Computation of Exact Bootstrap Confidence Intervals: Complexity and Deterministic Algorithms

Dimitris Bertsimas, Bradley Sturt

To cite this article:

Full terms and conditions of use: https://pubsonline.informs.org/Publications/Librarians-Portal/PubsOnLine-Terms-and-Conditions

This article may be used only for the purposes of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval, unless otherwise noted. For more information, contact permissions@informs.org.

The Publisher does not warrant or guarantee the article’s accuracy, completeness, merchantability, fitness for a particular purpose, or non-infringement. Descriptions of, or references to, products or publications, or inclusion of an advertisement in this article, neither constitutes nor implies a guarantee, endorsement, or support of claims made of that product, publication, or service.

Copyright © 2020, INFORMS

Please scroll down for article—it is on subsequent pages

With 12,500 members from nearly 90 countries, INFORMS is the largest international association of operations research (O.R.) and analytics professionals and students. INFORMS provides unique networking and learning opportunities for individual professionals, and organizations of all types and sizes, to better understand and use O.R. and analytics tools and methods to transform strategic visions and achieve better outcomes.
For more information on INFORMS, its publications, membership, or meetings visit http://www.informs.org
Methods

Computation of Exact Bootstrap Confidence Intervals: Complexity and Deterministic Algorithms

Dimitris Bertsimas, Bradley Sturt

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. In this paper, we present a new perspective on the bootstrap method through the lens of counting integer points in polyhedra. Through this new perspective, we make several advances for the bootstrap method, both theoretically and algorithmically. 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 (fully polynomial time approximation scheme) 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 on real and synthetic data sets with several hundreds of data points that the proposed deterministic algorithms can quickly produce confidence intervals that are substantially more accurate than those from randomized methods, and thus are practical alternatives in applications such as clinical trials.

Keywords: bootstrap method • counting problems • computational complexity • approximation algorithms • 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 \( \mu \). 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 practice, 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 by taking the 2.5% and 97.5% quantiles of the bootstrap distribution.

Despite common misconception, the algorithm described above is not the bootstrap method. Rather, it is a simulation approach that aims to approximate the exact bootstrap quantities, which are the results that we would obtain if we calculated all possible means generated from all possible bootstrap samples (Fisher and Hall 1991). For instance, conditioned on the historical data, the (exact) bootstrap distribution \( G(\alpha) \) of the sample mean is the proportion of all possible bootstrap samples that have mean less than or equal to \( \alpha \),

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

(1)

where \( \{z_1, \ldots, z_n\}^n \) is the set of all possible bootstrap samples of the data.\(^1\) From \( G(\alpha) \), we let \( H(\beta) \) denote the exact \( \beta \)th bootstrap quantile:

\[
H(\beta) := \min \{ \alpha : G(\alpha) \geq \beta \}.
\]

(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 comparison of bootstrap confidence intervals, we refer the reader to Hall (1988), Efron and Tibshirani (1994), DiCiccio and Efron (1996), and the references therein.

In contrast, the randomized approximation algorithm randomly samples $B$ 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

$$\hat{G}_B(\alpha) := \frac{1}{B} \sum_{b=1}^{B} \mathbb{1}\{\hat{\mu}^{*b} \leq \alpha\},$$  \hspace{1cm} (3)

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

$$\hat{H}_B(\beta) := \min\{\alpha : \hat{G}_B(\alpha) \geq \beta\}.$$  \hspace{1cm} (4)

This common method of using randomization to approximate $H(\beta)$ results in variability. As $B$ increases, the variability introduced by the randomization typically decreases. In practice, $B$ is always finite, and because randomness is involved, different results may be obtained each time that the algorithm is run. We define the randomization error as the difference between the approximate bootstrap quantile $\hat{H}_B(\beta)$ with finite $B$ and the exact bootstrap quantile $H(\beta)$.

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 posttreatment. 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 $B$ equals 1 million (see Section 6). In in examples such as clinical trials, poor estimates of the real effect of treatment are highly consequential, as they can potentially impact healthcare 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 values of $B$. 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, parallelized Monte Carlo simulation, and 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 approximate them with guaranteed bounds on the error. A deterministic method would remove uncertainty regarding randomization error and relieve practitioners from 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 (for example, $n \leq 9$), Fisher and Hall (1991) and Kisielinska (2013) propose methods for explicit enumeration of all possible bootstrap samples. For the iterated bootstrap method, where resampling is performed recursively, Chan and Lee (2001) propose an exact algorithm for small samples by constructing and analyzing a Markov chain. Huang (1991) and Hutson and Ernst (2000) discuss combinatorial and analytical approaches for calculating the exact bootstrap mean and variance for $L$ statistics using order statistics. Evans et al. (2006) propose a different method based on order statistics when the data fall 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 (Jerrum and Sinclair 1996, Bertsimas and Weismantel 2005, 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

1.1.1. 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 Garey and Johnson (1990) and Arora and Barak (2009).

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 $\{x \in \mathbb{R}^n : Ax \leq b\}$, 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 stochastic linear optimization problems (Dyer and Stougie 2006, Hanasusanto et al. 2016), computing the volume of a polyhedron (Dyer and Frieze 1988), and network reliability problems (Valiant 1979a). Examples from statistics include counting the exact number of $2 \times n$ contingency tables with specified column and row sums (Dyer et al. 1997).

1.1.2. Exact and Deterministic Approximation Algorithms. Given a polyhedron $\{x \in \mathbb{R}^n : Ax \leq b\}$ with $n$ variables and $m$ constraints, the integer counting problem asks for the number of integer points in the polyhedron. Because the integer counting problem is $\#P$-hard, no algorithm that is polynomial in $n$ and the size of $(A, b)$ 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 of which was presented in Barvinok (1994). Additional 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 $\{x \in \{0, 1\}^n : a^T x \leq b\}$ via the coefficients of the polynomial $\Pi_{i=1}^{n}(1 + t^a)$, 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 problem of counting binary vectors in a polyhedron with a single inequality constraint $\{x \in \mathbb{R}^n : a^T x \leq b\}$, was presented in the work of Dyer (2003), whose dynamic programming algorithm produced a $\sqrt{n} + 1$-factor approximation of $\#Knapsack$ in $O(n^2)$ time.

Štefankovic et al. (2012) proposed the first fully-polynomial time approximation scheme (FPTAS) for $\#Knapsack$ (see also Gopalan et al. 2011). A deterministic approximation algorithm is an FPTAS if, given any $\epsilon > 0$, it produces a solution with value that is within a $(1 + \epsilon)$ factor of the exact answer in time polynomial in the input size and $\epsilon^{-1}$. The algorithm of Štefankovic et al. (2012) has a bit complexity of $O(\frac{n^2}{\epsilon} \log(n) \log b)$ and is based on a dynamic programming formulation. This algorithm has also been extended to the integer variant of $\#Knapsack$ (Halman 2016) and the cumulative distribution function of the sum of nonidentical 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 leverage methodology from integer counting problems to develop 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 $H(\beta)$ is $\#P$-hard. To the best of our knowledge, these are the first 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 $\mathbb{P}(\sum_{i=1}^{n} X_i \leq \alpha)$ for independent and identically distributed (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 $H(\beta)$, via approaches similar to those in the work of Štefankovic et al. (2012) for knapsack counting problems. Specifically, for any data set of $n$ positive integers and $\epsilon > 0$, the algorithm produces a $(1+\epsilon)$-factor approximation of $H(\beta)$ with a bit complexity of $\tilde{O}(n^{2} \log z(\infty))$, where $z(\infty)$ is the largest data value.2 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 quantile $H(\beta)$ which has a bit complexity of $\tilde{O}(n^{2}z(\infty))$ 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 with those from the traditional randomized algorithm. First, we show using real and synthetic data sets that the confidence intervals produced using traditional methods have substantial error resulting from randomization, even when $B = 1$ million bootstrap samples are 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 of data points. This demonstrates that the proposed deterministic methods are practical alternatives to the traditional methods in applications such as clinical trials.

1.3. Structure
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 also show that computing $\mathbb{P}(\sum_{i=1}^{n} X_i \leq x)$ 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. Specifically, $G(\alpha)$ is equal to $\frac{1}{n} [P \cap \mathbb{Z}^n]$, where $P$ is the following polyhedron:

$$\left\{ \begin{array}{ll}
\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n} z_{ij} \gamma_{ij} \leq \alpha, \\
\gamma \in \mathbb{R}^{n \times n} : \\
\sum_{i=1}^{n} \gamma_{ij} = 1 & \text{for all } j = 1, \ldots, n, \\
\gamma_{ij} \geq 0 & \text{for all } i, j = 1, \ldots, n
\end{array} \right. \tag{5}$$

Indeed, the integer points $\gamma$ in the above set have a one-to-one mapping with the bootstrap samples $z^*$ that have mean less than or equal to $\alpha$. Specifically, each $\gamma$ corresponds to the bootstrap sample where the $j$th element $z^*_j$ in the bootstrap sample is equal to $z_i$ if $\gamma_{ij} = 1$.

2.1. Complexity of the Bootstrap Method
The complexity results in this section are based on the following lemma.

Lemma 1. Given $z \in \mathbb{N}^n$ and $\alpha \in \mathbb{N}$, computing

$$\left| \left\{ z^* \in \{z_1, \ldots, z_n\}^n : \sum_{i=1}^{n} z^*_i = \alpha \right\} \right|$$

exactly is $\#P$-hard.

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

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

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

$$\tilde{b} = nM^{n+1} + \sum_{i=1}^{n} M^i \mathbb{b}.$$
Consider the set \( \tilde{S}(a,b) \) defined as
\[
\tilde{S}(a,b) := \left\{ z^* \in \{ \tilde{a}_1, \ldots, \tilde{a}_{2n+1} \}^{2n+1} : \sum_{i=1}^{2n+1} z^*_i = b \right\}.
\]
\( \tilde{S}(a,b) \) is of the desired form in Lemma 1. In the remainder of the proof, we will show that \( |S(a,b)| = \frac{(n+1)!}{(2n+1)!} |\tilde{S}(a,b)| \), in which case the \#P-hard problem \( |S(a,b)| \) can be reduced in polynomial time to \( |\tilde{S}(a,b)| \).

- First, we show that \( |S(a,b)| \leq \frac{(n+1)!}{(2n+1)!} |\tilde{S}(a,b)| \). Consider any \( x \in S(a,b) \). Define \( z^* \in \mathbb{N}^{2n+1} \) as
\[
z^*_i = \begin{cases} 
\tilde{a}_i x_i + \tilde{a}_{i+n}(1-x_i) & \text{if } i \in \{1, \ldots, n\}, \\
\tilde{a}_{2n+1} & \text{if } i \in \{n+1, \ldots, 2n+1\}.
\end{cases}
\]
We assume without loss of generality that \( a_1, \ldots, a_n \) are strictly positive, in which case it follows from the definition of \( \tilde{a} \) and construction of \( z^* \) that \( z^*_1, \ldots, z^*_n \) take distinct values. Furthermore, it follows from the definition of \( \tilde{a} \) that \( \tilde{a}_i - \tilde{a}_{i+n} = a_i \) for each \( i \in \{1, \ldots, n\} \), which implies that
\[
\sum_{i=1}^{2n+1} z^*_i = \sum_{i=1}^{n} (\tilde{a}_i x_i + \tilde{a}_{i+n}(1-x_i)) = \sum_{i=1}^{n} \tilde{a}_i + \sum_{i=1}^{n} a_i x_i = b.
\]

We have thus shown that \( z^* \in \tilde{S}(a,b) \). The value of \( \sum_{i=1}^{2n+1} z^*_i \) is indifferent to the order of the elements; hence, each of the \( \frac{(2n+1)!}{(n+1)!} \) distinct permutations of \( z^* \) is also in \( \tilde{S}(a,b) \). This shows that \( |S(a,b)| \leq \frac{(n+1)!}{(2n+1)!} |\tilde{S}(a,b)| \).

- Second, we show that \( |S(a,b)| \geq \frac{(n+1)!}{(2n+1)!} |\tilde{S}(a,b)| \). Consider any element \( z^* \in \tilde{S}(a,b) \). Define \( y \in \mathbb{N}^{2n+1} \) such that \( y_i := \left\{ \left\lfloor \frac{b}{m} \right\rfloor \right\lfloor a_i \right\} \) for each \( i \in \{1, \ldots, 2n+1\} \). In words, \( y_i \) is the number of elements of \( z^* \) that are equal to \( \tilde{a}_i \). Then,
\[
\sum_{i=1}^{2n+1} y_i = \sum_{i=1}^{n} y_i + \sum_{i=1}^{n} y_i = \sum_{i=1}^{n} y_i = b,
\]
for \( z^* \) is in \( \tilde{S}(a,b) \). This shows that \( |S(a,b)| \geq \frac{(n+1)!}{(2n+1)!} |\tilde{S}(a,b)| \).

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

Using Lemma 1, we readily obtain the complexity of computing the exact bootstrap distribution \( G(a) \) for the sample mean.

**Theorem 1.** Computing \( G(a) \) exactly is \#P-hard.

**Proof.** For any \( z \in \mathbb{N}^n \) and \( a \in \mathbb{N} \), we can reduce (6) to \( G(a) \) as
\[
\left| \left\{ z^* \in \{z_1, \ldots, z_n\}^n : \sum_{i=1}^{n} z^*_i = a \right\} \right| = n^\alpha \left( \frac{G(a)}{\left\lfloor \frac{a}{m} \right\rfloor \left\lfloor a+1 \right\rfloor} \right).
\]
Next, we show the complexity of computing the exact bootstrap quantiles $H(\beta)$ for the sample mean.

**Theorem 2.** Computing $H(\beta)$ exactly is #P-hard.

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

### 2.2. Complexity of Probability

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

**Corollary 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.** Let $z_1, \ldots, z_n \in \mathbb{R}$ be distinct. Then, it follows directly from the definition of the exact bootstrap distribution of the sample mean (see Section 1) that $G(\alpha) = \mathbb{P}(\sum_{i=1}^n X_i \leq \alpha)$, where $X_1, \ldots, X_n$ are independent random variables which are 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 are for the sum of nonidentical Bernoulli random variables (Kleinberg et al. 1997, Halman et al. 2009). Specifically, it has been shown that $\mathbb{P}(\sum_{i=1}^n X_i \leq \alpha)$ is #P-hard when $X_1, \ldots, X_n$ are independent Bernoulli random variables with rates $p_1, \ldots, p_n$.

Interestingly, we can approximate $\mathbb{P}(\sum_{i=1}^n X_i \leq \alpha)$ 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 $\mu$ and standard deviation $\sigma$, then

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

Moreover, if $\mathbb{E}[|X_1 - \mu|^3] < +\infty$, 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 $Cn^{-1/2}\alpha^{-3}\mathbb{E}[|X_1 - \mu|^3]$ 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 $\tilde{O}(\frac{n}{\epsilon^2} \log z(n))$, where $z(n)$ is the largest data point. Our algorithm, which extends the techniques of Štefankovic et al. (2012) for #Knapsack, 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 the 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 assume 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 can 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 $z$ of positive integers, let $\gamma_i(\cdot)$ for $i = 1, \ldots, n$ be defined as

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

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

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

with a base case $\gamma_0$ 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

$$
\gamma_i(\alpha) = \sum_{j=1}^n \left|\{z^* \in \{z_1, \ldots, z_n\}^{i-1} : \sum_{j=1}^{i-1} z^*_j \leq \alpha - z_i\}\right|.
$$

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 $\alpha$. To describe the restricted set, we introduce some
terminology. Let $Q_0, \ldots, Q_s$ denote any sequence for which $Q_0 = 1$, $Q_{t+1}/Q_t \leq 1 + \log_2(e + 1)/(n + 1)$ for each $t$, and $Q_s \geq z(n)$. Intuitively, such a sequence behaves like a geometric progression over the range $[1,z(n)]$. Given any $\alpha \geq 1$, let $Q^{-1}(\alpha)$ be defined as the largest $Q_t$ for which $Q_t \leq \alpha$.

We now define approximations $\gamma_0, \ldots, \gamma_n$ of $\gamma_0, \ldots, \gamma_n$. Let $\gamma_0(\alpha) = \gamma_0(\alpha)$ for all $\alpha$. For $i \in \{1, \ldots, n\}$, let $\gamma_i$ be defined with a similar recursion to $\gamma_\ell$, with the distinction that each $\alpha$ is rounded down to the nearest $Q_t$:

$$\gamma_i(\alpha) := \sum_{j=1}^{n} \gamma_{i-1}(Q^{-1}(\alpha) - z_j).$$ (10)

For any $\alpha \in [Q_0, Q_s]$, $\gamma_i(\alpha) = \gamma_i(Q_t)$; thus, $\gamma_i(\alpha)$ is entirely specified by evaluation at $\alpha \in \{Q_0, \ldots, Q_s\}$. We claim that the functions $\gamma_1, \ldots, \gamma_n$ are indeed close approximations to $\gamma_1, \ldots, \gamma_n$, as formalized in the following lemma.

**Lemma 2.** For all $i \in \{0, \ldots, n\}$, $\gamma_i$ is nondecreasing, and for all $\alpha \in \mathbb{R}_+$, $\gamma_i(r^{-i}) \leq \gamma_i(\alpha) \leq \gamma_i(\alpha)$, for $r := 1 + \log_2(e + 1)/(n + 1)$.

**Proof.** The result follows from induction on $i$. If $i = 0$, then $\gamma_0 = \gamma_0$ implies that $\gamma_0$ is clearly nondecreasing and satisfies (11). If $i > 0$, then $\gamma_i$ is the sum of nondecreasing functions, which implies that it is nondecreasing. Next, we show that $\gamma_i$ satisfies (11) by showing the two sides of the inequality:

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

The first inequality is from the induction hypothesis. The second inequality follows, because $\gamma_{i-1}$ is nondecreasing. We now show the other side of the inequality:

$$\gamma_i(\alpha) \geq \sum_{j=1}^{n} \gamma_{i-1}(r^{-i} \alpha - z_j) \geq \sum_{j=1}^{n} \gamma_{i-1}(r^{-i} \alpha - z_j) r^{\ell(i-1)} \geq \sum_{j=1}^{n} \gamma_{i-1}(r^{-i} \alpha - z_j) = \gamma_i(r^{-i}).$$

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

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

$$\mathcal{H}(\beta) := \min\{\alpha : n^{-\beta} \gamma_n(\alpha n) \geq \beta\}.$$

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

**Lemma 3.** For all $e > 0$ and $\beta \in (0,1)$,

$$H(\beta) \leq \mathcal{H}(\beta) \leq (1 + e)H(\beta).$$ (12)

**Proof.** We show that each inequality holds. First, $H(\beta) = \min\{\alpha : n^{-\beta} \gamma_n(\alpha n) \geq \beta\} \leq \mathcal{H}(\beta)$, where the inequality follows from Lemma 2. Second, $\mathcal{H}(\beta) \leq \min\{\alpha : n^{-\beta} \gamma_n(\alpha n) \geq \beta\} = r^\beta H(\beta) \leq (1 + e)H(\beta)$, where $r = 1 + \log_2(e + 1)/(n + 1)$. The first inequality follows from Lemma 2, and the second inequality follows by the definition of $r$. □

We recall that $\gamma_i(\alpha) = \gamma_i(Q_t)$ for all $\alpha \in [Q_0, Q_s]$. Therefore, if we could efficiently obtain $\gamma_i(Q_t)$ for each $\ell \in \{0, \ldots, s\}$, then we can compute $\mathcal{H}(\beta)$ for any $\beta$ by a binary search over $\gamma_i(Q_0), \ldots, \gamma_i(Q_s)$.

We now describe an efficient algorithm to compute $\gamma_1, \ldots, \gamma_n$ based on dynamic programming. Let $L^{-1}(\ell, j)$ be defined as the largest index $\ell'$ for which $Q_{\ell'} \leq Q_{\ell} - z_j$. If $Q_{\ell} - z_j < 1$, then no such $\ell'$ exists, and $L^{-1}(\ell, j)$ returns a special symbol such as $-\infty$. Define $A$ as a two-dimensional array where $A[i, \ell] = \gamma_i(Q_\ell)$ for each $i \in \{0, \ldots, n\}$ and $\ell \in \{0, \ldots, s\}$. Then,

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

where $A[i-1, L^{-1}(\ell, j)]$ 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 positive integers $z = (z_1, \ldots, z_n)$ and error $e > 0$, determine $\mathcal{H}(\cdot)$.

1. Choose $Q_0, \ldots, Q_s$ such that $Q_0 = 1$, $Q_{t+1}/Q_t \leq 1 + \log_2(e + 1)/(n + 1)$, and $Q_s \geq z(n)$. Compute $L^{-1}(\ell, j)$ for each $\ell \in \{0, \ldots, s\}$ and $j \in \{1, \ldots, n\}$.

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

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 possibilities of $s$ and $Q_0, \ldots, Q_s$. For example, the requirements from Step 1 of Algorithm 1 can be satisfied by a geometric progression, where $Q_{t+1} = (1 + \log_2(e + 1)/(n + 1))$ and $s$ is the smallest integer that satisfies $(1 + \log_2(e + 1)/(n + 1))^s \geq z(n)$. However, this construction could require arbitrarily many bits to store the values of $Q_0, \ldots, Q_s$. Therefore, for
Algorithm 1 to be efficient, we must choose $Q_0, \ldots, Q_s$ such that their values can be represented compactly. The following section presents and analyzes an explicit construction of $Q_0, \ldots, Q_s$, which achieves these requirements.

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 $n^\ell = 2^{e \log n}$ since $A[n, s] = n^s$. 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 $O(n^2 \log n)$ bit operations. Therefore, Step 2 requires $O(sn^2 \log n)$ 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 that $x$ is a nonnegative integer. When stored as a binary value with $b$ bits, $x$ has the form $(x_b x_{b-1} \cdots x_1 x_0)$, 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 $\exp(x) := \lceil \log_2 x \rceil$. To reduce the number of bits, $x$ can be approximated as a floating-point value $(x)_m$, where

$$(x)_m := \frac{\exp(x)}{\exp(x) - m + 1} \cdot 2^m x_i.$$  

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

$$(1 - 2^{-m}) x \leq (x)_m \leq x.$$  

(13)

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 that $(x_1 + x_2)_m$ 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:

$$t := \log_2 \left( \frac{n + 1}{\log_2(1 + \epsilon)} \right)$$  

(14)

$$s := 1 + \log_{1 + 2^t}(nz_{(n)})$$  

(15)

$$m := 1 + \log_2 s + t.$$  

(16)

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

$$Q_\ell := \begin{cases} 1, & \text{if } \ell = 0, \\ ((1 + 2^{-t})Q_{\ell-1})_m, & \text{if } \ell \in \{1, \ldots, s\}. \end{cases}$$  

(17)

We now argue that the construction of $Q_0, \ldots, Q_s$ from (17) satisfies the desired properties. It holds from definition that $Q_0 = 1$. It remains to show the other two properties.

Lemma 4. Let $Q_0, \ldots, Q_s$ be defined as in (17). Then,

1. $Q_{\ell_1} \leq 1 + \log_2(1 + \epsilon) + \log_2(1 + \epsilon) = (1 + 2^{-t})Q_{\ell-1}$ for all $\ell = 0, \ldots, s - 1$, and
2. $Q_s \geq nz_{(n)}$.

Proof.

1. We observe that

$$\frac{Q_{\ell+1}}{Q_\ell} \leq 1 + 2^{-t} \leq 1 + \frac{\log_2(1 + \epsilon)}{n + 1}.$$  

The first inequality follows from (13) and (17). The second inequality follows from the definition of $t$.

2. We observe that

$$(1 + 2^{-t})^{s-1} \geq (1 + 2^{-t})^{\log_2(1 + \epsilon) + \log_2 s} = nz_{(n)}.$$  

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

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

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

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

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

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

We now analyze the bit complexity of Step 1. We observe that $t = O(\log(\log(\epsilon))$, $s = O(\log(\log(nz_{(n)})))$, and $m = O(\log(\log(\epsilon))) + \log_2(1 + \epsilon)$. Storing each $Q_\ell$ requires $m$ significant bits. Because $Q_s$ is at least as large as $nz_{(n)}$, $O(\log_2(1 + \epsilon))$ additional bits are required to store the exponent. In total, each $Q_\ell$ is stored in $O(m + \log n z_{(n)}) = O(m)$ bits. Given $Q_{(n-1)}$, we calculate $Q_\ell$ as $(Q_{(n-2)} Q_{(n-1)})_m$, which requires $O(m)$ bit operations. Thus, $Q_0, \ldots, Q_s$ can be computed in $O(sm)$ bit operations.

After $Q_0, \ldots, Q_s$ 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 $(Q_\ell - j)_m$ for each $\ell \in \{0, \ldots, s\}$ and $j \in \{1, \ldots, n\}$, which requires a total of $O(sm)$ bit operations. By construction, each $Q_\ell$ has $m$ significant
bits; thus, $Q_{\ell} \leq Q_{\ell} - z_j$ if and only if $Q_{\ell} \leq (Q_{\ell} - z_j)_m$.
Finally, for each $j \in \{1, \ldots, n\}$, we can compute $L^{-1}(\ell, j)$ by iterating from $\ell = 0$ to $s$. This requires $O(sn^m)$ 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 $O(sn^m)$ bit operations.

Combining the results of Steps 1 and 2, we conclude that the total bit complexity of the proposed algorithm is

$$O(sn^3 \log n + snm)$$

$$= O\left(\frac{n^2}{\epsilon} \log(nz_{(n)}) (n^2 \log n + \log e^{-1} + \log \log z_{(n)})\right)$$

$$= O\left(\frac{n^4}{\epsilon} \log z_{(n)}\right).$$

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 $z = (z_1, \ldots, z_n) \in \mathbb{N}^n$, the algorithm calculates $H(\beta)$ for each $\beta$ with a bit complexity of $O(n^2 z_{(n)})$, where $z_{(n)}$ is the largest data point. Our algorithm is based on the algorithm of Nesterov (2004) for #Knapsack as well as a technique of Kronecker (1882) for encoding polynomials as integers.

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 1,000 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 $x \in \{0, 1\}^n$ that satisfy a single equality constraint $a^T x = b$. Specifically, Nesterov (2004) 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 3.** The $\ell$th coefficient of $P(t) := (\sum_{i=1}^n t^{a_i})^n$ equals the number of bootstrap samples $z^* \in \{z_1, \ldots, z_n\}^n$ for which $\sum_{i=1}^n z_{\beta(i)}^* = \ell$.

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

$$c_{k, \ell} = \left\{ z^* \in \{z_1, \ldots, z_n\}^n : \sum_{i=1}^n z_{\beta(i)}^* = \ell \right\}.$$

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

$$c_{k, \ell} = \sum_{s \geq 0} (c_{1, s})(c_{k-1, \ell-s}) = \sum_{s \geq 0} c_{k-1, \ell-s},$$

$$= \sum_{i=1}^n \left\{ z^* \in \{z_1, \ldots, z_n\} : \sum_{i=1}^{k-1} z_{\beta(i)}^* = \ell - z_s \right\},$$

$$= \left\{ z^* \in \{z_1, \ldots, z_n\}^n : \sum_{i=1}^{k-1} z_{\beta(i)}^* = \ell, z_s = \ell - z_s \right\}.$$
Indeed, if \( a_0, \ldots, a_d \leq 2^M \), 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 Harvey (2009), Gathen and Gerhard (2013), and the references therein.

In our case of bootstrap, we want to obtain the coefficients of the polynomial \( P(t) := (\sum_{i=1}^n F_i)^n \). 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 \( P(2^{n\log_2 n}) \). Our general algorithm is as follows.

**Algorithm 2.** Given \( z_1, \ldots, z_n \in \mathbb{N} \), compute the coefficients \( c_0, \ldots, c_{nz_n} \) of \( (\sum_{i=1}^n F_i)^n \).

1. Let \( M \leftarrow \lceil n \log_2 n \rceil \), and compute \( v \leftarrow \sum_{i=1}^n (2M)^i \).
2. Compute \( v^n \).
3. For each \( i \in \{0, \ldots, nz_n \} \), obtain the coefficient \( c_i \) from the \( i \)th block of \( M \) bits in the binary representation of \( v^n \); that is,

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

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 6.3.

### 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 \( 2^{M_1}, \ldots, 2^{M_n} \). For each \( z_i \), we can compute \( 2^{M_i} \) by left shifting 1 by \( M_i \) bits, which requires a bit complexity of \( O(Mz_i) \). Adding two \( O(b) \)-bit integers requires \( O(b) \) bit operations; hence, computing \( 2^{M_1} + \cdots + 2^{M_n} \) has a bit complexity of \( O(nMz_n) = O(n^2z_n \log n) \).

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 \( \text{Exp}(v, n) \), where

\[
\text{Exp}(v, k) \left\{ \begin{array}{ll}
k & \text{if } k = 1, \\
\text{Exp}(v, \frac{k}{2}), & \text{if } k \geq 2 \text{ and } k \text{ is even}, \\
\frac{v \cdot \text{Exp}(v, \frac{k-1}{2})}{2}, & \text{if } k \geq 2 \text{ and } k \text{ is odd}.
\end{array} \right.
\]

Suppose that \( v \) 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, \( \text{Exp}(v, k) \) has a bit complexity of \( O(\sum_{i=1}^{\lfloor \log_2 n \rfloor} T(2b)) = O(T(nb)) \). In Step 2, \( v \) is represented in \( O(nz_n \log n) \) bits. Thus, Step 2 requires a bit complexity of \( O(T(n^2z_n \log n)) \), and \( v^n \) is represented by \( O(n^2z_n \log n) \) bits.

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 \( O(n^2z_n \log n) \) bit operations.

Because \( T(b) = \Omega(b) \), the total bit complexity of the proposed algorithm is determined by Step 2, which has a bit complexity of \( O(T(n^2z_n \log n)) \). 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 \( O(n^2z_n \log n) \).

### 5. Extensions

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

\[
\frac{1}{n^d} \sum_{i=1}^n f(z_i),
\]

where \( f : \mathbb{N} \rightarrow \mathbb{N} \) 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 \( f(i) = i^k \), which are useful for quantifying the spread of a distribution. For statistics of the form in (18), we define the exact bootstrap distribution \( G_f(\alpha) \) and the exact bootstrap quantile \( H_f(\beta) \) as

\[
G_f(\alpha) := \left\{ f(z_1, \ldots, z_n)^\alpha : \frac{1}{n^d} \sum_{i=1}^n f(z_i) \leq \alpha \right\},
\]

\[
H_f(\beta) := \min\left\{ \alpha : G_f(\alpha) \geq \beta \right\}.
\]

**Theorem 4.** For all data sets \( z_1, \ldots, z_n \in \mathbb{N}, f : \mathbb{N} \rightarrow \mathbb{N}, \) and \( \beta \in (0, 1) \), \( H_f(\beta) \) can be computed exactly with a bit complexity of \( \tilde{O}(n^2 \bar{f}) \), 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 \( \tilde{O}(\frac{n^4}{\epsilon} \log f) \).

**Proof.** We observe that

\[
G_f(\alpha) := \frac{1}{n^n} \left| \left\{ z^* \in \{f(z_1), \ldots, f(z_n))^{\times n} : \frac{1}{n} \sum_{i=1}^{n} z^*_i \leq \alpha \right\} \right|
\]

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

6. Computational Experiments

In this section, we empirically compare the proposed deterministic bootstrap algorithms with the traditional randomized algorithm. In Sections 6.1 and 6.2, we compare the accuracy of the proposed and traditional algorithms on real and synthetic data sets. The results demonstrate that the confidence intervals produced by the traditional randomized method can vary significantly between runs, whereas the proposed methods eliminate the randomization error entirely. In Section 6.3, we examine the empirical speed of the proposed deterministic algorithms. These computational experiments show that the proposed algorithms can find deterministic confidence intervals in minutes for a wide range of data sets with several hundreds of data points, which are sizes commonly found in applications such as clinical trials.

The proposed approximation algorithm from Section 3 was implemented in C++ and the MPFR multiple precision floating point library (Fousse et al. 2007). The proposed exact algorithm from Section 4 and the traditional randomized algorithm were implemented using the Julia programming language with the BigInt variable type for arbitrary precision integers, which uses the GMP library (Granlund 2017). For comparison in Section 6.3, we 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, which is presented in (5).

6.1. Framingham Heart Study

We first compare the deterministic and traditional approaches on a blood pressure data set from the Framingham Heart Study. The data set consists of examination results of 4,240 anonymized patients, and it includes each patient’s recorded diastolic blood pressure (mmHg). Of these patients, 124 are on blood pressure medication, and a histogram of the diastolic blood pressure of these patients is shown in Figure 1. We remark that a sample size of 124 subjects is typical in many clinical trials.

For this data set of patients on blood pressure medication, we consider the task of constructing bootstrap confidence intervals for the average diastolic blood pressure. To this end, we computed the exact bootstrap quantiles \( H(\cdot) \) using the proposed exact bootstrap algorithm, \( H(\cdot)(\epsilon = 0.01) \) using the proposed deterministic approximation algorithm, and \( H_B(\cdot)(B = 100,000) \) using the traditional randomized algorithm. To assess the variability of the randomized algorithm, we ran the randomized algorithm 300 separate times.

The accuracy of the various approaches as a function of \( \beta \) is displayed in Figure 2. The computational experiments demonstrate that the accuracy of the randomized algorithm as an estimate of \( H(\cdot) \) depends significantly on the quantile of interest. When the quantile \( \beta \) is close to zero, the randomized estimates \( H_B(\beta) \) become highly variable. Thus, for the randomized algorithm to accurately estimate \( H(\beta) \) for small quantiles for this data set, much greater choices of \( B \) become necessary. In contrast, Figure 2 shows that the accuracy of the proposed exact and

![Figure 1. Histogram of Diastolic Blood Pressure of Patients in the Framingham Heart Study on Blood Pressure Medication](image-url)
approximation algorithms is independent of $\beta$. In other words, the proposed deterministic algorithms always provide reliable computations for the bootstrap quantiles regardless of $\beta$. As an additional benefit, the proposed deterministic algorithms remove the temptation of a practitioner to run the randomized algorithm many times until a desirable estimate is obtained.

6.2. A Pathological Example

In the previous example we observed that the randomized approach was fairly accurate for quantiles $\beta$ that are not too close to zero. A natural question is as follows: can the randomized method be used reliably for quantiles outside of a tail regime? In this section, we show that the answer is no. Specifically, we present a data set in which the randomized method produces highly variable confidence intervals, even for very large $B$ and moderate choices of $\beta$.

Let us consider a data set of 81 elements where $z = (1010, 1020, \ldots, 1070, 1, \ldots, 1)$, and suppose that we are interested in obtaining a 95% bootstrap confidence interval using the percentile method. The percentile method requires computing $H(0.025)$ and $H(0.975)$. Similar to the previous section, we perform computational experiments to estimate $H(0.025)$. We computed $H(0.025)$ using the exact algorithm from Section 4, $\hat{H}_B(0.025)$ ($c = 0.05$) using the deterministic approximation algorithm from Section 3, and $\tilde{H}_B(0.025)$ ($B = 1$ million) on 200 separate runs of the randomization method. That is, for each of the 200 runs, we randomly generated 1 million bootstrap samples and computed the sample mean for each bootstrap sample.

The results, shown in Figure 3, demonstrate the significant variability of the randomized method for computing this confidence interval. The value of $H(0.025)$, found from the exact method, is approximately 38.4. However, the values of $\hat{H}_B(0.025)$ from the traditional randomized method varied substantially between the 200 separate runs. In particular, more than 25% of the 100 runs produced a $\hat{H}_B(0.025)$ that was less than 28, which differs from the exact bootstrap quantile by over 25%. Note that 1 million is an extremely large choice for $B$, as $B$ is typically chosen to be around 1,000 or 10,000. The explanation for this variability is because of discontinuities in the exact bootstrap distribution of this data set at $\beta = 0.025$; see the appendix for details.

The examples from the previous and present sections illustrate that there can be significant randomization error when using the traditional method, even when using large $B$. In applications, such as clinical trials or risk analysis, this error from randomization can have negative consequences, as the different confidence intervals may result in different healthcare and managerial decisions. The algorithms proposed in this work, in contrast, involve no randomization and have deterministic guarantees on their accuracy. Moreover, the proposed algorithms do not need the parameter $B$, alleviating the burden of needing to select and justify their choice of $B$. Although the proposed deterministic approximation algorithm requires the parameter $c > 0$, we
always have a guarantee that $H(\beta)$ is no less than $H(\beta)$ and no greater than $(1+\epsilon)H(\beta)$, and we can run this algorithm with smaller $\epsilon$ if more accuracy is desired.

### 6.3. Practicality of Deterministic Bootstrap Methods

To assess the practicality of the proposed deterministic algorithms, we performed experiments on a variety of data sets with varying sizes $n$ and magnitude $z(n)$. Specifically, for each $n \in \{20, 40, \ldots, 300\}$ and $z(n) \in \{10^3, 10^4, 10^5\}$, we randomly generated 100 data sets. Each of these data sets $z = (z_1, \ldots, z_n)$ was constructed with $z_1 = 1$ and $z_n = z(n)$, and the remaining $z_i$ were drawn independently, distributed uniformly over $\{1, \ldots, z(n)\}$. For each of these data sets, we ran the proposed exact algorithm and the deterministic approximation algorithm for $\epsilon = 0.05$ and 0.1.

The results of these experiments are presented in Figure 4. First, the results demonstrate that the proposed deterministic approximation algorithm can be solved in minutes for $n \leq 300$ and $\epsilon = 0.05$. Furthermore, the computation time of the deterministic approximation algorithm is relatively independent of $z(n)$. Thus, we conclude that the approximation algorithm is practical for any data set of size in the 300s, such as those frequently found in real-world applications, such as clinical trials and marketing. Second, the results show that the practicality of the proposed exact algorithm depends significantly on $z(n)$. On data sets with three and four significant digits ($z(n) = 10^3$ and $z(n) = 10^4$), the exact algorithm is faster than the deterministic approximation algorithm. However, on data sets with five significant digits ($z(n) = 10^5$) and moderate numbers of points ($n \geq 150$), the exact algorithm becomes impractical because of computer

**Figure 4.** Computation Times of the Proposed Exact and Deterministic Approximation Algorithms

*Notes.* The blue line is the deterministic approximation algorithm for $\epsilon = 0.1$, the red line is the deterministic approximation algorithm with $\epsilon = 0.05$, and the black line is the exact algorithm. The top, middle, and bottom panels correspond to $z(n) = 10^3$, $10^4$, and $10^5$, respectively. All computation times are averaged over 100 data sets, with the confidence band indicating the standard deviation. The dashed black line in the bottom plot signifies that the exact algorithm was unable to complete running on all 100 data sets owing to memory limitations (see Figure 5).
memory limitations. Indeed, the exact algorithm requires \( O(n^2 \log(n)) \) bits to encode the final output (see Section 4), which quickly exceeds the computer’s memory limitations to manipulate (see Figure 5). Therefore, we conclude that for data sets with few significant digits, the exact algorithm is highly practical and scales lightly in \( n \), and the deterministic approximation algorithm is otherwise preferable.

Finally, the computation time of Barvinok’s algorithm increased rapidly with the size of the data set. For the data set \( z = (1, 2, \ldots, 20) \), LattE took 466 seconds to count the number of integer points in the polyhedron defined in (5), and it took over an hour when \( n = 30 \). The reason is that our polyhedron has a number of constraints that scale linearly with the dimension of the polyhedron, as we have the constraints \( 0 \leq y' \) and \( y' \leq 1 \) for each variable.

7. Conclusion

In this paper, we presented a new perspective on the bootstrap method as a problem of counting integer points in a polyhedron. Through this connection, we leveraged advances from integer counting problems to obtain novel complexity results and the first practical deterministic algorithm for the bootstrap method. The proposed algorithms open 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.

Acknowledgments

The authors thank the associate editor and three reviewers for their thoughtful comments and suggestions as well as Prof. Jim Orlin for helpful discussions on the presentation of the complexity results, which have improved the paper significantly.

Appendix. Explanation of Section 6.2

Consider a data set of 81 entries defined as \( z = (1010, 1020, \ldots, 1070, 1, \ldots, 1) \). Let \( z' = (z'_1, \ldots, z'_81) \) denote a random bootstrap sample; hence, we can view \( z'_1, \ldots, z'_{81} \) as independent and identically distributed random variables where

\[
P(z'_1 = 1) = \frac{74}{81}, \quad \text{and} \quad P(z'_1 = 1010) = \cdots = P(z'_1 = 1070) = \frac{1}{81}.
\]

In Section 6.2, it is shown that \( \tilde{H}_8(0.025) \) has significant variability, even when \( B = 10^6 \). We prove in this section that this variability results from a discontinuity in the exact bootstrap distribution around the quantile \( \beta = 0.025 \). We begin the discussion with the following result.

Proposition A.1. For the data set constructed above, the random bootstrap sample has a distribution that satisfies

\[
P\left( \frac{1}{81} \sum_{i=1}^{81} z'_i \leq 27.5 \right) + P\left( \frac{1}{81} \sum_{i=1}^{81} z'_i \geq 38.4 \right) = 1
\]

and

\[
P\left( \frac{1}{81} \sum_{i=1}^{81} z'_i \leq 27.5 \right) \approx 0.0249055.
\]

Proof. For each \( i = 1, \ldots, 81 \), let \( X_i = \mathbb{I}\{z'_i \geq 1010\} \) be an indicator random variable, which equals 1 if \( z'_i \neq 1 \) and equals 0 otherwise. Then, it follows from the definition of the binomial distribution that

\[
P\left( \sum_{i=1}^{81} X_i \leq 2 \right) = \sum_{j=0}^{2} \binom{81}{j} \left( \frac{74}{81} \right)^j \left( \frac{7}{81} \right)^{81-j} \approx 0.0249055. \tag{A.1}
\]
We now consider two cases. First, if $\sum_{i=1}^{81} x_i \leq 2$, then the sample mean is bounded above by
\[
\frac{1}{81} \sum_{i=1}^{81} z_i^* \leq \frac{1 \times 79 + 1,070 \times 2}{81} \approx 27.4.
\]
Second, if $\sum_{i=1}^{81} x_i \geq 3$, then the sample mean is bounded below by
\[
\frac{1}{81} \sum_{i=1}^{81} z_i^* \geq \frac{1 \times 78 + 1,010 \times 3}{81} \approx 38.4.
\]
Therefore, we have shown that $\frac{1}{81} \sum_{i=1}^{81} z_i^* \leq 27.5$ if and only if $\sum_{i=1}^{81} x_i \leq 2$, and it is $\frac{1}{81} \sum_{i=1}^{81} z_i^* \geq 38.4$ otherwise. Combining this conclusion with (A.2), we obtain the desired results. □

**Proposition A.2.** When $B = 10^6$, $\mathbb{P}(H_B(0.025) \leq 27.5) \approx 0.2721$.

**Proof.** Let $z^1, \ldots, z^B$ be independent and identically distributed bootstrap samples, and let $\bar{\mu}^B = \frac{1}{B} \sum_{i=1}^{81} z_i^*$ be the corresponding means. For each $b = 1, \ldots, B$, let $Y_b = \varepsilon(\bar{\mu}^B \leq 27.5)$, in which case
\[
\nu := \mathbb{E}[Y_b] = \mathbb{P}(\bar{\mu}^B \leq 27.5) \approx 0.0249055,
\]
\[
\sigma := \sqrt{\text{Var}(Y_b)} \approx \sqrt{0.0249055(1 - 0.0249055)} 
\approx 0.155837.
\]
Therefore, it follows from the central limit theorem that
\[
\mathbb{P}(H_B(0.025) \leq 27.5)
= \mathbb{P}(\bar{G}_B(27.5) \geq 0.025)
= \mathbb{P} \left( \frac{\sum_{b=1}^{B} Y_b \geq 0.025}{\sqrt{B\sigma}} \right)
\approx \Phi \left( \frac{\sqrt{B}(0.025 - \nu)}{\sigma} \right).
\]
where $\Phi(\cdot)$ is the complement of the standard normal distribution. Therefore, when $B = 10^6$, the above expression with the above values of $\nu$ and $\sigma$ obtains the desired result. □

**Endnotes**

1 The collection of bootstrap samples $\{z_1, \ldots, z_n\}^n$ is defined rigorously as
\[
\{z_1, \ldots, z_n\}^n = \{(z_{k_1}, \ldots, z_{k_n}) : k_1, \ldots, k_n \in \{1, \ldots, n\}\}.
\]
For example, if $z = (1, 2, 2)$, then $\{1, 2, 2\}^3$ will contain one copy of $(1, 1, 1)$; two copies of $(1, 1, 2)$, $(1, 2, 1)$, and $(2, 1, 1)$; four copies of $(2, 2, 1)$, $(2, 1, 2)$, and $(1, 2, 2)$; and eight copies of $(2, 2, 2)$. Hence, there are exactly $n^n$ bootstrap samples in $\{z_1, \ldots, z_n\}^n$.

2 We use the notation that $f(n) = \tilde{O}(g(n))$ if $f(n) = O(g(n) \log^k g(n))$ for some $k$.

3 The Framingham data set can be found in the online companion of Bertsimas et al. (2016).

**References**


Dimitris Bertsimas is the Boeing Professor of Operations Research in the Sloan School of Management and codirector of the MIT Operations Research Center. His research interests include analytics and optimization with applications in many areas, including healthcare and transportation. He is a member of the National Academy of Engineering.

Bradley Sturt is a PhD candidate at the MIT Operations Research Center. His research interests include scalable algorithms for data-driven optimization and machine learning with applications in supply chain management, clinical trials, and finance.