Computation of exact bootstrap confidence intervals: complexity and deterministic algorithms

Dimitris Bertsimas and Bradley Sturt*

August 23, 2017

Abstract

The bootstrap is a nonparametric approach for calculating quantities, such as confidence intervals, directly from data. Since calculating exact bootstrap quantities is believed to be intractable, randomized resampling algorithms are traditionally used. Motivated by the fact that the variability from randomization can lead to inaccurate outputs, we propose a deterministic approach. First, we establish several computational complexity results for the exact bootstrap method, in the case of the sample mean. Second, we present the first efficient, deterministic approximation algorithm (FPTAS) for producing exact bootstrap confidence intervals which, unlike traditional methods, has guaranteed bounds on the approximation error. Third, we develop a simple exact algorithm for exact bootstrap confidence intervals based on polynomial multiplication. We provide empirical evidence involving several hundreds (and in some cases over one thousand) data points that the proposed deterministic algorithms can quickly produce confidence intervals that are substantially more accurate compared to those from randomized methods, and are thus practical alternatives in applications such as clinical trials.

Index terms— Bootstrap method, computational complexity, deterministic approximation algorithms, integral points in polyhedra, Monte Carlo simulation.

1 Introduction

Given a sample $z_1, \ldots, z_n \in \mathbb{R}$, a fundamental task is measuring the closeness of its sample mean $\hat{\mu} = n^{-1}\sum_{i=1}^{n} z_i$ to the underlying population mean μ. Quantities such as confidence intervals on the sample mean help to provide insight. If the data comes from some known probability distribution, such as the exponential distribution, confidence intervals can be constructed directly. However, in real life, the distribution is typically unknown. Alternatively, if n is large, asymptotic theory provides justification for confidence intervals of the form $[\hat{\mu} - a, \hat{\mu} + a]$. When n is not large, the central limit theorem may provide a poor approximation of the sampling distribution, particularly when the data is asymmetric. In these circumstances, resampling methods, in particular the bootstrap method, are widely used.

The bootstrap method [Efron, 1979, Efron and Tibshirani, 1994] is a computational technique for performing statistical inference directly from the data. Its use by practitioners is ubiquitous across management science, risk analysis, and clinical trials, among many others. The bootstrap is typically computed with a randomized algorithm. The practitioner randomly generates B new data sets by drawing with replacement from the original data set. The sample statistic, such as the mean, is calculated for each of the B "bootstrap samples", and the empirical distribution of the means constitutes the "bootstrap distribution" of $\hat{\mu}$. The bootstrap distribution forms the foundation for inference; for example, an approximate 95% confidence interval for $\hat{\mu}$ can be calculated taking the 2.5% and 97.5% quantiles of the bootstrap distribution.

The bootstrap method is hence a simulation approach, aiming to approximate the exact bootstrap quantities, which are the results we would obtain if we calculated *all* possible means generated from *all* possible bootstrap samples [Fisher and Hall, 1991]. For instance, the exact bootstrap distribution $G(\alpha)$ of the sample mean is the proportion of all possible bootstrap samples that have mean less or equal to a number α,

*Operations Research Center, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, dbertsim@mit.edu, bsturt@mit.edu.

$$G(\alpha) := \frac{1}{n^n} \left| \left\{ \mathbf{z}^* \in \{z_1, \ldots, z_n\}^n : \frac{1}{n} \sum_{i=1}^{n} z_i^* \leq \alpha \right\} \right|, \tag{1}$$

where $\{z_1, \ldots, z_n\}^n$ is the n^n-length set of all possible bootstrap samples $\mathbf{z}^*$ of the data $z_1, \ldots, z_n$. From $G(\alpha)$, we let $H(\beta)$ denote the exact β-th bootstrap quantile,

$$H(\beta) := \min \{\alpha : G(\alpha) \geq \beta\}. \tag{2}$$

$H(\beta)$ is the foundation of popular approaches for constructing bootstrap confidence intervals for the sample mean, such as the percentile, percentile-t, and bias-corrected and accelerated (BCa) methods [Efron, 1979, 1987]. For a detailed account and comparsion of bootstrap confidence intervals, we refer the reader to DiCiccio and Efron [1996], Efron and Tibshirani [1994], Hall [1988], and the references therein.

The common technique of using randomization to approximate $H(\beta)$ results in variability. As B increases, the variability introduced by the randomization typically decreases. In the real world, B is always finite, and since randomness is involved, different results may be obtained each time the algorithm is run. We define the randomization error as the difference between the results from bootstrap calculations with finite B and the exact bootstrap quantities.

The error caused by randomization can have significant negative consequences. For example, consider a clinical trial where a treatment to shrink tumor sizes is used on a group of subjects. The bootstrap can be used to estimate confidence intervals around the average change in tumor sizes post treatment. Due to the practitioner's choice for the number of bootstrap samples B, randomization error will be present in the confidence intervals. This error can be surprisingly large, even when $B = 1$ million (see the example in Section 6.1). In this clinical example, poor estimates of the real effect of treatment are highly consequential, as they can potentially impact health care outcomes.

The uncertainty of the extent of randomization error can cast doubt on the validity of the result. The availability of fast computational resources enables one to use increasingly large B's. At the same time, this also allows for running the algorithm many times, possibly allowing one to run many iterations of the algorithm and present only the "lucky" iteration that had the desirable result, such as small confidence intervals. This problem persists even when using importance sampling or other efficient simulation schemes. When presented with a confidence interval from a clinical trial, we cannot be certain whether the results are representative of a typical confidence interval produced by the randomized algorithm.

A better option is to use a deterministic algorithm, to either precisely calculate the exact bootstrap quantities or to approximate them with guaranteed bounds on the error. A deterministic method would remove uncertainty regarding randomization error, and alleviate practitioners of the task of choosing B. A recent body of literature has demonstrated the power of deterministic methods over randomized methods, such as in experimental design for controlled trials [Bertsimas et al., 2015].

Existing literature has proposed deterministic methods for calculating exact bootstrap quantities. For small samples (e.g., $n \leq 9$), Fisher and Hall [1991] proposed a method for explicit enumeration of all possible bootstrap samples. Huang [1991] and Hutson and Ernst [2000] present combinatorial and analytical approaches for calculating the exact bootstrap variance for L-statistics using order statistics. Evans et al. [2006] proposes a different method based on order statistics for when the data falls in a discrete set. The exact bootstrap distribution of the sample median can be found in closed form [Efron, 1982]. These approaches are specific for certain quantities, such as the standard deviation of the bootstrap distribution, or are specialized for the median. Other analytic approximations have been developed for specific types of bootstrap confidence intervals, such as the ABC approximation for BCa confidence intervals [DiCiccio and Efron, 1992]. However, the ABC approximation of the BCa confidence intervals for the sample mean can be inaccurate when the underlying distribution has heavy tails [Efron and Tibshirani, 1994, Section 14.5].

In this paper, we consider deterministic algorithms and associated computational complexity results for computing exact bootstrap quantiles for the sample mean, from which confidence intervals can be obtained. Our approach is to view $G(\alpha)$ and $H(\beta)$ as *counting* problems; in particular, we show in Section 2 that $G(\alpha)$ is equivalent to counting the number of integral points in a polyhedron. Such counting problems have attracted significant interest in operations research as a result of their connection to integer optimization, sampling methods in simulation, and approximation algorithms [Bertsimas and Weismantel, 2005, Jerrum and Sinclair, 1996, Lasserre, 2009]. By relating the exact bootstrap method to integer counting problems, we develop new insights and deterministic approaches for the bootstrap method.

1.1 Literature review

The complexity of counting problems.

The study of counting problems spans several decades in operations research and computer science. Valiant [1979a,b] developed the computational complexity class $\#P$, which contains the problems of counting the number of solutions to a decision problem. Some problems in $\#P$ can be solved in polynomial time. Examples include counting paths in a directed acyclic graph via topological sort [Cormen et al., 2009] and counting spanning trees in a network using the Cauchy-Binet formula [Harris et al., 2008, Section 1.3.4].

Informally, a problem is considered $\#P$-hard if it is at least as hard as every problem in $\#P$. Problems that are $\#P$-hard include the counting versions of many problems that are NP-complete. Additionally, the counting versions of some problems in P are $\#P$-hard, such as counting the number of distinct matchings in a bipartite graph. The class $\#P$ is theoretically at least as hard as NP. In practice, exactly solving $\#P$-hard problems is considered highly intractable. The existence of polynomial-time algorithms for every problem in $\#P$ would have significant implications, including that $P = NP$. For a comprehensive background on NP and $\#P$, we refer the interested reader to [Arora and Barak, 2009, Garey and Johnson, 1990].

Many fundamental problems in operations research and statistics are $\#P$-hard. Of particular relevance to bootstrap is that of counting integer points in a polyhedron $\{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$, which is $\#P$-hard even if there is only a single constraint [Dyer et al., 1993]. Other examples of $\#P$-hard problems include counting the number of vertices of a polyhedron [Linial, 1986], solving two-stage linear stochastic optimization problems [Dyer and Stougie, 2006, Hanasusanto et al., 2016], computing the volume of a polyhedron [Dyer and Frieze, 1988], and the network reliability problem [Valiant, 1979b]. Examples from statistics include counting the exact number of $2 \times n$ contingency tables with specified column and row sums [Dyer et al., 1997].

Exact and deterministic approximation algorithms.

Given a polyhedron $\{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ with n variables and m constraints, the *integer counting problem* asks for the number of integer points in the polyhedron. As the integer counting problem is $\#P$-hard, no algorithm that is polynomial in n and the size of $(\mathbf{A}, \mathbf{b})$ is known.

Algorithms have been proposed to exactly solve the integer counting problem that are efficient under certain circumstances. First, there are polynomial time algorithms for the integer counting problem when the dimension n is fixed, the first presented in Barvinok [1994]. Further work on fixed-dimension algorithms has been done (see Lasserre [2009] and the references therein), and Barvinok's algorithm has been implemented in a package called `LattE` [De Loera et al., 2004]. In the case of the bootstrap, however, the algorithm of Barvinok is neither theoretically nor empirically efficient, as discussed in Section 6. Second, Nesterov [2004] proposed counting the number of binary points in knapsack polyhedron $\{\mathbf{x} \in \{0,1\}^n : \mathbf{a}^T\mathbf{x} \leq b\}$ via the coefficients of the polynomial $\Pi_{i=1}^n (1 + t^{a_i})$, which could be computed via the Fast Fourier Transform (FFT). In Section 4, we develop a specific and fast algorithm for exact bootstrap quantiles motivated by polynomial multiplication, and provide a detailed analysis of its bit complexity.

Although it is unlikely that polynomial-time algorithms exist for the $\#P$-hard integer counting problem, deterministic polynomial-time *approximation* algorithms have been developed. The first deterministic approximation algorithm for #Knapsack, the binary counting problem for a polyhedron with a single inequality constraint $\{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^T\mathbf{x} \leq b\}$, was presented in Dyer [2003], whose dynamic programming algorithm produced a $\sqrt{n+1}$-factor approximation of #Knapsack in $O(n^3)$ time.

Štefankovic et al. [2012] and Gopalan et al. [2011] proposed the first fully-polynomial time approximation scheme (FPTAS) for #Knapsack. A deterministic approximation algorithm is an FPTAS if, given any $\epsilon > 0$, it produces a solution with value that is within a $(1 \pm \epsilon)$ factor of the exact answer in time polynomial in the input size and ϵ^{-1}. Their algorithm has a bit complexity of $\tilde{O}\left(\frac{n^3}{\epsilon} \log b\right)$ and is based on a dynamic programming formulation (we use the notation that $f(n) = \tilde{O}(g(n))$ if $f(n) = O(g(n) \log^k g(n))$ for some k). The FPTAS algorithm has also been extended to the integer variant of #Knapsack [Halman, 2016] and the cumulative distribution function of the sum of non-identical discrete random variables with countable support [Li and Shi, 2014]. In Section 3, we develop an FPTAS for the bootstrap based on similar techniques.

1.2 Contributions and Structure

In this paper, we present theoretical results and practical deterministic algorithms for the bootstrap method, for the case of the sample mean as well as higher moments. The main contributions are as follows:

1. We develop several computational complexity results for the exact bootstrap method. Specifically, we show that computing $G(\alpha)$ and $H(\beta)$ are $\#P$-hard. To the best of our knowledge, these are the first such complexity results for the bootstrap method, and underscore the computational difficulty of exact bootstrap computations in many cases. Additionally, we show that the computation of $\mathbb{P}(\sum_{i=1}^n X_i \leq x)$ for X_i i.i.d. discrete random variables is $\#P$-hard.

2. We propose the first efficient deterministic approximation algorithm (FPTAS) for computing the exact bootstrap quantile $H(\beta)$. Specifically, for any data set of n points and $\epsilon > 0$, the algorithm produces a $(1+\epsilon)$-factor approximation of $H(\beta)$ with a bit complexity of $\tilde{O}\left(\frac{n^4}{\epsilon}\log z_{(n)}\right)$, where $z_{(n)}$ is the largest data value. The algorithm thus directly allows for deterministic computation of confidence intervals, removing randomization from the bootstrap computations.

3. We present and analyze an exact algorithm for the exact bootstrap quantiles based on polynomial multiplication and a technique of Kronecker [1882]. The algorithm has a bit complexity of $\tilde{O}\left(n^2 z_{(n)}\right)$, and is practically tractable for data sets of values represented with several significant digits.

4. We perform computational experiments that compare deterministic bootstrap confidence intervals to those from the traditional randomized algorithm. First, we show an example where the confidence intervals produced using traditional methods have substantial error resulting from randomization, even when $B = 1$ million bootstrap samples were generated. This underscores the importance of determinism in bootstrap computations. Second, we show that the proposed algorithms can find bootstrap confidence intervals without any randomization error in minutes for data sets containing several hundreds (and, in some cases, over one thousand) data points. This demonstrates that the proposed deterministic methods are practical alternatives to the traditional methods in certain applications such as clinical trials.

We have structured our paper as follows. In Section 2, we present the main computational complexity results. In Section 3, we propose a deterministic approximation algorithm for calculating exact bootstrap quantiles. In Section 4, we present an exact algorithm for calculating bootstrap quantiles. In Section 5, we show extensions of the deterministic algorithms from Sections 3 and 4 to statistics beyond the sample mean. In Section 6, we discuss computational experiments that exemplify the tractability and accuracy of deterministic bootstrap computations over the traditional randomized approach. In Section 7, we conclude and discuss future directions.

2 Computational Complexity

In this section, we present computational complexity results for exact bootstrap calculations for the sample mean. Specifically, we show that computing $G(\alpha)$ and $H(\beta)$ is $\#P$-hard. To the best of our knowledge, these are the first computational complexity results regarding the bootstrap method. They underscore the widely-held belief that calculating exact bootstrap quantities is difficult. As a corollary, we present the result that the exact calculation of $\mathbb{P}(\sum_{i=1}^n X_i \leq x)$ for i.i.d. discrete random variables is $\#P$-hard.

The key intuition in this section is that the exact bootstrap method is directly equivalent to the problem of counting the number of integer points in a polyhedron. For example, $G(\alpha)$ is equal to $\frac{1}{n^n}|P \cap \mathbb{Z}^n|$, where

P is the following polyhedron:

$$P \coloneqq \left\{ \gamma \in \mathbb{R}^{n \times n} : \begin{array}{rl} \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} z_i \gamma_{ij} \leq \alpha, & \\ \sum_{i=1}^{n} \gamma_{ij} = 1 & \text{for all } j = 1, \dots, n \\ 0 \leq \gamma_{ij} \leq 1 & \text{for all } i, j = 1, \dots, n \end{array} \right\}. \tag{3}$$

Indeed, the γ's in the above set have a one-to-one mapping to the bootstrap samples $\mathbf{z}^*$ with mean less than or equal to α. Specifically, each γ in P corresponds to the bootstrap sample $\mathbf{z}^*$ where $z_j^* = z_i$ if and only if $\gamma_{ij} = 1$.

2.1 Complexity of the bootstrap method

The complexity results in this section are based on the following lemma.

Lemma 2.1. *Given* $\mathbf{z} \in \mathbb{N}^n$ *and* $\alpha \in \mathbb{N}$, *computing*

$$\left| \left\{ \mathbf{z}^* \in \{z_1, \dots, z_n\}^n : \sum_{i=1}^{n} z_i^* = \alpha \right\} \right| \tag{4}$$

exactly is #P-hard.

Proof. Proof Given $\mathbf{a} \in \mathbb{N}^n$ and $b \in \mathbb{N}$, let $S(\mathbf{a}, b) \coloneqq \left\{ \mathbf{x} \in \{0,1\}^n : \mathbf{a}^T\mathbf{x} = b \right\}$. The problem of computing $|S(\mathbf{a}, b)|$, i.e., counting the number of binary vectors $\mathbf{x} \in \{0,1\}^n$ such that $\mathbf{a}^T\mathbf{x} = b$, is well-known to be #$P$-hard [Dyer et al., 1993]. Our reduction consists of a reduction from $|S(\mathbf{a}, b)|$.

Given $\mathbf{a}, b$, let $M = 2n(n+1)b$, and construct new vectors $\tilde{\mathbf{a}} \in \mathbb{N}^{2n+1}$ and $\tilde{b} \in \mathbb{N}$, where

$$\tilde{a}_i = \begin{cases} M^{n+1} + M^i + a_i, & \text{if } i \in \{1, \dots, n\}, \\ M^{n+1} + M^{i-n}, & \text{if } i \in \{n+1, \dots, 2n\}, \\ 0, & \text{if } i = 2n+1, \end{cases}$$

$$\tilde{b} = nM^{n+1} + \sum_{i=1}^{n} M^i + b.$$

Consider the set $\tilde{S}(\mathbf{a}, b)$, defined as

$$\tilde{S}(\mathbf{a}, b) \coloneqq \left\{ \mathbf{z}^* \in \{\tilde{a}_1, \dots, \tilde{a}_{2n+1}\}^{2n+1} : \sum_{i=1}^{2n+1} z_i^* = \tilde{b} \right\}.$$

$\tilde{S}(\mathbf{a}, b)$ is of the desired form in Lemma 2.1. In the remainder of the proof, we will show that $|S(\mathbf{a}, b)| = \frac{(n+1)!}{(2n+1)!}|\tilde{S}(\mathbf{a}, b)|$, in which case the #$P$-hard problem $|S(\mathbf{a}, b)|$ can be reduced in polynomial time to $|\tilde{S}(\mathbf{a}, b)|$.

- First, we show that $|S(\mathbf{a}, b)| \leq \frac{(n+1)!}{(2n+1)!}|\tilde{S}(\mathbf{a}, b)|$. Consider any $\mathbf{x} \in S(\mathbf{a}, b)$. Define $\mathbf{z}^* \in \mathbb{N}^{2n+1}$ as

$$z_i^* = \begin{cases} \tilde{a}_i x_i + \tilde{a}_{i+n}(1 - x_i), & \text{for } i \in \{1, \dots, n\}, \\ \tilde{a}_{2n+1}, & \text{for } i \in \{n+1, \dots, 2n+1\}. \end{cases}$$

By construction,

$$\sum_{i=1}^{2n+1} \mathbf{z}_i^* = \sum_{i=1}^{n} (\tilde{a}_i x_i + \tilde{a}_{i+n}(1 - x_i)) = nM^{n+1} + \sum_{i=1}^{n} M^i + \sum_{i=1}^{n} a_i x_i = \tilde{b},$$

which shows that $\mathbf{z}^* \in \tilde{S}(\mathbf{a}, b)$. The value of $\sum_{i=1}^{2n+1} \mathbf{z}_i^*$ is indifferent to the order of the elements, hence each permutation of $\mathbf{z}^*$ is also in $\tilde{S}(\mathbf{a}, b)$. By construction, $\mathbf{z}^*$ has $\frac{(2n+1)!}{(n+1)!}$ distinct permutations. Therefore, $|S(\mathbf{a}, b)| \leq \frac{(n+1)!}{(2n+1)!}|\tilde{S}(\mathbf{a}, b)|$.

- Second, we show that $|S(\mathbf{a}, b)| \geq \frac{(n+1)!}{(2n+1)!}|\tilde{S}(\mathbf{a}, b)|$. Consider any element $\mathbf{z}^* \in \tilde{S}(\mathbf{a}, b)$. Define $\mathbf{y} \in \mathbb{N}^{2n+1}$ such that $y_i := |\{j : z_j^* = \tilde{a}_i\}|$ for each $i \in \{1, \dots, 2n+1\}$. In words, y_i is the number of elements of $\mathbf{z}^*$ that are equal to z_i^*. Then,

$$\sum_{i=1}^{2n+1} z_i^* = \sum_{i=1}^{2n+1} \tilde{a}_i y_i = \left(\sum_{i=1}^{2n} y_i\right) M^{n+1} + \sum_{i=1}^{n} \left((y_i + y_{i+n}) M^i\right) + \sum_{i=1}^{n} y_i a_i.$$

Combining the above with $\sum_{i=1}^{2n+1} z_i^* = \tilde{b}$ produces

$$\left(n - \sum_{i=1}^{2n} y_i\right) M^{n+1} + \sum_{i=1}^{n} (1 - (y_i + y_{i+n})) M^i + \left(b - \sum_{i=1}^{n} y_i a_i\right) = 0.$$

For notational convenience, let $\mathbf{c} = (c_0, \dots, c_{n+1}) \in \mathbb{Z}^{n+2}$ denote the coefficients on $M^0, \dots, M^{n+1}$; that is,

$$\sum_{i=0}^{n+1} c_i M^i = 0,$$

where $c_0 = b - \sum_{i=1}^{n} y_i a_i$, $c_i = 1 - (y_i + y_{i+n})$ for $i \in \{1, \dots, n\}$, and $c_{n+1} = n - \sum_{i=1}^{2n} y_i$.

We will now show that each coefficient $c_0, \dots, c_{n+1}$ equals 0. First, without loss of generality, we will assume that $a_i \leq b$ for every $i \in \{1, \dots, n\}$ (if $a_i > b$ for some i, then $x_i = 0$ for every $\mathbf{x} \in S(\mathbf{a}, b)$, which implies that we may remove the i-th variable without changing $|S(\mathbf{a}, b)|$). This implies that $\tilde{a}_i \leq \tilde{b}$ for all $i \in \{1, \dots, 2n\}$. Therefore, for all $i \in \{1, \dots, 2n\}$,

$$y_i \leq \frac{\tilde{b}}{\tilde{a}_i} < n + 1 = \frac{M}{2nb}.$$

By plugging in this strict inequality into the definitions of $c_0, \dots, c_{n+1}$, we observe that $-M < c_i < M$ for each $i \in \{0, \dots, n+1\}$. Therefore, it must be the case that $c_i = 0$ for all $i \in \{0, \dots, n+1\}$. It immediately follows that

$$\begin{aligned} \sum_{i=1}^{n} a_i y_i &= b, \\ y_i + y_{i+n} &= 1 \qquad \forall i \in \{1, \dots, n\}, \\ \sum_{i=1}^{n} y_i &= n. \end{aligned}$$

Hence, for any $\mathbf{z}^* \in \tilde{S}(\mathbf{a}, b)$, $n + 1$ of its components are 0. Out of the remaining n components, there is exactly one occurrence of either $\tilde{a}_i$ or $\tilde{a}_{i+n}$ for each $i \in \{1, \dots, n\}$. Therefore, each $\mathbf{z}^*$ corresponds to exactly one $\mathbf{x} \in S(\mathbf{a}, b)$ by the transformation described in the first part of the proof. Hence, we have shown that $|S(\mathbf{a}, b)| \geq \frac{(n+1)!}{(2n+1)!}|\tilde{S}(\mathbf{a}, b)|$.

Combining the previous two results, we have proved that $|S(\mathbf{a}, b)| = \frac{(n+1)!}{(2n+1)!}|\tilde{S}(\mathbf{a}, b)|$. We thus can reduce the $\#P$-hard problem $|S(\mathbf{a}, b)|$ to counting the number of points in $\tilde{S}(\mathbf{a}, b)$. This proves that the problem of computing $|\{\mathbf{z}^* \in \{z_1, \dots, z_n\}^n : \sum_{i=1}^{n} z_i^* = \alpha\}|$ is $\#P$-hard. □

Using Lemma 2.1, we readily obtain the complexity of computing the exact bootstrap distribution $G(\alpha)$ for the sample mean.

Theorem 2.1. *Computing $G(\alpha)$ exactly is $\#P$-hard.*

Proof. Proof For any $\mathbf{z} \in \mathbb{N}^n$ and $\alpha \in \mathbb{N}$, we can reduce Equation (4) to $G(\alpha)$ as

$$\left| \left\{ \mathbf{z}^* \in \{z_1, \ldots, z_n\}^n : \sum_{i=1}^n z_i^* = \alpha \right\} \right| = n^n \left(G\left(\frac{\alpha}{n}\right) - G\left(\frac{\alpha - 1}{n}\right) \right).$$

□

Next, we show the complexity of computing the exact bootstrap quantiles $H(\beta)$ for the sample mean.

Theorem 2.2. *Computing $H(\beta)$ exactly is #P-hard.*

Proof. Proof By definition, $H(\beta)$ is monotonically increasing in β. Thus, for any $\mathbf{z} \in \mathbb{N}^n$ and $\alpha \in \mathbb{N}$, we can reduce $G(\alpha)$ to a binary search on $H(\beta)$ over β. There are n^n different bootstrap samples, which implies that $H(\beta)$ takes on $O(n^n)$ distinct values. Thus, the binary search requires $O(\log(n^n)) = O(n \log n)$ oracle calls to $H(\beta)$. □

2.2 Complexity of probability

The next result, while not directly related to the bootstrap, we find to be of independent interest.

Corollary 2.1. *Let $X_1, \ldots, X_n$ be i.i.d. discrete random variables with support containing at least n distinct values. Then, computing $\mathbb{P}(\sum_{i=1}^n X_i \leq \alpha)$ exactly is #P-hard.*

Proof. Proof Let $z_1, \ldots, z_n \in \mathbb{R}$ be distinct. Then, $G(\alpha) = \mathbb{P}(\sum_{i=1}^n X_i \leq \alpha)$, where $X_1, \ldots, X_n$ are independent random variables, uniformly distributed over $\{z_1, \ldots, z_n\}$. □

To the best of our knowledge, this is the first result of #P-hardness for computing the cumulative distribution function of a sum of identically distributed random variables. The closest results of which we are aware are for the sum of non-identical Bernoulli random variables [Halman et al., 2009, Kleinberg et al., 1997]. Specifically, it has been shown that $\mathbb{P}(\sum_{i=1}^n X_i \leq \alpha)$ is #P-hard when X_i are independent and distributed as a Bernoulli with rate p_i.

Interestingly, we can approximate $\mathbb{P}\left(\sum_{i=1}^n X_i \leq x\right)$ using a normal distribution via the central limit theorem. If $\Phi(\cdot)$ is the cumulative distribution function of a standard normal, and each $X_1, \ldots, X_n$ has mean μ and standard deviation σ, then

$$\mathbb{P}\left(\sum_{i=1}^n X_i \leq \alpha\right) \approx \Phi\left(\frac{\alpha - n\mu}{\sqrt{n}\sigma}\right). \tag{5}$$

Moreover, if $\mathbb{E}\left[|X_1 - \mu|^3\right] < +\infty$, then the approximation error from the normal distribution is bounded uniformly via the well-known Berry-Esseen theorem. Specifically, the normal distribution calculation is a $Cn^{-1/2}\sigma^{-3}\mathbb{E}[|X_1 - \mu|^3]$ additive error approximation, where $C < 3$ is a constant that does not depend on X (we refer the interested reader to Durrett [2010] for a detailed discussion of Berry-Esseen). Thus, the normal distribution provides a constant-time approximation algorithm for the #P-hard problem with error that is computed from the data.

3 A deterministic approximation algorithm for bootstrap

In this section, we present an efficient, deterministic approximation algorithm for the exact bootstrap quantile $H(\beta)$, from which confidence intervals are obtained. Specifically, for any $\epsilon > 0$, data set $(z_1, \ldots, z_n)$ of positive integers, and $\beta \in (0, 1)$, the proposed algorithm produces a $(1 + \epsilon)$-factor approximation of $H(\beta)$ with a bit complexity of $\tilde{O}\left(\frac{n^4}{\epsilon} \log z_{(n)}\right)$, where $z_{(n)}$ is the largest data point. This result adds the problem of computing $H(\beta)$ to the growing list of #P-hard problems in operations research and statistics that have an FPTAS.

In Section 3.1, we describe our algorithm, which is based on dynamic programming. In Section 3.2, we analyze its bit complexity. Apart from the theoretical tractability, the proposed algorithm is fast in practice, see Section 6.

The proposed approximation algorithm, as well as the exact algorithm in Section 4, assumes that the data points are integral. Nevertheless, if the data $z_1, \ldots, z_n$ are positive numbers each having m significant bits, then the data readily be transformed into integers via multiplying each value by 2^m.

3.1 A dynamic programming algorithm

We begin with a recursive perspective of $G(\alpha)$. Given a data vector $\mathbf{z} \in \mathbb{R}^n_{>0}$, let $\gamma_i(\cdot)$ for $i = 1, \ldots, n$ be defined as

$$\gamma_i(\alpha) \coloneqq \left| \left\{ \mathbf{z}^* \in \{z_1, \ldots, z_n\}^i : \sum_{j=1}^{i} z_j^* \leq \alpha \right\} \right|. \tag{6}$$

In words, $\gamma_i(\alpha)$ is the problem of counting the number of vectors $\mathbf{z}^* \in \{z_1, \ldots, z_n\}^i$ for which the sum of its elements does not exceed α. If $i = n$, then $n^{-n}\gamma_n(n\alpha) = G(\alpha)$. The following recursion holds:

$$\gamma_i(\alpha) = \sum_{j=1}^{n} \gamma_{i-1}(\alpha - z_j),$$

with a base case γ_0 defined as

$$\gamma_0(\alpha) = \begin{cases} 1, & \text{if } \alpha \geq 0, \\ 0, & \text{if } \alpha < 0. \end{cases}$$

Indeed, the recursion follows since $\gamma_i(\alpha) = \sum_{\ell=1}^{n} |\{\mathbf{z}^* \in \{z_1, \ldots, z_n\}^{i-1} : \sum_{j=1}^{i-1} z_j^* \leq \alpha - z_\ell^*\}|$. Computing $\gamma_n(\alpha)$ exactly is $\#P$-hard, as shown in Section 2.1. Instead, we consider approximating $\gamma_1(\alpha), \ldots, \gamma_n(\alpha)$ by evaluation only at a restricted set of α. To describe the restricted set, we introduce some terminology. Let $Q_0, \ldots, Q_s$ denote any sequence for which $Q_0 = 1$, $Q_{\ell+1}/Q_\ell \leq 1 + \log(\epsilon + 1)/(n + 1)$ for each ℓ, and $Q_s \geq z_{(n)}n$. Intuitively, such a sequence behaves like a geometric progression over the range $[1, z_{(n)}n]$. Given any $\alpha \geq 1$, let $Q^{-1}(\alpha)$ be defined as the largest Q_ℓ for which $Q_\ell \leq \alpha$.

We now define approximations $\tilde{\gamma}_0, \ldots, \tilde{\gamma}_n$ of $\gamma_0, \ldots, \gamma_n$. Let $\tilde{\gamma}_0(\alpha) \coloneqq \gamma_0(\alpha)$ for all α. For $i \in \{1, \ldots, n\}$, let $\tilde{\gamma}_i$ be defined with a similar recursion to γ_i, with the distinction that each α is rounded down to the nearest Q_ℓ:

$$\tilde{\gamma}_i(\alpha) \coloneqq \sum_{j=1}^{n} \tilde{\gamma}_{i-1}\left(Q^{-1}(\alpha) - z_j\right). \tag{7}$$

For any $\alpha \in [Q_\ell, Q_{\ell+1})$, $\tilde{\gamma}_i(\alpha) = \tilde{\gamma}_i(Q_\ell)$; thus, $\tilde{\gamma}_i(\alpha)$ is entirely specified by evaluation at $\alpha \in \{Q_0, \ldots, Q_s\}$. We claim that the functions $\tilde{\gamma}_1, \ldots, \tilde{\gamma}_n$ are indeed close approximations to $\gamma_1, \ldots, \gamma_n$, as formalized in the following lemma.

Lemma 3.1. *For all $i \in \{0, \ldots, n\}$, $\tilde{\gamma}_i$ is non-decreasing, and for all $\alpha \in \mathbb{R}_+$,*

$$\gamma_i\left(r^{-i}\alpha\right) \leq \tilde{\gamma}_i(\alpha) \leq \gamma_i(\alpha), \tag{8}$$

where $r \coloneqq 1 + \log(\epsilon + 1)/(n + 1)$.

Proof. Proof The result follows from induction on i. If $i = 0$, then $\tilde{\gamma}_0 = \gamma_0$ implies that $\tilde{\gamma}_0$ is clearly non-decreasing and satisfies Equation (8). If $i > 0$, then $\tilde{\gamma}_i$ is the sum of non-decreasing functions, which implies it is non-decreasing. Next, we show that $\tilde{\gamma}_i$ satisfies Equation (8) by showing the two sides of the inequality.

$$\tilde{\gamma}_i(\alpha) \leq \sum_{j=1}^{n} \gamma_{i-1}\left(Q^{-1}(\alpha) - z_j\right) \leq \sum_{j=1}^{n} \gamma_{i-1}(\alpha - z_j) = \gamma_i(\alpha).$$

The first inequality is from the induction hypothesis. The second inequality follows since γ_{i-1} is non-decreasing. We now show the other side of the inequality.

$$\tilde{\gamma}_i(\alpha) \geq \sum_{j=1}^{n} \tilde{\gamma}_{i-1}\left(r^{-1}\alpha - z_j\right) \geq \sum_{j=1}^{n} \gamma_{i-1}\left(\left(r^{-1}\alpha - z_j\right) r^{-(i-1)}\right) \geq \sum_{j=1}^{n} \gamma_{i-1}\left(r^{-i}\alpha - z_j\right) = \gamma_i\left(r^{-i}\alpha\right).$$

The first inequality follows by two observations: first, $Q_{\ell+1}/Q_\ell \leq r$ implies that $Q^{-1}(\alpha) \geq r^{-1}\alpha$; second, by the induction hypothesis, $\tilde{\gamma}_{i-1}$ is non-decreasing. The second inequality follows from the induction hypothesis. The third inequality follows since $-z_j \leq -z_j r^{-(i-1)}$ and γ_{i-1} is non-decreasing. □

For any $\beta \in (0,1)$, let $\mathcal{H}(\beta)$ be defined as

$$\mathcal{H}(\beta) := \min\left\{\alpha : n^{-n}\tilde{\gamma}_n(n\alpha) \geq \beta\right\}.$$

Then, for every $\beta \in (0,1)$, $\mathcal{H}(\beta)$ is a $(1+\epsilon)$-factor deterministic approximation of $H(\beta)$, as shown in the following result.

Lemma 3.2. *For all $\epsilon > 0$ and $\beta \in (0,1)$,*

$$H(\beta) \leq \mathcal{H}(\beta) \leq (1+\epsilon)H(\beta). \tag{9}$$

Proof. Proof We show that each inequality holds. First,

$$H(\beta) = \min\left\{\alpha : n^{-n}\gamma_n(n\alpha) \geq \beta\right\} \leq \mathcal{H}(\beta),$$

where the inequality follows from Lemma 3.1. Second,

$$\mathcal{H}(\beta) \leq \min\left\{\alpha : n^{-n}\gamma_n(r^{-n}n\alpha) \geq \beta\right\} = r^n H(\beta) \leq (1+\epsilon)H(\beta),$$

where $r = 1 + \log(\epsilon+1)/(n+1)$. The first inequality follows from Lemma 3.1, and the second inequality follows by the definition of r. □

We recall that $\tilde{\gamma}_i(\alpha) = \tilde{\gamma}_i(Q_\ell)$ for all $\alpha \in [Q_\ell, Q_{\ell+1})$. Therefore, if we could efficiently obtain $\tilde{\gamma}_n(Q_\ell)$ for each $\ell \in \{0,\ldots,s\}$, then we can compute $\mathcal{H}(\beta)$ for any β by a binary search over $\tilde{\gamma}_n(Q_0),\ldots,\tilde{\gamma}_n(Q_s)$.

We now describe an efficient algorithm to compute $\tilde{\gamma}_1,\ldots,\tilde{\gamma}_n$ based on dynamic programming. Let $L^{-1}(\ell,j)$ be defined as the largest index ℓ' for which $Q_{\ell'} \leq Q_\ell - z_j$. If $Q_\ell - z_j < 1$, then no such ℓ' exists, and $L^{-1}(\ell,j)$ returns a special symbol such as $-\infty$. Define A as a two-dimensional array where $A[i,\ell] = \tilde{\gamma}_i(Q_\ell)$ for each $i \in \{0,\ldots,n\}$ and $\ell \in \{0,\ldots,s\}$. Then,

$$A[i,\ell] = \sum_{j=1}^{n} A\left[i-1, L^{-1}(\ell,j)\right],$$

where $A\left[i-1, L^{-1}(\ell,j)\right]$ is set to 0 if $L^{-1}(\ell,j)$ is the special symbol. The general dynamic programming algorithm is presented in Algorithm 1.

```
Algorithm 1: Given data $\mathbf{z} = (z_1,\ldots,z_n) \in \mathbb{N}^n_{>0}$ and error $\epsilon > 0$, determine $\mathcal{H}(\cdot)$.

    Step 1: Choose $Q_0,\ldots,Q_s$ such that $Q_0 = 1$, $\frac{Q_{\ell+1}}{Q_\ell} \leq 1 + \frac{\log(1+\epsilon)}{(n+1)}$, and $Q_s \geq nz_{(n)}$.
            Compute $L^{-1}(\ell,j)$ for each $\ell \in \{0,\ldots,s\}$ and $j \in \{1,\ldots,n\}$.

    Step 2: For all $\ell \in \{0,\ldots,s\}$, $A[0,\ell] \leftarrow 1$.
            For all $i \in \{1,\ldots,n\}$ and $\ell \in \{0,\ldots,s\}$, $A[i,\ell] \leftarrow \sum_{j=1}^{n} A[i-1, L^{-1}(\ell,j)]$.

    Step 3: Return the function $\mathcal{H}(\cdot)$, where $\mathcal{H}(\beta)$ is computed by a binary search over
            $A[n,0],\ldots,A[n,s]$ for any $\beta \in (0,1)$.
```

The algorithm as stated is not fully specified, as there are many possibly choices of s and $Q_0,\ldots,Q_s$, In the following section, we specify s and present and analyze an explicit construction of $Q_0,\ldots,Q_s$.

3.2 Bit complexity of approximation algorithm

In this section, we analyze the bit complexity of the proposed approximation algorithm.

We start by analyzing the bit complexity of Step 2. The values of $A[i,\ell]$ can be as large as $n^n = 2^{n \log_2 n}$, since $A[n,s] = n^n$. Thus, each $A[i,\ell]$ requires $O(n \log n)$ bits to be represented exactly. Computing each $A[i,\ell]$ requires summing n $O(n \log n)$-bit numbers, which requires a total of $O(n^2 \log_2 n)$ bit operations. Therefore, Step 2 requires $O(sn^3 \log n)$ bit operations.

In order to analyze the bit complexity of Step 1, we first describe how to construct a sequence $Q_0, \ldots, Q_s$ that meets the necessary requirements. We begin with a review of binary representation of integers. Suppose x is an non-negative integer. When stored as a binary value with b bits, x has the form $(x_{b-1}x_{b-2}\cdots x_1 x_0)$, where each $x_i \in \{0,1\}$ and $x = \sum_{i=0}^{b-1} 2^i x_i$. We note that the number of bits b must be greater than or equal to $\exp(x) \coloneqq \lfloor \log_2 x \rfloor$. To reduce the number of bits, x can be approximated as a floating-point value $\langle x \rangle_m$, where

$$\langle x \rangle_m \coloneqq \sum_{i=\exp(x)-m+1}^{\exp(x)} 2^i x_i.$$

Intuitively, $\langle x \rangle_m$ is the m most significant bits of x with the remaining bits truncated off. If x is a b-bit integer, then $\langle x \rangle_m$ requires $m-1$ bits to store $(x_{\exp(x)-1}\cdots x_{\exp(x)-m})$ ($x_{\exp(x)}$ always equals 1, and thus does not need to be stored) as well as $\lfloor \log_2 b \rfloor$ bits to store the value of $\exp(x)$. It is readily observed that

$$\left(1 - 2^{-m}\right) x \le \langle x \rangle_m \le x. \tag{10}$$

Given two floating-point numbers x_1, x_2 with m_1 and m_2 significant bits, we assume that they can be compared and $\langle x_1 + x_2 \rangle_{m_1}$ can be computed in bit complexity $O(m_1 + m_2 + \exp(x) + \exp(y))$.

We now describe how to construct $Q_0, \ldots, Q_s$. Define the following constants:

$$t \coloneqq \left\lceil \log_2 \left(\frac{n+1}{\log(1+\epsilon)} \right) \right\rceil \tag{11}$$

$$s \coloneqq \left\lceil 1 + \log_{1+2^{-t}} \left(n z_{(n)} \right) \right\rceil \tag{12}$$

$$m \coloneqq \lceil 1 + \log_2 s + t \rceil \tag{13}$$

For each $\ell \in \{0, \ldots, s\}$, let

$$Q_\ell \coloneqq \begin{cases} 1, & \text{if } \ell = 0, \\ \left\langle \left(1 + 2^{-t}\right) Q_{\ell-1} \right\rangle_m, & \text{if } \ell \in \{1, \ldots, s\}. \end{cases} \tag{14}$$

We now argue that the construction of $Q_0, \ldots, Q_s$ from Equation (14) satisfies the desired properties. It holds from definition that $Q_0 = 1$. It remains to show the other two properties.

Lemma 3.3. *Let $Q_0, \ldots, Q_s$ be defined as in Equation (14). Then,*

1. $\frac{Q_{\ell+1}}{Q_\ell} \le 1 + \frac{\log(1+\epsilon)}{n+1}$ *for all ℓ.*
2. $Q_s \ge n z_{(n)}$.

Proof. Proof

1. We observe that

$$\frac{Q_{\ell+1}}{Q_\ell} \le 1 + 2^{-t} \le 1 + \frac{\log(1+\epsilon)}{n+1}.$$

The first inequality follows from Equations (10) and (14). The second inequality follows from the definition of t.

2. We observe that

$$(1+2^{-t})^{s-1} \geq (1+2^{-t})^{\log_{1+2^{-t}}\left(nz_{(n)}\right)} = nz_{(n)},$$

where the inequality follows from the definition of s. Thus,

$$Q_s \geq (1-2^{-m})^s (1+2^{-t})^s \geq nz_{(n)}(1-2^{-m})^s(1+2^{-t}).$$

It remains to show that that $(1-2^{-m})^s \geq (1+2^{-t})^{-1}$. First,

$$m \geq \log_2 s + t + 1 \geq \log_2(s(2^t+1)+1),$$

which implies that $2^m - 1 \geq s(2^t+1)$. Therefore,

$$\left(1-2^{-m}\right)^{s(2^t+1)} \geq \left(1-2^{-m}\right)^{2^m-1} > e^{-1} > (1+2^{-t})^{-(2^t+1)},$$

which proves that $(1-2^{-m})^s \geq (1+2^{-t})^{-1}$.

□

We now analyze the bit complexity of Step 1. We observe that $t = O(\log(n\epsilon^{-1}))$, $s = O(n\epsilon^{-1}\log(nz_{(n)}))$, and $m = O(\log(n\epsilon^{-1}) + \log\log z_{(n)})$. Storing each Q_ℓ requires m significant bits. Since Q_s is at least as large as $nz_{(n)}$, $O(\log\log nz_{(n)})$ additional bits are required to store the exponent. In total, each Q_ℓ is stored in $O(m + \log\log nz_{(n)}) = O(m)$ bits. Given $Q_{\ell-1}$, we calculate Q_ℓ as $\langle Q_{\ell-1} + 2^{-t}Q_{\ell-1}\rangle_m$, which requires $O(m)$ bit operations. Thus, $Q_0, \ldots, Q_s$ can be computed in $O(sm)$ bit operations.

Once $Q_0, \ldots, Q_s$ are obtained, Step 1 requires computing $L^{-1}(\ell, j)$ for each $\ell \in \{0, \ldots, s\}$ and $j \in \{1, \ldots, n\}$. In order to compute each $L^{-1}(\ell, j)$ efficiently, we first compute $\langle Q_\ell - z_j\rangle_m$ for each $\ell \in \{0, \ldots, s\}$ and $j \in \{1, \ldots, n\}$, which requires a total of $O(snm)$ bit operations. By construction, each Q_ℓ has m significant bits; thus, $Q_{\ell'} \leq Q_\ell - z_j$ if and only if $Q_{\ell'} \leq \langle Q_\ell - z_j\rangle_m$. Finally, for each $j \in \{1, \ldots, n\}$, we can compute $L^{-1}(\ell, j)$ by iterating from $\ell = 0$ to s. This requires $O(snm)$ bit operations as well. Therefore, computing $L^{-1}(\ell, j)$ for each $\ell \in \{0, \ldots, s\}$ and $j \in \{1, \ldots, n\}$ can be done in $O(snm)$ bit operations.

Combining the results of Step 1 and Step 2, we conclude that the total bit complexity of the proposed algorithm is $O(sn^3\log n + snm) = O\left(\frac{n^2}{\epsilon}\log(nz_{(n)})\left(n^2\log n + \log\epsilon^{-1} + \log\log z_{(n)}\right)\right) = \tilde{O}\left(\frac{n^4}{\epsilon}\log z_{(n)}\right)$.

4 An exact algorithm for bootstrap

In this section, we present a deterministic exact algorithm for computing the exact bootstrap quantiles $H(\beta)$, from which confidence intervals are obtained. Specifically, for any data set $\mathbf{z} = (z_1, \ldots, z_n) \in \mathbb{N}^n$, the algorithm calculates $H(\beta)$ for each β with a bit complexity of $\tilde{O}\left(n^2 z_{(n)}\right)$, where $z_{(n)}$ is the largest data point.

In Section 4.1, we describe our algorithm, which is based on polynomial multiplication. In Section 4.2, we analyze its bit complexity. In Section 6, we show that the algorithm can find exact bootstrap confidence intervals for over one thousand of data points in minutes.

4.1 An exact algorithm based on polynomial multiplication

Our method is motivated by the technique of Nesterov [2004] for counting the number of binary points $\mathbf{x} \in \{0,1\}^n$ that satisfy a single equality constraint $\mathbf{a}^T\mathbf{x} = b$. Specifically, Nesterov showed that the number of binary solutions was equal to the b-th coefficient of the polynomial $\prod_{i=1}^n(1+t^{a_i})$. We consider a similar polynomial representation of the exact bootstrap distribution for the sample mean, which we describe in the following result.

Theorem 4.1. *The ℓ-th coefficient of $P(t) := \left(\sum_{i=1}^n t^{z_i}\right)^n$ equals the number of bootstrap samples $\mathbf{z}^* \in \{z_1, \ldots, z_n\}^n$ for which $\sum_{i=1}^n z_i^* = \ell$.*

Proof. Proof For any $k \in \mathbb{N}$, let $\mathbf{c}_k$ be the coefficients of the polynomial $\left(\sum_{i=1}^n t^{z_i}\right)^k$, that is, $\left(\sum_{i=1}^n t^{z_i}\right)^k = \sum_{\ell \geq 0} c_{k,\ell} t^\ell$. We claim that for all $\ell \geq 0$,

$$c_{k,\ell} = \left| \left\{ \mathbf{z}^* \in \{z_1, \dots, z_n\}^k : \sum_{i=1}^k z_i^* = \ell \right\} \right|.$$

The claim follows from an induction argument. If $k = 1$, then $c_{1,\ell}$ is the ℓ-th coefficient of $\sum_{i=1}^n t^{z_i}$, which implies that $c_{1,\ell} = |\{z^* \in \{z_1, \dots, z_n\} : z_i = \ell\}|$. Next, assume the claim holds for all $k' = 1, \dots, k-1$. Then,

$$\begin{aligned} c_{k,\ell} &= \sum_{s \geq 0} (c_{1,s})(c_{k-1,\ell-s}) = \sum_{i=1}^n c_{k-1,\ell-z_i} = \sum_{i=1}^n \left| \left\{ \mathbf{z}^* \in \{z_1, \dots, z_n\}^{k-1} : \sum_{i=1}^{k-1} z_i^* = \ell - z_i \right\} \right| \\ &= \left| \left\{ \mathbf{z}^* \in \{z_1, \dots, z_n\}^k : \sum_{i=1}^k z_i^* = \ell \right\} \right|. \end{aligned}$$

The first equality follows from multiplying $\sum_{i=1}^n t^{z_i}$ with $(\sum_{i=1}^n t^{z_i})^{k-1}$, the second equality follows because $c_{1,s}$ equals the number of z_i that are equal to s, and the third equality follows from the induction hypothesis. Thus, the claim holds for all k, in particular $k = n$, which is what we wanted to show. $\square$

Given $z_1, \dots, z_n \in \mathbb{N}$, suppose we had an efficient algorithm for calculating the coefficients $\mathbf{c}$ of the polynomial $(\sum_{i=1}^n t^{z_i})^n$. Then, for any $\alpha \in \mathbb{R}$, $G(\alpha)$ can be computed directly from the coefficients. Indeed,

$$\begin{aligned} G(\alpha) &= \frac{1}{n^n} \left| \left\{ \mathbf{z}^* \in \{z_1, \dots, z_n\}^n : \frac{1}{n} \sum_{i=1}^n z_i^* \leq \alpha \right\} \right| \\ &= \frac{1}{n^n} \left| \left\{ \mathbf{z}^* \in \{z_1, \dots, z_n\}^n : \sum_{i=1}^n z_i^* \leq \lfloor n\alpha \rfloor \right\} \right| \quad \text{since } z_1, \dots, z_n \text{ are integral} \\ &= \frac{1}{n^n} \sum_{\ell=0}^{\lfloor n\alpha \rfloor} c_\ell. \end{aligned}$$

We can subsequently compute $H(\beta)$ by a binary search over $G(\alpha)$. All that remains is showing that $\mathbf{c}$ can be computed efficiently. One option is to use the FFT to convolve the polynomials directly, as described in Nesterov [2004], since the FFT can convolve two polynomials in $O(d \log d)$ arithmetic operations. However, in our case, the coefficients of the polynomials quickly become very large, making each arithmetic operation time consuming.

Instead, we propose a simple algorithm for computing the coefficients of $(\sum_{i=1}^n t^{z_i})^n$ based on Kronecker substitution, a technique for encoding the coefficients of polynomials in large integers [Kronecker, 1882].

Proposition 4.1 (Kronecker substitution)**.** Let $P(t) = \sum_{i=0}^d a_i t^i$ be a polynomial with nonnegative integer coefficients $a_0, \dots, a_d$. If each $a_i \leq 2^M$ for a known $M \in \mathbb{N}$, then the values of $a_0, \dots, a_d$ can be obtained from the binary representation of $P(2^M)$.

Indeed, if $a_0, \dots, a_d \leq 2^M$, then the binary representation of $P(2^M)$ contains at most $(d+1)M$ bits. By partitioning those bits into $d+1$ blocks of M bits, it is readily observed that the first block of M bits corresponds to a_0, the second block corresponds to a_1, and so on. For a detailed discussion of Kronecker substitution, we refer the interested reader to Gathen and Gerhard [2013], Harvey [2009], and the references therein.

In our case of bootstrap, we want to obtain the coefficients of the polynomial $P(t) := (\sum_{i=1}^n t^{z_i})^n$. In order to use Kronecker substitution, we must bound the largest coefficient of $P(t)$. We observe that the sum of the coefficients of $P(t)$ is equal to $P(1) = n^n$; hence, the value of each coefficient of $P(t)$ is at most n^n. Moreover, if $z_1 = \cdots = z_n$, then the nz_1-th coefficient of $P(t)$ is n^n, showing that the n^n bound is tight. Therefore, it follows from Kronecker substitution that the coefficients of $P(t)$ can be obtained from the binary representation of $P(2^{\lceil n \log_2 n \rceil})$. Our general algorithm is as follows:

Algorithm 2: Given $z_1, \dots, z_n \in \mathbb{N}$, compute the coefficients $c_0, \dots, c_{nz_{(n)}}$ of $\left(\sum_{i=1}^{n} t^{z_i}\right)^n$.

Step 1: Let $M \leftarrow \lceil n \log_2 n \rceil$, and compute $v \leftarrow \sum_{i=1}^{n} (2^M)^{z_i}$.

Step 2: Compute v^n.

Step 3: For each $i \in \{0, \dots, nz_{(n)}\}$, obtain the coefficient c_i from the i-th block of M bits in the binary representation of v^n, that is,

$$c_i \leftarrow \left\lfloor \frac{v^n}{2^{iM}} \right\rfloor \pmod{2^M}.$$

The proposed algorithm is simple to implement. Moreover, most of the computational burden is contained in the large integer multiplications of Step 2, for which many open-source and highly-optimized libraries are available, such as `GMP` [Granlund, 2017]. The implementation of the proposed algorithm, and discussions of its performance, are found in Section 6.2.

4.2 Bit complexity of exact algorithm

In this section, we analyze the bit complexity of the proposed exact algorithm.

We begin with Step 1. Computing $M = \lceil n \log_2 n \rceil$ is trivial, and computing v requires summing the integers $2^{Mz_1}, \dots, 2^{Mz_n}$. For each z_i, we can compute 2^{Mz_i} by left-shifting 1 by Mz_i bits, which requires a bit complexity of $O(Mz_{(n)})$. Adding two $O(b)$-bit integers requires $O(b)$ bit operations, hence computing $2^{Mz_1} + \cdots + 2^{Mz_n}$ has a bit complexity of $O(nMz_{(n)}) = O(n^2 z_{(n)} \log n)$. Since $v \leq 2^{(M+1)z_{(n)}}$, it follows that v is represented in $O(nz_{(n)} \log n)$ bits.

In Step 2, we calculate v^n using a standard recursive algorithm for exponentiation. Namely, we compute v^n as $\textsc{Exp}(v, n)$, where

$$\textsc{Exp}(v,k) \leftarrow \begin{cases} k, & \text{if } k = 1, \\ \textsc{Exp}\left(v, \frac{k}{2}\right)^2, & \text{if } k \geq 2 \text{ and } k \text{ is even}, \\ v * \textsc{Exp}\left(v, \frac{k-1}{2}\right)^2, & \text{if } k \geq 2 \text{ and } k \text{ is odd}. \end{cases}$$

Suppose v is a $O(b)$-bit integer, and let $T(b) = \Omega(b)$ denote the bit complexity of multiplying two $O(b)$-bit integers (where the $\Omega(b)$ bound trivially holds since every bit in the operands must be examined). Thus, $\textsc{Exp}(v, k)$ has a bit complexity of $O(\sum_{\ell=1}^{\lfloor \log_2 n \rfloor} T(2^\ell b)) = O(T(nb))$. In Step 2, v is represented in $O(nz_{(n)} \log n)$ bits. Thus, Step 2 requires a bit complexity of $O(T(n^2 z_{(n)} \log_2 n))$. Finally, we analyze Step 3. We can assume that v^n is represented as an array of bits, which can be indexed in a constant number of bit operations. Each bit of v^n is examined exactly once in Step 3; hence, Step 3 can be performed in $O(n^2 z_{(n)} \log n)$ bit operations.

Since $T(b) = \Omega(b)$, the total bit complexity of the proposed algorithm is determined by Step 2, which has a bit complexity of $O(T(n^2 z_{(n)} \log n))$. The algorithms of Schönhage and Strassen [1971] and Fürer [2009] perform integer multiplication algorithm with a bit complexity of $T(b) = \tilde{O}(b)$. Therefore, the proposed algorithm has a bit complexity of $\tilde{O}(n^2 z_{(n)})$.

5 Extensions

The proposed algorithms from Sections 3 and 4 readily extends to statistics beyond just the sample mean. In general, the proposed approaches can be directly applied to sample statistics of the form

$$\frac{1}{n} \sum_{i=1}^{n} f(z_i), \tag{15}$$

where $f : \mathbb{N} \to \mathbb{N}$ is any transformation of the data $z_1, \ldots, z_n \in \mathbb{N}$. This general form encompasses statistics such as the k-th raw sample moment, for which $f(\zeta) = \zeta^k$, which are useful for quantifying the spread of a distribution. For statistics of the form in (15), we define the exact bootstrap distribution $G_f(\alpha)$ and the exact bootstrap quantile $H_f(\beta)$ as

$$G_f(\alpha) \coloneqq \frac{1}{n^n} \left| \left\{ \mathbf{z}^* \in \{z_1, \ldots, z_n\}^n : \frac{1}{n} \sum_{i=1}^{n} f\left(z_i^*\right) \leq \alpha \right\} \right|,$$
$$H_f(\beta) \coloneqq \min \left\{ \alpha : G_f(\alpha) \geq \beta \right\}.$$

Theorem 5.1. *For all data sets $z_1, \ldots, z_n \in \mathbb{N}$, $f : \mathbb{N} \to \mathbb{N}$, and $\beta \in (0, 1)$, $H_f(\beta)$ can be computed exactly with a bit complexity of $\tilde{O}\left(n^2 \bar{f}\right)$, where $\bar{f} = \max\{f(z_1), \ldots, f(z_n)\}$. If it also holds that $f(z_1), \ldots, f(z_n) > 0$, then for all $\epsilon > 0$, a $(1 + \epsilon)$-factor approximation of $H_f(\beta)$ can be computed with a bit complexity of $\tilde{O}\left(\frac{n^4}{\epsilon} \log \bar{f}\right)$.*

Proof. Proof We observe that

$$G_f(\alpha) \coloneqq \frac{1}{n^n} \left| \left\{ \mathbf{z}^* \in \{f(z_1), \ldots, f(z_n)\}^n : \frac{1}{n} \sum_{i=1}^{n} z_i^* \leq \alpha \right\} \right|.$$

Hence, the desired algorithms are obtained by using the algorithms from Sections 3 and 4 on the data set $(f(z_1), \ldots, f(z_n))$.

□

6 Experiments

In this section, we empirically compare the proposed bootstrap algorithms to the traditional randomized algorithm. In Section 6.1, we compare the accuracy of the proposed and traditional algorithms. We find that the confidence intervals produced by the traditional randomized method can output confidence intervals that vary significantly between runs, whereas the proposed methods eliminate the randomization error entirely. In Section 6.2, we examine the empirical speed of the proposed algorithms. We find that the proposed algorithms can find deterministic confidence intervals in minutes for a wide range of data sets with several hundred (and in some cases over one thousand) data points, which are sizes commonly found in applications such as clinical trials.

6.1 Accuracy

We performed experiments to evaluate the accuracy of confidence intervals produced by the traditional randomized method and our proposed algorithms. To begin, we recall the randomized algorithm for the bootstrap method: the practitioner randomly generates B bootstrap samples, and calculates the mean $\hat{\mu}^{*,1}, \ldots, \hat{\mu}^{*,B}$ for each bootstrap sample. The exact bootstrap distribution $G(\alpha)$ is approximated as

$$\tilde{G}_B(\alpha) \coloneqq \frac{1}{B} \sum_{b=1}^{B} \mathbb{1}\left\{\hat{\mu}^{*,b} \leq \alpha\right\}. \tag{16}$$

and the exact bootstrap quantile $H(\beta)$ is approximated by

$$\tilde{H}_B(\beta) \coloneqq \min \left\{ \alpha : \tilde{G}_B(\alpha) \geq \beta \right\}. \tag{17}$$

The quantile $\tilde{H}_B(\beta)$ is fundamental to computing many types of bootstrap confidence intervals. For example, the percentile method produces a 95% confidence interval as $[H(0.025), H(0.975)]$.

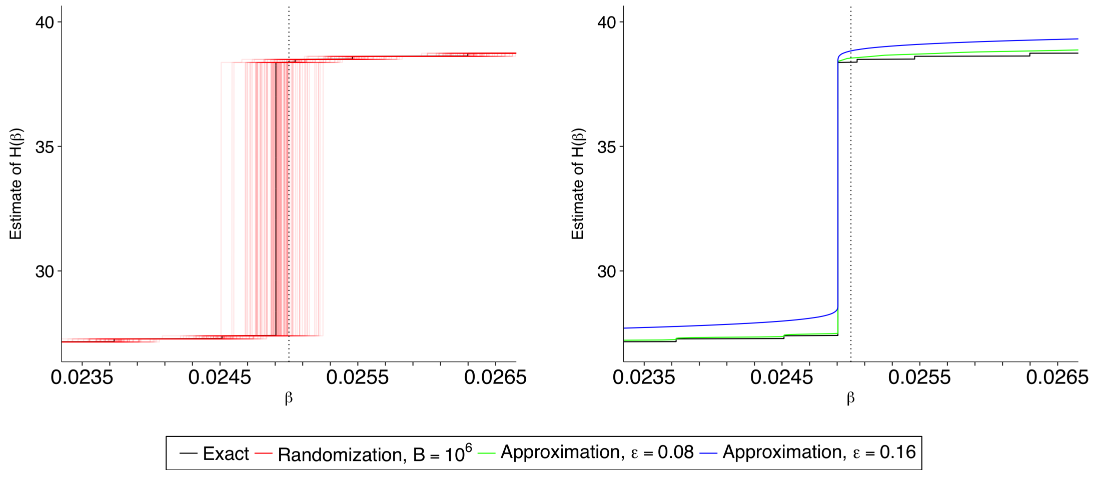

Figure 1. The estimates of $H(\cdot)$ using the exact algorithm (black), the traditional randomized method $\tilde{H}_B(\cdot)$ using $B = 10^6$ bootstrap samples from 100 separate runs (red), and the approximation algorithm $\mathcal{H}(\cdot)$ using $\epsilon = 0.08$ (green) and $\epsilon = 0.16$ (blue). The intersections with the vertical dotted line at $\beta = 0.025$ are the estimates of $H(0.025)$.

We first observe that $\tilde{G}_B(\cdot)$ converges to $G(\alpha)$ uniformly. Indeed, for any $\epsilon > 0$, the probability that $|\tilde{G}_B(\alpha) - G(\alpha)| > \epsilon$ decreases exponentially with B, via the Dvoretzky-Kiefer-Wolfowitz inequality [Dvoretzky et al., 1956, Massart, 1990]:

$$\mathbb{P}\left(\sup_{\alpha\in\mathbb{R}}|\tilde{G}_B(\alpha) - G(\alpha)| > \epsilon\right) \leq 2e^{-2B\epsilon^2},$$

Hence, the traditional randomized method often produces a good approximation for $G(\alpha)$. However, we find that the randomization error for quantiles can be quite significant, even for very large B. We illustrate via the following example. Consider a data set $\mathbf{z}$ of 81 elements where $\mathbf{z} = (1010, 1020, \ldots, 1070, 1, \ldots, 1)$. Suppose we are interested in obtaining a 95% confidence interval using the percentile method, which requires computing $H(0.025)$ and $H(0.975)$.

We performed experiments to compute $H(0.025)$ using the traditional randomized method, the proposed deterministic approximation algorithm from Section 3, and the proposed exact method from Section 4. First, we calculated $\tilde{H}_B(0.025)$ on 100 separate runs of the randomization method, each time using $B = 1$ million. That is, for each of the 100 runs, we randomly generated 1 million bootstrap samples and computed the sample mean for each bootstrap sample. Second, we calculated $\mathcal{H}(0.025)$ using the approximation algorithm from Section 3, using $\epsilon = 0.16$ and $\epsilon = 0.08$. Finally, we calculated $H(0.025)$ using the exact algorithm from Section 4.

The results are shown in Figure 1. The true value of $H(0.025)$, found from the exact method, is approximately 37.08. However, the values of $\tilde{H}_B(0.025)$ from the traditional randomized method varied substantially between the 100 separate runs. In particular, more than 25% of the 100 runs produced a $\tilde{H}_B(0.025)$ that was less than 25, which is incorrect by over 30%. This substantial variation is entirely attributed to the randomization. Note that 1 million is an extremely large choice for B, as B is typically chosen to be around 1,000 or 10,000. This example illustrates that there can be significant randomization error when using the traditional method, even when using a large B. In applications, such as in clinical trials or risk analysis, this error from randomization can have negative consequences, as the different confidence intervals may result in different health care and managerial decisions.

The proposed approximation algorithm $\mathcal{H}(\cdot)$, in contrast, involves no randomization and is deterministically close to the true $H(\beta)$, as observed in Figure 1. Moreover, the proposed approximation algorithm does not need the parameter B, alleviating the burden on the practitioner from needing to select and justify their choice of B. While we must choose ϵ, we always have a guarantee that $\mathcal{H}(\beta)$ is smaller than $(1+\epsilon)H(\beta)$, and we can run the algorithm with smaller and smaller ϵ if more accuracy is desired.

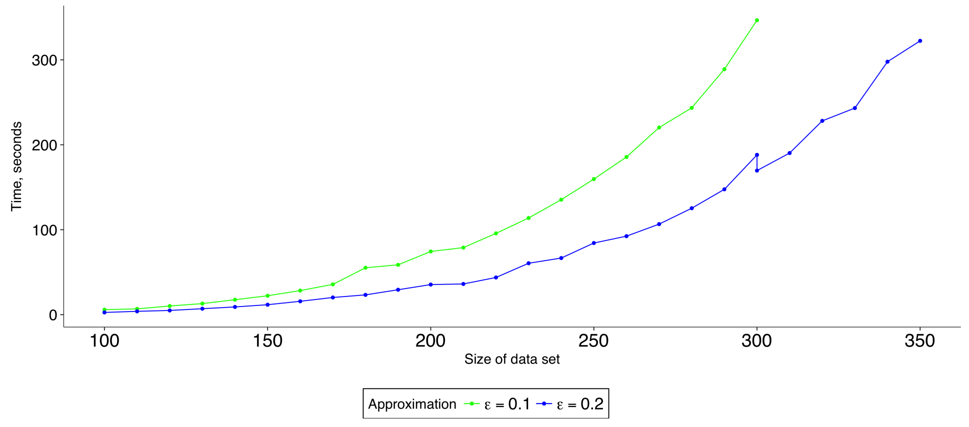

Figure 2. Each line shows the running time of the approximation algorithm from Section 3 with varying sized data sets $\mathbf{z}$, where each j-th data point is $z_j = \sqrt{j}$. The blue line corresponds to the approximation algorithm with $\epsilon = 0.2$. The orange line corresponds to the approximation algorithm with $\epsilon = 0.1$.

6.2 Tractability

In order to assess the real-world tractability of our proposed algorithms, we performed a sequence of experiments. We implemented the proposed approximation algorithm from Section 3 using the C++ programming language and the MPFR multiple-precision floating-point library [Fousse et al., 2007]. We implemented the proposed exact algorithm from Section 4 using the Julia programming language with the `BigInt` variable type for arbitrary-precision integers, which uses the `GMP` library [Granlund, 2017]. Finally, we also ran Barvinok's algorithm (described in Section 1.1) using the `LattE` implementation from De Loera et al. [2004] on the formulation of bootstrap as an integer counting problem, presented in Equation (3). All experiments were run on a 2.4 GHz Intel Core i5 processor.

First, we evaluated the speed of the approximation algorithm $\mathcal{H}(\beta)$ from Section 3. For varying values of n, we generated data sets of the form $\mathbf{z} = (\sqrt{1}, \ldots, \sqrt{n})$, with each value stored in 32 significant bits. We then ran the approximation algorithm for different values of ϵ. The results, shown in Figure 2, reveal that the proposed deterministic approximation algorithm runs in minutes on data sets of length up to 300. We note that these results are independent of the particular values of the data, as the running times did not change significantly for other data sets stored with 32 bits of accuracy. We conclude that the approximation algorithm is practical for any data sets in the 300s, such as those frequently found in real-world applications such as clinical trials and marketing. Importantly, the approximation algorithm is fast even if data values have many significant digits of accuracy.

Second, we ran the exact method from Section 4 on various data sets. To illustrate the impact of n and $z_{(n)}$ on the running time, we generated data sets of the form $\mathbf{z} = (z_1, \ldots, z_n)$, where $z_j = \left\lfloor \frac{j z_{(n)}}{n} \right\rfloor$ for varying sizes of n and $z_{(n)}$. The results in Figure 3 demonstrate the impact of the data values on the speed of the exact algorithm. When the data set consisted of integers with three significant digits (i.e., $z_{(n)} \leq 1000$), the proposed algorithm calculated the exact bootstrap distribution for over $n = 1200$ points in less than 5 minutes. For data sets consisting of integers with four significant digits, the algorithm runs with over $n = 400$ points in less than 5 minutes. These results show that, for data sets with only a few significant digits, the exact values of $H(\beta)$ can be computed in minutes for data sets with over 1000 data points.

Finally, Barvinok's algorithm scaled very slowly with the size of the data set. For the data set $\mathbf{z} = (1, 2, \ldots, 20)$, `LattE` took 466 seconds to count the number of integer points in the polyhedron defined in Equation (3), and took over an hour when $n = 30$. The reason is that our polyhedron has a number of constraints that scales linearly with the dimension of the polyhedron, as we have the constraints $0 \leq \gamma_{ij}$ and $\gamma_{ij} \leq 1$ for each variable.

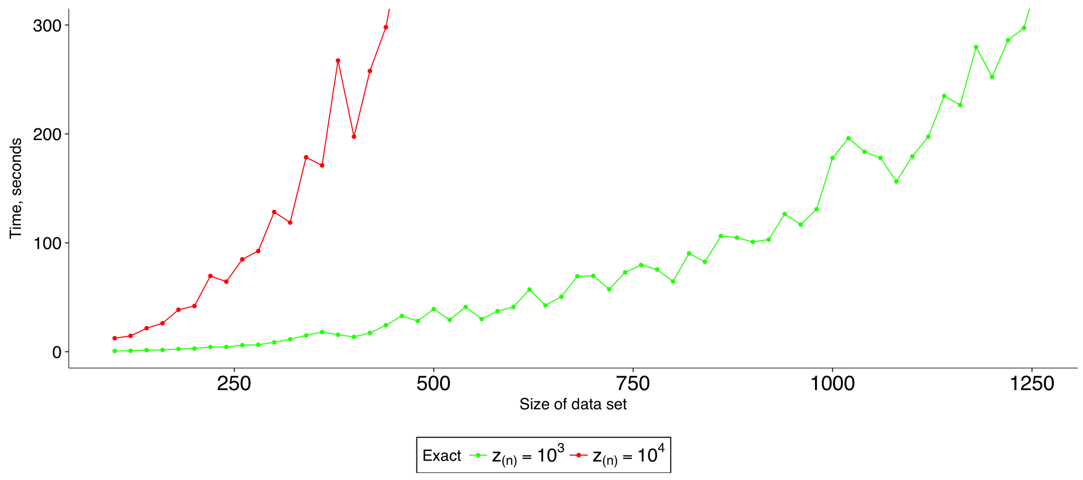

Figure 3. Each line shows the running time of the exact algorithm from Section 4 with varying sized data sets. The red line includes data sets with values ranging from 0 to 1,000. The blue line includes data sets with values ranging from 0 to 10,000. These correspond to data sets with 3 and 4 significant digits, respectively.

7 Conclusion

In this paper, we developed theoretical and empirical results for *if* and *when* deterministic bootstrap computations for the sample mean are computationally tractable. We presented several new complexity results, proposed an FPTAS and exact algorithm for the bootstrap, and demonstrated the practical significance and tractability of the proposed deterministic methods over the traditional randomized algorithms.

The proposed algorithms opens the door to deterministic techniques for the bootstrap method for a variety of sample statistics, beyond the sample mean and sample moments. Future research directions include designing efficient deterministic algorithms for other popular resampling methods that currently rely on randomization.

Acknowledgements

The authors thank Jim Orlin for helpful discussions and suggestions on the presentation of the complexity results.

References

Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, Cambridge, 1 edition, 2009.

Alexander I. Barvinok. A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. *Mathematics of Operations Research*, 19(4):769–779, 1994.

Dimitris Bertsimas and Robert Weismantel. *Optimization over integers*. Dynamic Ideas, Belmont, 2005.

Dimitris Bertsimas, Mac Johnson, and Nathan Kallus. The Power of Optimization Over Randomization in Designing Experiments Involving Small Samples. *Operations Research*, 63(4):868–876, 2015.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, volume 3. MIT press, Cambridge, 2009.

Jesús A. De Loera, Raymond Hemmecke, Jeremiah Tauzer, and Ruriko Yoshida. Effective lattice point counting in rational convex polytopes. *Journal of Symbolic Computation*, 38(4):1273–1302, 2004.

T DiCiccio and B Efron. More accurate confidence limits in exponential families. *Biometrika*, 79(2):231–245, 1992.

Thomas DiCiccio and Bradley Efron. Bootstrap Confidence Intervals. *Statistical Science*, 11(3):189–212, 1996.

Rick Durrett. *Probability: theory and examples.* Cambridge university press, 2010.

A. Dvoretzky, J. Kiefer, and J. Wolfowitz. Asymptotic Minimax Character of the Sample Distribution Function and of the Classical Multinomial Estimator. *The Annals of Mathematical Statistics*, 27:642—-669, 1956.

Martin Dyer. Approximate Counting by Dynamic Programming. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, pages 693–699, 2003.

Martin Dyer and Leen Stougie. Computational complexity of stochastic programming problems. *Mathematical Programming*, 106(3):423–432, 2006.

Martin Dyer, Alan Frieze, Ravi Kannan, Ajai Kapoor, Ljubomir Perkovic, and Umesh Vazirani. A mildly exponential time algorithm for approximating the number of solutions to a multidimensional knapsack problem. *Combinatorics, Probability & Computing*, 2:271–284, 1993.

Martin Dyer, Ravi Kannan, and John Mount. Sampling Contingency Tables. *Random Structures and Algorithms*, 10(4):487–506, 1997.

Martin E. Dyer and Alan M. Frieze. On the Complexity of Computing the Volume of a Polyhedron. *SIAM Journal on Computing*, 17(5):967–974, 1988.

Bradley Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

Bradley Efron. *The Jackknife, the Bootstrap and Other Resampling Plans.* SIAM, Philadelphia, 1982.

Bradley Efron. Better bootstrap confidence intervals. *Journal of the American statistical Association*, 82 (397):171–185, 1987.

Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap.* CRC press, New York, 1994.

Diane L. Evans, Lawrence M. Leemis, and John H. Drew. The Distribution of Order Statistics for Discrete Random Variables with Applications to Bootstrapping. *INFORMS Journal on Computing*, 18(1):19–30, 2006.

Nicholas I. Fisher and Peter Hall. Boostrap algorithms for small samples. *Journal of Statistical Planning and Inference*, 27:157–169, 1991.

Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. MPFR: A Multiple-Precision Binary Floating-Point Library with Correct Rounding. *ACM Transactions on Mathematical Software*, 33(2):13, 2007.

Martin Fürer. Faster integer multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2009.

Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, 1990.

Joachim Von Zur Gathen and Jurgen Gerhard. *Modern Computer Algebra.* Cambridge University Press, New York, 3rd edition, 2013. ISBN 1107039037, 9781107039032.

Parikshit Gopalan, Adam Klivans, Raghu Meka, Daniel Stefankovic, Santosh Vempala, and Eric Vigoda. An FPTAS for #knapsack and related counting problems. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 817–826. IEEE, 2011.

Torbjörn Granlund. The GNU Multiple Precision Arithmetic Library, 2017. URL http://gmplib.org.

Peter Hall. Theoretical comparison of Bootstrap Confidence Intervals. *The Annals of Statistics*, 16(3): 927–953, 1988.

Nir Halman. A deterministic fully polynomial time approximation scheme for counting integer knapsack solutions made easy. *Theoretical Computer Science*, 645:41–47, 2016.

Nir Halman, Diego Klabjan, Mohamed Mostagir, Jim Orlin, and David Simchi-Levi. A Fully Polynomial-Time Approximation Scheme for Single-Item Stochastic Inventory Control with Discrete Demand. *Mathematics of Operations Research*, 34(3):674–685, 2009.

Grani A. Hanasusanto, Daniel Kuhn, and Wolfram Wiesemann. A comment on "computational complexity of stochastic programming problems". *Mathematical Programming*, 159(1-2):557–569, 2016.

John M. Harris, Jeffry L. Hirst, and Michael J. Mossinghoff. *Combinatorics and Graph Theory*. Springer-Verlag, New York, 2 edition, 2008.

David Harvey. Faster polynomial multiplication via multipoint Kronecker substitution. *Journal of Symbolic Computation*, 44(10):1502–1510, 2009.

J.S. Huang. Efficient computation of the performance of bootstrap and jackknife estimators of the variance of L-statistics. *Journal of Statistical Computation and Simulation*, 38(1-4):45–56, 1991.

Alan D. Hutson and Michael D. Ernst. The Exact Bootstrap Mean and Variance of an L-Estimator. *Journal of the Royal Statistical Society. Series B (Methodological)*, 62(1):89–94, 2000.

Mark Jerrum and Alistair Sinclair. The Markov chain Monte Carlo method: an approach to approximate counting and integration. In D. S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 482–520. PWS Publishing., 1996.

Jon Kleinberg, Yuval Rabani, and Éva Tardos. Allocating Bandwidth for Bursty Connections. *SIAM J. Comput*, 30(1):191–215, 1997.

Leopold Kronecker. Grundzuge einer arithmetischen Theorie der algebraischen Grössen. *Journal fur die reine und angewandte Mathematik*, 1(92):1–122, 1882.

Jean-Bernard Lasserre. *Linear and Integer Programming vs Linear Integration and Counting*. Springer-Verlag, New York, 2009.

Jian Li and Tianlin Shi. A fully polynomial-time approximation scheme for approximating a sum of random variables. *Operations Research Letters*, 42(3):197–202, 2014.

Nathan Linial. Hard Enumeration Problems in Geometry and Combinatorics. *SIAM Journal on Algebraic Discrete Methods*, 7(2):331–335, 1986.

P. Massart. The Tight Constant in the Dvoretzky-Kiefer- Wolfowitz inequality. *The Annals of Probability*, 18(3):1269—-1283, 1990.

Yurii Nesterov. Fast Fourier Transform and its applications to integer knapsack problems. 2004.

A Schönhage and V Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7(3):281–292, sep 1971.

Daniel Štefankovic, Santosh Vempala, and Eric Vigoda. A deterministic polynomial-time approximation scheme for counting knapsack solutions. *SIAM Journal on Computing*, 41(2):356–366, 2012.

Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM Journal on Computing*, 8(3):410–421, 1979a.

Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979b.