our LP was well-scaled in some concrete way). Moreover, in the spirit of by-the-book analysis, we were only content to allow our bounds to depend on M provided we could verify that it is bounded in practice, and indeed it is actually a strength of our shift from geometric diameter to mean width that the mean width is more easily measured (or rather, accurately estimated) than the geometric diameter. However, it does remain to be seen whether the simplex method's performance truly has a meaningful dependence on mean width; as we note in Section 6, this requires further investigation.

5.4 Outcomes

Although our results are a significant step forward in the study of the simplex method, they do not constitute a complete quantitative explanation of its real-world performance. We can evaluate the analysis and the theorem statement both quantitatively and qualitatively.

Scaling with problem size. Our bound contains a large constant factor dependence, much bigger than the scaling with problem size observed in practice. In the course of this work, only limited effort was expended to reduce this constant factor.

Our running time bound scales faster with the problem size d than the linear growth observed in practice. At the moment of writing it is unclear which losses happen due to inefficiencies in the proof, and which happen due to insufficient accounting of the geometry of the feasible set.

Size of perturbations. Larger and more aggressive perturbations are known to improve performance [10]. A common technique in solvers, used when the simplex method stalls with repeated zero-length pivot steps, is to add increasingly large perturbations until stalling resolves. Although these perturbations do help the simplex method make progress, it comes at a cost: the perturbations must later be removed in order to return a solution that is feasible up to the desired tolerance.

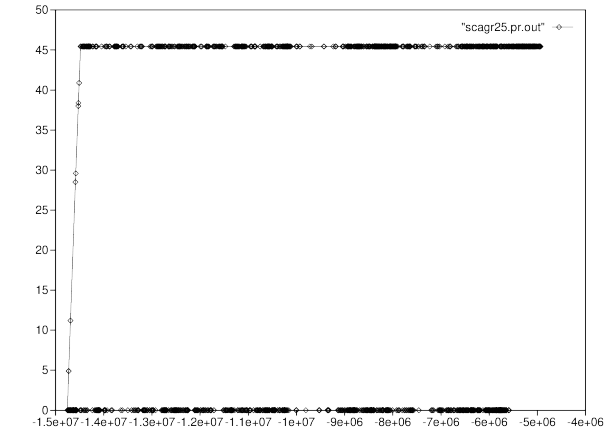
The running time bounds in our theorems contain a one-over-square-root dependence on the feasibility tolerance. Interpreted naively, this would suggest that changing the feasibility tolerance by a factor 100 would change the running time by as much as a factor 10. The observed effect of changing the tolerances or perturbation sizes is not nearly this large. The likely source of this disparity is that, in our analysis, step length is governed only by the perturbations: we did not use the geometry of the feasible set.

In order to improve the running time bounds, and have a bound which does not scale as strongly with the tolerance, taking into account this geometry is necessary. Otherwise, a running time dependence of $\sqrt{M/\eta}$ is tight, matching a lower bound construction up to polynomial factors in d and poly-logarithmic factors in M/η. This lower bound is obtained by taking $n = (c/\eta)^{d/2}$ constraints, for some absolute constant $c > 1$, and having the feasible set approximate a Euclidean unit ball.

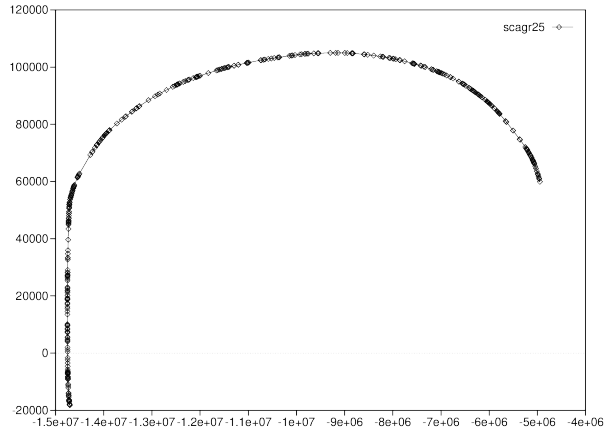
Length and curvature. In our mathematical analysis, we prove that vertices visited by the semi-random shadow vertex method tend to take up space in the primal, by having at least one long shadow edge incident, or take up space in the dual, by creating curvature of the shadow polygon. Without randomizing the shadow plane with respect to the data, the theory would not predict this effect to happen.

This prediction aligns with findings from Fischer. In his Diplomarbeit [36], he used the shadow vertex method to draw projections of the Netlib LP problems. On the horizontal axis he plotted the value of the objective $c^T x$. For the vertical axis he either used fixed objectives x_0, x_1, or $a_i^\top x$, or random objectives $z^\top x$ with $z \in [-1, 1]^d$ uniformly random. Four consecutive shadow paths were followed $c \to z \to -c \to -z \to c$ to find the complete shadow image. In cases where the shadow is unbounded, only the vertices are drawn. Upon request, we obtained the images from G. M. Ziegler.

For the instance `scagr25`, Fischer drew two plots: one with a deterministic shadow $(x_0, c^\top x)$ and one with a semi-random shadow $(z^\top x, c^\top x)$. These can be seen in Figure 2 which depict the two projections. For the semi-random shadow in Figure 2 (b), we see a strong correspondence: when a given part of the boundary with constant length has more vertices (and thus, shorter edges), then it is more curved. For the deterministic shadow in Figure 2 (a), we do not see the same correspondence. At least on the qualitative level, Fischer's plots agree with the prediction from our analysis.



(a) x_0 for vertical axis



(b) Random objective for vertical axis

Figure 2: Two-dimensional projections of `scagr25`, LP objective for horizontal axis [36].

6 Conclusion

In this paper we introduced the framework of by-the-book analysis and used it to study the performance of the simplex method. We have presented a running time analysis for a simplex method with bound perturbations, obtaining an upper bound on the expected running time of

$$O\left(nd^{1.5}\ln(n+d)\sqrt{\frac{M}{\text{feasTol}}\ln\left(\frac{d\kappa N\ln(n)}{\text{feasTol}\cdot\text{optTol}}\right)}\right)$$

pivot steps required to maximize the largely fixed objective $\mathbf{c} + \varepsilon\boldsymbol{\theta}, \boldsymbol{\theta} \in \mathbb{S}^{d-1}$, with bound perturbations of order η. Although this work is predominantly dedicated to introducing and performing a by-the-book analysis of the simplex method, we note that this bound on the expected running time holds for arbitrary inputs and is mathematically sound even outside of the context of by-the-book analysis. We highlight the two most significant improvements over earlier work.

First, from a purely theoretical lens, our analysis works for sparse constraint data, in contrast to smoothed and average-case analysis. Since all practical LPs are sparse (see, e.g., [74, p. 64] and [67, p. 50]), this is an important feature. Second, due to the fact that we followed the by-the-book framework proposed herein, the assumptions we make are grounded in observations from simplex method implementations, LP modeling best practices, and measurements from practical benchmark instances. Our running time primarily depends on the LP being well scaled and the simplex method featuring bound perturbations. Both of these properties are essential to any large-scale LP solvers [67, p. 110, 241]. In this manner, the results of this by-the-book analysis correspond well with established knowledge on the simplex method as it is used in practice. This is an advantage of our above bound, even when it is interpreted purely as a highly parameterized theoretical result;

the parameters themselves were chosen not out of mathematical convenience but due to the fact that our research and experiments indicate that these parameters are bounded in practice, and may indeed play a genuine role in its behavior.

The exact role of scaling for linear programs has previously not been well understood, but is known to reduce the number of pivot steps [67, p. 110-111] [76, p. 98]. There is no broad agreement on how to define when an LP is well scaled [86]. We take our theorems and experiments to indicate that the mean width of the feasible set is bounded for "well-scaled" instances. A more in-depth investigation is in order to substantiate or refute that belief. Are the instances with small mean width indeed the ones that would be subjectively judged as well-scaled? Are instances with large mean width typically harder to solve for a simplex method? These are only some of the questions that need to be answered in order to confidently speak about whether small mean width is indeed a desirable property of linear programs.

Despite the limitations highlighted in Section 5, we believe that the present work is a significant step forward towards a theoretical understanding of the simplex method's real-world performance. Moreover, we expect our proposed framework of by-the-book analysis to be applicable to many more algorithms. Any algorithm could be the subject of a by-the-book analysis, provided that there is enough computational experience and/or high-quality open-source code available. A by-the-book analysis may be especially desirable in situations where existing theoretical results struggle to match the known practical efficiency of an algorithm or require unrealistic assumptions.

References

[1] Ilan Adler, Richard M. Karp, and Ron Shamir. 1987. A simplex variant solving an $m \times d$ linear program in $O(\min(m^2, d^2))$ expected number of pivot steps. *Journal of Complexity* 3, 4 (1987), 372–387. doi:10.1016/0885-064X(87)90007-0

[2] Ilan Adler and Nimrod Megiddo. 1985. A simplex algorithm whose average number of steps is bounded between two quadratic functions of the smaller dimension. *Journal of the ACM (JACM)* 32, 4 (1985), 871–895. doi:10.1145/4221.4222

[3] Nina Amenta and Günter M. Ziegler. 1998. Deformed Products and Maximal Shadows. *Contemporary Math.* 223 (1998), 57–90.

[4] Erling D. Andersen. 2013. *How to use Farkas' lemma to say something important about infeasible linear problems.* Technical Report. MOSEK. https://docs.mosek.com/whitepapers/infeas.pdf

[5] Neculai Andrei. 2004. On the complexity of MINOS package for linear programming. *Studies in Informatics and Control* 13, 1 (2004), 35–46.

[6] David Avis and Vasek Chvátal. 1978. Notes on Bland's pivoting rule. In *Polyhedral Combinatorics*. Springer, 24–34. doi:10.1007/BFb0121192

[7] Eleon Bach and Sophie Huiberts. 2025. Optimal Smoothed Analysis of the Simplex Method. In *2025 IEEE 66th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1829–1856. doi:10.1109/focs63196.2025.00096

[8] Keith Ball. 1997. *Flavors of Geometry*. MSRI Book Series, Vol. 31. Chapter An Elementary Introduction to Modern Convex Geometry. https://library.slmath.org/books/Book31/files/ball.pdf

[9] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. 2009. *Robust Optimization*. Princeton University Press. https://www2.isye.gatech.edu/~nemirovs/FullBookDec11.pdf

[10] R. Bixby. 2002. Solving real-world linear programs: A decade and more of progress. *Operations Research* 50, 1 (2002), 3–15.

[11] Alexander Black. 2023. Small Shadows of Lattice Polytopes. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, 1669–1679. doi:10.1137/1.9781611977554.ch63

[12] A. Black, J. De Loera, S. Kafer, and L. Sanità. 2025. On the simplex method for 0/1-polytopes. *Mathematics of Operations Research* 50, 2 (2025), 1398–1420. doi:10.1287/moor.2021.0345

[13] Alexander E. Black. 2025. *Exponential Lower Bounds for Many Pivot Rules for the Simplex Method*. Springer Nature Switzerland, 86–99. doi:10.1007/978-3-031-93112-3_7

[14] Robert G. Bland. 1977. New Finite Pivoting Rules for the Simplex Method. *Mathematics of Operations Research* 2, 2 (May 1977), 103–107. doi:10.1287/moor.2.2.103

[15] Nicolas Bonifas, Marco Di Summa, Friedrich Eisenbrand, Nicolai Hähnle, and Martin Niemeier. 2012. On sub-determinants and the diameter of polyhedra. In *Proceedings of the twenty-eighth annual symposium on Computational geometry (SoCG '12)*. ACM, 357–362. doi:10.1145/2261250.2261304

[16] Gilles Bonnet, Daniel Dadush, Uri Grupel, Sophie Huiberts, and Galyna Livshyts. 2026. Asymptotic Bounds on the Combinatorial Diameter of Random Polytopes. *Discrete & Computational Geometry* (Feb. 2026). arXiv:2112.13027 doi:10.1007/s00454-025-00814-6

[17] Karl-Heinz Borgwardt. 1982. The Average number of pivot steps required by the Simplex-Method is polynomial. *Zeitschrift für Operations Research* 26, 1 (1982), 157–177. doi:10.1007/BF01917108

[18] Karl Heinz Borgwardt. 1977. *Untersuchungen zur Asymptotik der mittleren Schrittzahl von Simplexverfahren in der linearen Optimierung*. Ph. D. Dissertation. Universität Kaiserslautern.

[19] Karl Heinz Borgwardt. 1987. *The Simplex Method: A Probabilistic Analysis*. Vol. 1. Springer-Verlag, Berlin. doi:10.1007/978-3-642-61578-8

[20] Karl Heinz Borgwardt. 1999. A Sharp Upper Bound for the Expected Number of Shadow Vertices in Lp-Polyhedra Under Orthogonal Projection on Two-Dimensional Planes. *Mathematics of Operations Research* 24, 3 (1999), 544–603. doi:10.1287/moor.24.4.925

[21] Tobias Brunsch and Heiko Röglin. 2013. Finding Short Paths on Polytopes by the Shadow Vertex Algorithm. In *Automata, Languages, and Programming (ICALP '13)*. Springer Berlin Heidelberg, Berlin, Heidelberg, 279–290. doi:10.1007/978-3-642-39206-1_24

[22] Leola Cutler and Philip Wolfe. 1963. *Experiments in Linear Programming*. Technical Report. The RAND Corporation. https://apps.dtic.mil/sti/tr/pdf/AD0297757.pdf

[23] Daniel Dadush and Nicolai Hähnle. 2016. On the shadow simplex method for curved polyhedra. *Discrete Comput. Geom.* 56, 4 (2016), 882–909. doi:10.1007/s00454-016-9793-3

[24] Daniel Dadush and Sophie Huiberts. 2020. A Friendly Smoothed Analysis of the Simplex Method. *SIAM J. Comput.* 49, 5 (Jan. 2020), STOC18–449–499. doi:10.1137/18m1197205

[25] Daniel Dadush and Sophie Huiberts. 2021. *Smoothed Analysis of the Simplex Method*. Cambridge University Press, 309–333. doi:10.1017/9781108637435.019

[26] George Dantzig. 1963. *Linear Programming and Extensions*. Princeton University Press. doi:10.1515/9781400884179

[27] George B Dantzig. 1951. Maximization of a linear function of variables subject to linear inequalities. *Cowles Commission Monograph 13: Activity analysis of production and allocation* (1951), 339–347.

[28] George B. Dantzig. 1990. Origins of the simplex method. 141–151 pages. doi:10.1145/87252.88081

[29] Amit Deshpande and Daniel A Spielman. 2005. Improved smoothed analysis of the shadow vertex simplex method. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*. 349–356. doi:10.1109/SFCS.2005.44

[30] Yann Disser, Oliver Friedmann, and Alexander V. Hopp. 2022. An exponential lower bound for Zadeh's pivot rule. *Mathematical Programming* (July 2022). doi:10.1007/s10107-022-01848-x

[31] Yann Disser and Nils Mosis. 2023. A Unified Worst Case for Classical Simplex and Policy Iteration Pivot Rules. In *34th International Symposium on Algorithms and Computation (ISAAC 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 283)*, Satoru Iwata and Naonori Kakimura (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 27:1–27:17. doi:10.4230/LIPIcs.ISAAC.2023.27

[32] Martin Dyer and Alan Frieze. 1994. Random walks, totally unimodular matrices, and a randomised dual simplex algorithm. *Mathematical Programming* 64, 1–3 (March 1994), 1–16. doi:10.1007/bf01582563

[33] Friedrich Eisenbrand and Santosh Vempala. 2016. Geometric random edge. *Mathematical Programming* 164, 1–2 (Nov. 2016), 325–339. doi:10.1007/s10107-016-1089-0

[34] Farbod Ekbatani, Bento Natura, and László A. Végh. 2022. *Circuit Imbalance Measures and Linear Programming*. Cambridge University Press, 64–114. doi:10.1017/9781009093927.004

[35] FICO. 2023. FICO Xpress Optimizer Reference Manual. https://www.fico.com/fico-xpress-optimization/docs/dms2023-04/solver/optimizer/HTML/chapter4.html?scroll=subsection400

[36] Stefan Fischer. 1998. *Zweidimensionale Projektionen von linearen Programmen*. Technical Report. TU Berlin.

[37] John J. Forrest and Donald Goldfarb. 1992. Steepest-edge simplex algorithms for linear programming. *Mathematical Programming* 57, 3 (1992), 341–374. doi:10.1007/BF01581089

[38] Oliver Friedmann. 2011. A Subexponential Lower Bound for Zadeh's Pivoting Rule for Solving Linear Programs and Games. In *Proceedings of 15th International*

Conference on Integer Programming and Combinatorial Optimization, Oktay Günlük and Gerhard J. Woeginger (Eds.). Springer, 192–206. doi:10.1007/978-3-642-20807-2_16

[39] Oliver Friedmann, Thomas Dueholm Hansen, and Uri Zwick. 2011. Subexponential Lower Bounds for Randomized Pivoting Rules for the Simplex Algorithm. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*. 283–292. doi:10.1145/1993636.1993675

[40] Saul Gass and Thomas Saaty. 1955. The computational algorithm for the parametric objective function. *Naval research logistics quarterly* 2, 1-2 (1955), 39–45.

[41] David M Gay. 1985. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Newsletter* 13 (1985), 10–12. https://netlib.org/lp/data/

[42] Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright. 1989. A practical anti-cycling procedure for linearly constrained optimization. *Mathematical Programming* 45, 1–3 (Aug. 1989), 437–474. doi:10.1007/bf01589114

[43] Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. 2021. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation* (2021). doi:10.1007/s12532-020-00194-3

[44] Donald Goldfarb. 1976. Using the steepest-edge simplex algorithm to solve sparse linear programs. In *Sparse matrix computations*. Elsevier, 227–240. doi:10.1016/B978-0-12-141050-6.50018-0

[45] Donald Goldfarb. 1983. Worst case complexity of the shadow vertex simplex algorithm. *preprint, Columbia University* (1983).

[46] Donald Goldfarb and William Y Sit. 1979. Worst case behavior of the steepest edge simplex method. *Discrete Applied Mathematics* 1, 4 (1979), 277–285. doi:10.1016/0166-218X(79)90004-0

[47] Gurobi Optimization, LLC. 2025. Gurobi Optimizer Reference Manual. https://docs.gurobi.com/_/downloads/optimizer/en/13.0/pdf/

[48] M. Haimovich. 1983. *The simplex method is very good! On the expected number of pivot steps and related properties of random linear programs.* Technical Report. Columbia University.

[49] J. A. J. Hall and K. I. M. McKinnon. 2005. Hyper-Sparsity in the Revised Simplex Method and How to Exploit it. *Computational Optimization and Applications* 32, 3 (Dec. 2005), 259–283. doi:10.1007/s10589-005-4802-0

[50] Thomas Dueholm Hansen and Uri Zwick. 2015. An improved version of the random-facet pivoting rule for the simplex algorithm. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. 209–218. doi:10.1145/2746539.2746557

[51] Paula M. J. Harris. 1973. Pivot selection methods of the Devex LP code. *Mathematical Programming* 5, 1 (Dec. 1973), 1–28. doi:10.1007/bf01580108

[52] J. N. Hooker. 1994. Needed: An Empirical Science of Algorithms. *Operations Research* 42, 2 (April 1994), 201–212. doi:10.1287/opre.42.2.201

[53] Qi Huangfu. 2013. *High performance simplex solver*. Ph. D. Dissertation. University of Edingburgh. http://hdl.handle.net/1842/7952

[54] Q. Huangfu and J. A. J. Hall. 2017. Parallelizing the dual revised simplex method. *Mathematical Programming Computation* 10, 1 (Dec. 2017), 119–142. doi:10.1007/s12532-017-0130-5

[55] Sophie Huiberts, Yin Tat Lee, and Xinzhi Zhang. 2023. Upper and Lower Bounds on the Smoothed Complexity of the Simplex Method. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*. ACM. doi:10.1145/3564246.3585124

[56] Robert G Jeroslow. 1973. The simplex algorithm with the pivot rule of maximizing criterion improvement. *Discrete Mathematics* 4, 4 (1973), 367–377. doi:10.1016/0012-365X(73)90171-4

[57] Gil Kalai. 1992. A subexponential randomized simplex algorithm. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. 475–482. doi:10.1145/129712.129759

[58] Jonathan A. Kelner and Daniel A. Spielman. 2006. A randomized polynomial-time simplex algorithm for linear programming. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, Jon M. Kleinberg (Ed.). ACM, 51–60. doi:10.1145/1132516.1132524

[59] Tomonari Kitahara and Shinji Mizuno. 2011. A bound for the number of different basic solutions generated by the simplex method. *Mathematical Programming* 137, 1-2 (Aug. 2011), 579–586. doi:10.1007/s10107-011-0482-y

[60] Tomonari Kitahara and Shinji Mizuno. 2012. On the number of solutions generated by the dual simplex method. *Operations Research Letters* 40, 3 (May 2012), 172–174. doi:10.1016/j.orl.2012.01.004

[61] Victor Klee and George J Minty. 1972. How good is the simplex algorithm. *Inequalities : III : proceedings of the 3rd Symposium on inequalities* (1972), 159–175.

[62] Achim Koberstein. 2005. *The Dual Simplex Method, Techniques for a fast and stable implementation*. Ph. D. Dissertation. Universitat Paderborn. https://digital.ub.uni-paderborn.de/hsmig/content/titleinfo/3885/full.pdf

[63] Jakub Komárek and Martin Koutecký. 2024. Experimental Analysis of LP Scaling Methods Based on Circuit Imbalance Minimization. In *22nd International Symposium on Experimental Algorithms (SEA 2024) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 301)*, Leo Liberti (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 18:1–18:21. doi:10.4230/LIPIcs.SEA.2024.18

[64] Kirill Kukharenko and Laura Sanità. 2024. *On the Number of Degenerate Simplex Pivots*. Springer Nature Switzerland, 252–264. doi:10.1007/978-3-031-59835-7_19

[65] Takahito Kuno, Yoshio Sano, and Takahiro Tsuruda. 2018. Computing Kitahara–Mizuno's bound on the number of basic feasible solutions generated with the simplex algorithm. *Optimization Letters* 12, 5 (May 2018), 933–943. doi:10.1007/s11590-018-1276-4

[66] Andrew Makhorin. 2017. GLPK (GNU Linear Programming Kit) documentation.

[67] István Maros. 2002. *Computational Techniques of the Simplex Method*. Springer. doi:10.1007/978-1-4615-0257-9

[68] Jiří Matoušek, Micha Sharir, and Emo Welzl. 1996. A subexponential bound for linear programming. *Algorithmica* 16, 4-5 (1996), 498–516. doi:10.1145/142675.142678

[69] Nimrod Megiddo. 1986. Improved asymptotic analysis of the average number of steps performed by the self-dual simplex algorithm. *Math. Programming* 35, 2 (1986), 140–172. doi:10.1007/BF01580645

[70] MOSEK ApS. 2025. MOSEK Modeling Cookbook 3.4.0. https://docs.mosek.com/modeling-cookbook/index.html

[71] MOSEK ApS. 2025. MOSEK Optimizer API for Python: Release 11.0.28. https://docs.mosek.com/11.0/pythonapi.pdf

[72] K. Murty. 2008. Complexity of degeneracy. In *Encyclopedia of Optimization*. Springer, 419–425.

[73] Katta G Murty. 1980. Computational complexity of parametric linear programming. *Mathematical programming* 19, 1 (1980), 213–219. doi:10.1007/BFb0120782

[74] William Orchard-Hays. 1968. *Advanced linear-programming computing techniques*. McGraw-Hill.

[75] William Orchard-Hays, Leola Cutler, and Harold Judd. 1956. *Manual for the RAND-IBM Code for Linear Programming*. Technical Report. The RAND Corporation.

[76] Ping-Qi Pan. 2016. *Linear Programming Computation*. Springer. doi:10.1007/978-3-642-40754-3

[77] Laurent Perron and Vincent Furnon. 2025. OR-Tools. https://developers.google.com/optimization/

[78] Tim Roughgarden (Ed.). 2020. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press. doi:10.1017/9781108637435

[79] Emanuel Schnalzger. 2014. *Lineare Optimierung mit dem Schatteneckenalgorithmus im Kontext probabilistischer Analysen*. Ph. D. Dissertation. Universität Augsburg. English translation by K.H. Borgwardt..

[80] Rolf Schneider. 2013. *Convex Bodies: The Brunn–Minkowski Theory* (2 ed.). Cambridge University Press.

[81] Ron Shamir. 1987. The efficiency of the simplex method: a survey. *Management science* 33, 3 (1987), 301–334. doi:10.1287/mnsc.33.3.301

[82] Daniel A Spielman and Shang-Hua Teng. 2004. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM (JACM)* 51, 3 (2004), 385–463. doi:10.1145/990308.990310

[83] Marco Di Summa, Friedrich Eisenbrand, Yuri Faenza, and Carsten Moldenhauer. 2014. On largest volume simplices and sub-determinants. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 315–323. doi:10.1137/1.9781611973730.23

[84] Masaya Tano, Ryuhei Miyashiro, and Tomonari Kitahara. 2019. Steepest-edge rule and its number of simplex iterations for a nondegenerate LP. *Operations Research Letters* 47, 3 (May 2019), 151–156. doi:10.1016/j.orl.2019.02.003

[85] Michael J. Todd. 1986. Polynomial expected behavior of a pivoting algorithm for linear complementarity and linear programming problems. *Mathematical Programming* 35, 2 (1986), 173–192. doi:10.1007/BF01580646

[86] J. A. Tomlin. 1975. *On scaling linear programming problems*. Springer Berlin Heidelberg, 146–166. doi:10.1007/bfb0120718

[87] Roman Vershynin. 2009. Beyond Hirsch conjecture: walks on random polytopes and smoothed complexity of the simplex method. *SIAM J. Comput.* 39, 2 (2009), 646–678. doi:10.1137/070683386

[88] Yinyu Ye. 2005. A New Complexity Result on Solving the Markov Decision Problem. *Mathematics of Operations Research* 30, 3 (Aug. 2005), 733–749. doi:10.1287/moor.1050.0149