Algorithmica (2015) 73:637–651
DOI 10.1007/s00453-015-0077-8

Exact Sublinear Binomial Sampling

Martín Farach-Colton[1] · **Meng-Tsung Tsai**[1]

Received: 31 January 2014 / Accepted: 5 October 2015 / Published online: 15 October 2015

Abstract Drawing a random variate from a given binomial distribution $B(n, p)$ is an important subroutine in many large-scale simulations. The naive algorithm takes $\mathcal{O}(n)$ time w.h.p. in the WordRAM model, which is too slow in many settings, though to its credit, it does not suffer from precision loss. The problem of sampling from a binomial distribution in sublinear time has been extensively studied and implemented in such packages as R [22] and the GNU Scientific Library [11], however, all previous sublinear-time algorithms involve precision loss, which introduces artifacts such as discontinuities into the sampling. In this paper, we present the first algorithm, to the best of our knowledge, that samples binomial distributions in sublinear time with no precision loss. We assume that each bit of p can be obtained in $\mathcal{O}(1)$ time.

Keywords Binomial distribution · Exact sampling · Sublinear sampling

1 Introduction

Let $B(n, p)$ be the binomial distribution of n trials and success rate p. Drawing a random variate b from $B(n, p)$ means that

$$\Pr[b = k] = p^k(1-p)^{n-k}\binom{n}{k} \quad \text{for all } k \in \{0, 1, \ldots, n\}. \tag{1}$$

This research was supported by NSF Grants IIS-1247750 and CCF-1114930.

✉ Meng-Tsung Tsai
mtsung.tsai@cs.rutgers.edu

Martín Farach-Colton
farach@cs.rutgers.edu

[1] Rutgers University, New Brunswick, NJ 08901, USA

The naive algorithm to accomplish such a sampling is to realize n Bernoulli trials of success rate p and count how many of them have a positive outcome in $\mathcal{O}(n)$ steps w.h.p. and $\mathcal{O}(n|p|)$ in the worst case, where $|p|$ is the number of bits in the representation of p. In other words, realizing n Bernoulli trials can be used as an alternative for binomial sampling.

Many applications rely on efficient implementations of binomial sampling. Examples include the efficient generation of random graphs from $\mathcal{G}(n, p)$ [3,4,20], logistic regression [10], generating virtual data sets in GLM analysis [1], generating random data or parameters [21] and speeding up Monte-Carlo simulation systems [25].

1.1 Previous Work

Several sublinear-time algorithms have been described [8,15,18], although all of these trade precision for speed. Algorithms BINV [15] and BG [7,15] both run in expected $\mathcal{O}(np)$ time. The former requires calculating $(1-p)^n$; the latter requires calculating the ratio of two logarithms. Algorithm BALIAS [15,19] requires calculating $\binom{n}{k}$ for all k in $\{0, 1, \ldots, n\}$ and constructing an alias table [19] based on the calculated values. The alias table can be constructed in $\mathcal{O}(n)$ time and then each variate generation can be computed in $\mathcal{O}(1)$ time on a machine with $\Omega(n)$ word size. Algorithm BTPE [15] divides the binomial distribution into parts and approximates each part by an upper-bounding function. To pick a variate at random, the algorithm samples a variate following the distribution composed of the upper bound functions and accepts it with a probability equal to the ratio of the binomial distribution and upper bound function. The procedure is repeated if the test fails. This method is known as the accept/reject rule, used e.g. in [8,15,18,25]. BTPE runs in sublinear time and is used by default in the statistical software R and GNU Scientific Library [11,22]. Because the distribution is divided piecewise and the piece is selected by an approximation to the true probability, BTPE does not exactly compute the binomial distribution. See [2,8,13,14,23] for more $\mathcal{O}(1)$-time algorithms in real computation model.

These algorithms run in sublinear time only if the precision of the calculations is truncated. When full precision is used, in calculating ratios, logarithms, or exponential functions, the time grows to linear or more. It is not clear how to modify them to be both accurate to full precision and sublinear. When these algorithms are implemented on a WordRAM, with, say, $\Theta(\log n)$-bit words, they accumulate inaccuracies that are caused by round-off error incurred in arithmetic operations. Thus, for example, common implementations of BTPE [15], in the GNU Scientific Library [11] and the statistical software R [22], perform such roundoff. Figure 2 shows that the distribution generated by BTPE has discontinuities. In particular, it substantially overestimates the probability of the tail of the distribution. In Fig. 2, the overestimation is by a factor of 2.59, or 0.74 % of the total samples. Thus, the occurrence of a rare event in BTPE cannot be trusted. The problems seen in BTPE are typical of any algorithm that fails to account for bounded precision, although the details will vary from algorithm to algorithm and implementation to implementation.

The discontinuity in the tail distribution of BTPE demonstrates the importance of an *exact sampling algorithm*. We call an algorithm that samples variate b with the exact

probability specified by Eq. 1 an exact sampling algorithm. Recently, efficient exact sampling algorithms for some distributions have been developed, e.g. for normal and geometric distributions [5,16].

1.2 Our Results

In this paper, we present what is, to the best of our knowledge, the **first sublinear-time algorithm** for drawing an **exact sample** from a binomial distribution in the WordRAM model, assuming the words have $\Omega(\log n)$ bits. In particular, we show that:

Theorem 1 *Given a positive integer n, one can sample a random variate from the binomial distribution $B(n, 1/2)$ in $\mathcal{O}(1)$ time with probability $1 - 1/n^{\Omega(1)}$ and in expectation after $\mathcal{O}(n^{1/2+\varepsilon})$-time preprocessing for any constant $\varepsilon > 0$, assuming that $\Omega(\log n)$ bits can be operated on in $\mathcal{O}(1)$ time. The preprocessing can be reused for any $n' = \mathcal{O}(n)$.*

Theorem 2 *Given an algorithm that can draw a sample from $B(n', 1/2)$ in $\mathcal{O}(f(n))$ time with probability $1 - 1/n^{\Omega(1)}$ and in expectation for any $n' \leq n$, then drawing a sample from $B(n, p)$ for any real p can be done in $\mathcal{O}(f(n) \log n)$ time with probability $1 - 1/n^{\Omega(1)}$ and in expectation, assuming each bit of p can be obtained in $\mathcal{O}(1)$ time.*

Since the publication of the conference version of this result [9], a new algorithm for generating $B(n, 1/2)$ without preprocessing was devised [6]. Their runtime is $\mathcal{O}(1)$ on average and polylog w.h.p. in the WordRAM model.[1] Our runtime is $\mathcal{O}(1)$ w.h.p. but needs $\mathcal{O}(n^{1/2+\varepsilon})$ preprocessing for any constant $\varepsilon > 0$.

In Sect. 4, we observe that it is possible to use their expected constant time algorithm to get a high-probability constant time sampling scheme at the cost of some preprocessing, by combining their algorithm with some of our ideas. We achieve the bounds stated in Theorem 3.

Theorem 3 *Given a positive integer n, one can sample a random variate from the binomial distribution $B(n, 1/2)$ in $\mathcal{O}(1)$ time with probability $1 - 1/n^{\Omega(1)}$ and in expectation after $\mathcal{O}(n^{\varepsilon})$-time preprocessing for any constant $\varepsilon > 0$, assuming that $\Omega(\log n)$ bits can be operated on in $\mathcal{O}(1)$ time. The preprocessing can be reused for any $n' = \mathcal{O}(n)$.*

Some papers [9,17] have assumed that it takes $\mathcal{O}(1)$ time to generate a single random bit, rather than $\log n$ random bits. In [17], an $\Omega(\log n)$ lower bound for sampling from $B(n, 1/2)$ was demonstrated under the random bit assumption. This bound is matched here by noting in the random bit model, $\log n$ random bits can be generated in $\mathcal{O}(\log n)$ time, followed by the binomial sampling step, which takes constant time.

[1] The exponent of the polylog is not specified in their paper, and depends on such factors as the runtime to compute the digits of $\sqrt{2\pi}$, but we note here that it is more than 1.

1.3 Organization

In Sect. 2, preliminary definitions and building blocks are introduced. In Sect. 3, we devise an algorithm to generate random variates from the binomial distribution $B(n, 1/2)$ with some preprocessing followed by showing the preprocessing can be reused for $B(n', 1/2)$ for $n' = \mathcal{O}(n)$. In Sect. 4, we present how to reduce the preprocessing from $\mathcal{O}(n^{1/2+\varepsilon})$ to $\mathcal{O}(n^{\varepsilon})$ by combining our algorithm with that of [6]. Then, in Sect. 5, we present a set of experiments to compare the algorithm used in the GNU Scientific Library with the proposed one.

2 Preliminaries

In this section, we prove Theorem 2 and revisit a data structure we will use in our algorithm. We analyze algorithms under the WordRAM model, in which it takes $\mathcal{O}(1)$ time to perform an arithmetic operation on two operands of w bits and also to generate a binary random integer of w bits, where w is assumed to be $\Omega(\log n)$.

We begin by reducing the computation of $B(n, p)$ to that of $B(n, 1/2)$. To see how, consider reducing $B(1, p)$ to $B(1, 1/2)$, that is, determining a single Bernoulli trial T with success rate $p = (0.a_1a_2\cdots)_2$, using a fair coin as follows. Start by comparing a fairly-generated random bit u with a_1. If $u < a_1$, T returns a positive outcome; if $u > a_1$, T returns a negative outcome; otherwise $u = a_1$, in which case proceed to the next bit and repeat the procedure. Then, $\mathcal{O}(1)$ comparisons are needed in expectation.

Proof of Theorem 2 A binomial variate b can be sampled from $B(n, p)$ by checking how many of n Bernoulli trials have a positive outcome. We can mimic the single Bernoulli trial procedure by replacing a comparison with p by a sequence of comparisons with a fair coin. The variate b is initialized to 0. Suppose we sample b_1 from $B(n, 1/2)$. That means that b_1 trials had value 0 at the first sampled bit and the remaining $n - b_1$ trials had value 1. If $a_1 = 1$, then $b = b + b_1$, because all b_1 trials are less than p no matter what the remaining sampled bits are. Having determined the outcome of b_1 Bernoulli trials, set $n = n - b_1$. If $a_1 = 0$, then $n = b_1$, because $n - b_1$ trials are greater than p. We repeat this procedure until $n = 0$, which takes $\mathcal{O}(\log n)$ rounds w.h.p. and in expectation because the probability that a trial remains undetermined after $c \log n$ rounds is $1/n^c$ for constant $c > 0$ and by union bound n will be reduced to 0 in $\mathcal{O}(\log n)$ rounds with probability $1 - 1/n^{\Omega(1)}$. □

Then, we revisit a data structure called *alias table*, also known as the alias-urn method in Devroye's book [8], which is a building block of the computation of $B(n, 1/2)$ introduced in Sect. 3.

2.1 Alias Table

Using an *alias table*, one can sample a random variate from a discrete distribution $\mathcal{D} = \{(i, p_i) : i \in [n]\}$ in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$-time preprocessing in the RealRAM model. Here we revisit the proof for the RealRAM model [19]. Then, in Sect. 3, we

will describe how to make the alias table work for the binomial distribution $B(n, 1/2)$ in the WordRAM model.

The idea of an alias table is to refine the distribution $\mathcal{D}$ so that each event becomes one or more events, but so that the new distribution can be sampled through a uniform distribution. Let

$$\mathcal{D}' = \left\{ (i, p_{ik}) : \sum_{k \in [n]} p_{ik} = p_i,\ p_{ik} \geq 0 \quad \text{for } i \in [n] \right\}$$

be a refinement of $\mathcal{D}$ such that for every j there are at most two non-zero p_{kj}'s and $\sum_k p_{kj} = 1/n$. Thus, one can construct a uniform distribution by setting

$$\hat{\mathcal{D}} = \left\{ (j, q_j) : q_j = \sum_{k \in [n]} p_{kj} = \frac{1}{n} \quad \text{for } j \in [n] \right\},$$

where for each j the probability q_j equals the average $1/n$ and at most two p_{kj} are non-zero, denoted by $p_{\alpha(j)j}$ and $p_{\beta(j)j}$. See Fig. 1 for an example.

Then, drawing a sample from $\mathcal{D}$ can be achieved by the following two steps. First, sample a random integer j from $[n]$ uniformly at random, which represents the selection of q_j in $\hat{\mathcal{D}}$. Second, select a random real number r from the range $[0, 1/n)$. Note that $p_{\alpha(j)j} + p_{\beta(j)j} = 1/n$. If $r < p_{\alpha(j)j}$, then say $\alpha(j)$ is drawn from $\mathcal{D}$; otherwise, say $\beta(j)$ is drawn. To sum up, a sample can be drawn by a random selection followed by a comparison, which takes $\mathcal{O}(1)$ time in the RealRAM model.

Such a distribution $\hat{\mathcal{D}}$ always exists and can be found by the following greedy process in $\mathcal{O}(n)$ time. Since $1/n$ is the average of n p_i's, there exists a $p_i \geq 1/n$. If $p_i = 1/n$, set $p_{in} = 1/n$, $p_{kn} = 0$ for $k \neq i$, and proceed greedily with the remainder. Otherwise, $p_i > 1/n$ and thus a $p_j < 1/n$ exists. Set $p_{jn} = p_j$, $p_{in} = 1/n - p_j$, $p_{kn} = 0$ for $k \neq i, k \neq j$ and replace p_i with $p_i - (1/n - p_j)$. Proceed greedily.

The greedy process is completed after n steps. Note that the selection of p_i and p_j takes $\mathcal{O}(1)$ time by maintaining two lists, one containing all $p_i \geq 1/n$ and the other containing the rest.

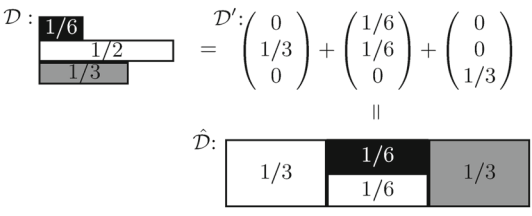

Fig. 1 The alias table of an instance $D = \{(1, 1/6), (2, 1/2), (3, 1/3)\}$

3 Exact Binomial Sampling

We now present how to sample a random variate from the binomial distribution $B(n, 1/2)$ in $\mathcal{O}(1)$ time with probability $1 - 1/n^{\Omega(1)}$ and in expectation. The main idea of our algorithm is to build an alias table for $\Theta(\log n)$-bit approximations of the probabilities $\Pr[b = k]$ ($= \binom{n}{k}/2^n$ in this section) and to perform a much slower algorithm for the residual probabilities, which are polynomially small. The binomial distribution only has $\mathcal{O}(\sqrt{n \log n})$ probabilities whose $\Theta(\log n)$-bit approximation is non-zero, so we can build a small alias table for those few non-zero $\Theta(\log n)$-bit approximations.

Here we assume that the $\Theta(\log n)$-bit approximations of $\Pr[b = k]$ for $k \in [0, n]$ can be obtained and defer the discussion for obtaining these bits. To sample a variate from the binomial distribution $B(n, 1/2)$ in $\mathcal{O}(1)$ time w.h.p. and in expectation, we employ an alias table. Below we describe the structure, properties and construction of the alias table.

3.1 Alias Table

S(n, c). By $S(n, c)$ for any constant $c > 0$ we denote such an alias table that can be applied to sample a random variate from $B(n, 1/2)$ in $\mathcal{O}(1)$ time with probability $1 - \mathcal{O}(1/n^c)$ and in expectation.

To adapt the alias table in Sect. 2 to the WordRAM model, we need to handle the precision issues. That is, the average $1/n$ and the probabilities p_i's might use many bits in their representations. In particular, we need $\Omega(n)$ bits to represent the probabilities in the binomial distribution $B(n, 1/2)$ because $\Pr[b = k]$ varies from $1/2^n$ to $\Theta(1/\sqrt{n})$.

To resolve the precision issue for the average $1/n$, we round n up to the nearest power of two by adding fewer than n dummy zero probabilities. Then $1/n$ becomes a power of two and requires $\mathcal{O}(\log n)$ bits to represent. To resolve the precision issue for the probabilities $\Pr[b = k]$, we construct an alias table on their $\Theta(\log n)$-bit approximations.

To sample exactly by an alias table only for $\Theta(\log n)$-bit approximations, we decompose each $\Pr[b = k]$ into a high probability part h_k and low probability part ℓ_k, where $\Pr[b = k] = h_k + \ell_k$, and then build an alias table on the $n + 1$ probabilities $h_0, h_1, \ldots, h_n$, and a failure event with probability

$$\ell^* = \sum_{0 \le k \le n} \ell_k.$$

Suppose each h_k is a multiple of 2^{-s} for some value s, then ℓ^* is also a multiple of 2^{-s}, which reduces the precision to represent these probabilities from n down to s. Then, drawing a sample from the alias table yields an event ℓ^* or some event h_k. If a sample yields event h_k, then return a variate k; otherwise, it is necessary to draw an event from the distribution $\mathcal{L} = \{(k, \ell_k/\ell^*)\}$, which requires computing the ℓ_k for $k \in [0, n]$. Given the full precision (n bits) of $\Pr[b = k]$, one can calculate the full precision of $\Pr[b = k + 1]$ by multiplying $\Pr[b = k]$ with $(n - k)$ and dividing it by $(k + 1)$. Note that $(n - k)$ and $(k + 1)$ have $\mathcal{O}(\log n)$ bits and an arithmetic operation of

that length takes $\mathcal{O}(1)$ time in our model. Therefore, the multiplication and the division takes $\mathcal{O}(n)$ time by long multiplication/division. In total, computing these ℓ_k's takes $\mathcal{O}(n^2)$ time. Then, drawing a sample from $\mathcal{L}$ takes $\mathcal{O}(n^2)$ time by comparing the prefix sums of the ℓ_k's with a random number. This step is time-consuming but rare.

Set $s = \Theta(\log n)$ and thus $\ell_k = 1/n^{\Omega(1)}$. Then, $\ell^* = 1/n^{\Omega(1)}$, a polynomially small value. In addition, only $\mathcal{O}(\sqrt{n \log n})$ h_k's are non-zero for such an s because $B(n, 1/2)$ is highly concentrated in the region $k \in [n/2 \pm \Theta(\sqrt{n \log n})]$ with probability $1 - 1/n^{\Omega(1)}$. As a result, we have the following lemma.

Lemma 4 *Sampling a random variate from $B(n, 1/2)$ takes $\mathcal{O}(1)$ time with probability $1-1/n^{\Omega(1)}$ and in expectation after $\mathcal{O}(n^{1/2+\varepsilon})$-time preprocessing for any constant $\varepsilon > 0$, assuming that the $\Theta(\log n)$-bit approximations of $\Pr[b = k]$ for $k \in [0, n]$ are given.*

Then, we show how to compute the $\Theta(\log n)$-bit approximations of $\Pr[b = k]$. The goal is to have a preprocessing step that runs in time and space $\mathcal{O}(n^{1/2+\varepsilon})$ for any constant $\varepsilon > 0$. Let h_s be the s-bit approximation of the probability $\Pr[b = k]$ such that

$$h_s + \ell_s = \binom{n}{k} 2^{-n},\ \ell_s = \mathcal{O}(2^{-s}) > 0 \quad \text{and } h_s \text{ is a multiple of } 2^{-s}.$$

To guarantee that a random variate can be generated from the binomial distribution $B(n, 1/2)$ in $\mathcal{O}(1)$ time with probability $1 - 1/n^{\Omega(1)}$, we construct $S(n, c)$ for any constant $c > 0$, $s = c \log n$. We obtain these s bits of $\Pr[b = k]$ for each $k \in [0, n]$ by two kinds of reduction.

3.2 Reducing the Calculation of $\Pr[b = k]$ to $\Pr[b = n/2]$

To simplify the notation, here we assume n is even. For odd n's, replacing $n/2$ with $\lfloor n/2 \rfloor$ suffices.

Lemma 5 *Given an algorithm that can obtain the s-bit approximation of the probability $\Pr[b = n/2]$ for any constant $c > 0$, $s = c \log n$, in $T(n)$ time, then one can compute the $c' \log n$-bit approximation of probabilities $\Pr[b = n/2 - k]$ for any constant $c' > 0$, $k \in [-\sigma, \sigma]$ in $T(n) + \mathcal{O}(\sigma)$ time.*

Proof Due to the symmetry of the binomial distribution, we only discuss the case for $k \in [1, \sigma]$. Suppose one has h_s for $\Pr[b = n/2]$, by definition

$$\Pr[b = n/2] - h_s = \ell_s > 0 \text{ and } \ell_s = \mathcal{O}(2^{-s}).$$

Let $\lfloor a \rfloor_s$ be the s-bit approximation of $a < 1$. Consider that

$$\begin{aligned}\Pr[b = n/2 - 1] &= \Pr[b = n/2](n/2)/(n/2+1) \\ &= (h_s + \ell_s)\delta_1 \qquad (\delta_t = (n/2+1-t)/(n/2+t) < 1 \text{ for } t \geq 1) \\ &= \lfloor h_s\delta_1 \rfloor_s + (h_s\delta_1 - \lfloor h_s\delta_1 \rfloor_s) + \ell_s\delta_1 \\ &= \lfloor h_s\delta_1 \rfloor_s + \mathcal{O}(1/2^s)\end{aligned}$$

Hence, the s-bit approximation of $\Pr[b = n/2 - 1]$ can be obtained in $\mathcal{O}(1)$ time (i.e. $\lfloor h_s\delta_1 \rfloor_s$) because the operands of the multiplication and division have $\Theta(\log n)$ bits. Iteratively, one has

$$\begin{aligned}\Pr[b = n/2 - k] &= \lfloor \lfloor \lfloor h_s\delta_1 \rfloor_s \delta_2 \rfloor_s \cdots \delta_k \rfloor_s + \mathcal{O}(k/2^s) \\ &= \lfloor \lfloor \lfloor h_s\delta_1 \rfloor_s \delta_2 \rfloor_s \cdots \delta_k \rfloor_s + \mathcal{O}(1/2^{s-\log n}) \qquad (k = \mathcal{O}(n))\end{aligned}$$

Since $T(n)$ suffices to compute h_s for any constant $c > 0$, $s = c \log n$, we are done by picking $c = c' + 1$. □

3.3 Reducing the Calculation of $\Pr[b = m]_{2m}$ to $\Pr[b = m/2]_m$

We extend the notation $\Pr[b = k]$ to $\Pr[b = k]_m$ where $\Pr[b = k]_m$ is defined to be $\binom{m}{k}/2^m$. Given the following identities, we have Lemma 6. Note that to make the recursion work, we also require the probability $\Pr[b = m]_{2m+1}$. This can be calculated by multiplying $\Pr[b = m]_{2m}$ with $(2m+1)/2m$.

$$\Pr[b = m]_{2m} = \sum_{k=0}^{m} \Pr[b = k]_m \Pr[b = m - k]_m \tag{2}$$

$$= 1/m^{\Omega(1)} + \sum_{k \in [m/2 \pm \Theta(\sqrt{m \log m})]} \Pr[b = k]_m \Pr[b = m - k]_m \tag{3}$$

Lemma 6 *Given an algorithm that can compute the s-bit approximation of the probability $\Pr[b = m/2]_m$ in $T(m)$ time for any constant $c > 0$, $s = c \log n$, then the s'-bit approximation of the probability $\Pr[b = m]_{2m}$ can be computed in $T(m) + \mathcal{O}(m)$ time for any constant $c' > 0$, $s' = c' \log n$ by Eq. (2). Furthermore, the s'-bit approximation of the probability $\Pr[b = m]_{2m}$ can be approximated in $T(m) + \mathcal{O}(m^{1/2+\varepsilon})$ time for any constant $\varepsilon > 0$ by bounding the precision loss by $1/m^{\Omega(1)}$ by Eq. (3).*

Proof By Lemma 5, all the terms on the right hand sides of Eqs. (2) and (3) can be obtained in $\mathcal{O}(m)$ time and in $\mathcal{O}(m^{1/2+\varepsilon})$ time, respectively. We are done by summing up the products. □

There is a tradeoff in the recursive computation for $\Pr[b = n/2]_n$. For $m \geq n^\delta$ for some constant $\delta \in (0, 1)$, one can use the approximation to obtain a speedup; however, for $m < n^\delta$, to retain sufficient precision, one needs to use the exact computation. Thus,

let $T(m)$ be the computation time for the s-bit approximation of $\Pr[b = m/2]_m$ for $s = c \log n$. We have the following recursion relation:

$$T(m) = \begin{cases} \mathcal{O}(m^{1/2+\varepsilon}) + T(m/2) & \text{if } m \geq n^{\delta}, \\ \mathcal{O}(m) + T(m/2) & \text{if } m < n^{\delta}. \end{cases}$$

Solving the relation by choosing $\delta = 1/2$, we have $T(n) = \mathcal{O}(n^{1/2+\varepsilon})$ for any constant $\varepsilon > 0$, as noted in Lemma 7.

Lemma 7 *The s-bit approximations of the probabilities $\Pr[b = k]_n$ for all $k \in [0, n]$ can be computed in $\mathcal{O}(n^{1/2+\varepsilon})$ time for any constant $\varepsilon, c > 0$ and where $s = c \log n$.*

Combining Lemma 7 and Lemma 4, one can generate a random variate from the binomial distribution $B(n, 1/2)$ in $\mathcal{O}(1)$ time with probability $1 - 1/n^{\Omega(1)}$ and in expectation.

In Theorem 2, one needs to sample variates from $B(n', 1/2)$, for some $n' \leq n$, in order to sample a variate from $B(n, p)$. If one prepares $S(n', c)$, for each $n' \leq n$, to make the theorem work, then the preprocessing requirement is $\mathcal{O}(n^{3/2+\varepsilon})$ rather than the claimed $\mathcal{O}(n^{1/2+\varepsilon})$. Here we introduce a reduction to resolve this issue.

3.4 Replacing $S(n', c)$ *for all* $n' \leq n$ with some $S(n_i, c(n_i))$

The goal is to preprocess $S(n_i, c(n_i))$ for a few n_i so that, for any $n' \leq n$, $S(n', c)$ can be replaced by those that have been preprocessed. For example, to sample a random variate from $S(n', c)$, for $n' = n_1 + n_2$, one can alternatively sample random variates from $S(n_1, c(n_1))$ and $S(n_2, c(n_2))$ and compute the sum of the variates, which follows the distribution $B(n', 1/2)$.

The probability that the alternative sampling procedure fails to generate a random variate is then, by the union bound,

$$1/n_1^{c(n_1)} + 1/n_2^{c(n_2)}.$$

To make the failure probability bounded by $\mathcal{O}(1/n^c)$, as claimed, we set $c(n_i)$ to be large enough so that $n_i^{c(n_i)} \geq n^c$, for $i \in \{1, 2\}$. To simplify the discussion, we set $c(n_i) = 4c$ if $n_i \geq n^{1/4}$ or set $c(n_i) = \infty$ otherwise. By $c(n_i) = \infty$ we denote that the underlying alias table contains the $c \log n$-bit approximations, for some constant $c > 0$ of all the probabilities $\Pr[b = k]$ for $k \in [0, n_i]$. The failure rate of sampling a random variate from $S(n_i, \infty)$ is thus bounded by $\mathcal{O}(1/n^c)$. As in the proofs of the space- and time-complexity of $S(n, c)$, one can show that the space usage of $S(n_i, \infty)$ and its construction time are both $\mathcal{O}(n_i \log n) = \mathcal{O}(n^{1/4+\varepsilon})$ for $n_i < n^{1/4}$.

We now show how to select the small set of n_i as follows so that for any $n' \leq n$, $S(n', c)$ can be replaced by a few $S(n_i, c(n_i))$.

3.4.1 Simple Selection

Given $S(n_i, c(n_i))$, for all $n_i \in \mathcal{R}_1$, such that $\mathcal{R}_1$ contains all the powers of two no more than n, one can replace $S(n', c)$, for any $n' \leq n$, with $\mathcal{O}(\log n)$ preprocessed $S(n_i, c(n_i))$ because every $n' \leq n$ can be represented by $\log n$ bits. The preprocessing of these $S(n_i, c(n_i))$ requires $\mathcal{O}(n^{1/2+\varepsilon})$ space and runtime. Though the preprocessing requirement now matches the claimed $\mathcal{O}(n^{1/2+\varepsilon})$, the runtime for sampling a random variate from $B(n', 1/2)$, for any $n' \leq n$, increases to $\mathcal{O}(\log n)$, because it is now implemented by sampling $\mathcal{O}(\log n)$ random variates. We consider an alternative selection as follows.

3.4.2 Refined Selection

Given $S(n_i, c(n_i))$, for all $n_i \in \mathcal{R}_2$, such that $\mathcal{R}_2$ contains all the square numbers no more than n, one can replace $S(n', c)$ for any $n' \leq n$ by $\mathcal{O}(1)$ preprocessed $S(n_i, c(n_i))$ because every natural number is a sum of four square numbers [24]. In this way, sampling a random variate from $B(n', 1/2)$ for any $n' \leq n$ is implemented by sampling $\mathcal{O}(1)$ random variates. Thus, the runtime is $\mathcal{O}(1)$ with probability $1 - \mathcal{O}(1/n^c)$ as claimed. However, the preprocessing increases to

$$\sum_{n_i \in \mathcal{R}_2, n_i < n^{1/4}} \mathcal{O}\left(n^{1/4+\varepsilon}\right) + \sum_{n_i \in \mathcal{R}_2, n_i \geq n^{1/4}} \mathcal{O}\left(n_i^{1/2+\varepsilon}\right) = \mathcal{O}\left(n^{1+\varepsilon}\right).$$

To reduce the preprocessing, one can prepare $S(n_i, c(n_i))$ for $n_i \in \mathcal{R}_k \cup \mathcal{R}'_k$, where k is a square number,

$$\mathcal{R}_k = \{r \leq k \mid r \in \mathcal{R}_2\} \text{ and } \mathcal{R}'_k = \{kr \mid r \in \mathcal{R}_2, r \leq n/k\}.$$

Then, the preprocessing reduces to $\mathcal{O}\left((k + n/\sqrt{k})\sqrt{\log n}\right)$ or $\mathcal{O}(n^{2/3+\varepsilon})$, if we choose $k = n^{2/3}$. In this way, each n' is decomposed into 8 square numbers in $\mathcal{R}_k \cup \mathcal{R}'_k$. Thus, the runtime still matches the claimed one. One can further reduce the preprocessing to $\mathcal{O}(n^{1/2+\varepsilon})$ by means of a decomposition into more groups of square numbers rather than only 2 groups. In the case of $t + 1 = \mathcal{O}(1)$ groups, the preprocessing reduces to

$$\mathcal{O}\left((k_1 + k_2/\sqrt{k_1} + \cdots + n/\sqrt{k_t})\sqrt{\log n}\right) = \mathcal{O}\left(n^{2^t/(2^{t+1}-1)}\sqrt{\log n}\right),$$

which is $\mathcal{O}(n^{1/2+\varepsilon})$ for sufficiently large constant t.

In the above reduction, we need to decompose an integer $n' \leq n$ into $\mathcal{O}(1)$ square numbers. We present how to do the decomposition in $\mathcal{O}(1)$ time with $\mathcal{O}(n^\varepsilon)$ preprocessing, for any constant $\varepsilon > 0$.

3.5 Decomposition into Square Numbers

We achieve this by table lookup. The table used to answer the decomposition of $n' \leq n$ into four square numbers can be precomputed in $\mathcal{O}(n^2)$ time by enumerating all sums of 4 square numbers. One can apply the idea used in the refined selection again. To decompose $n' \leq n$ into 8 square numbers, one first represents $n' = n'_1 + kn'_2$ where k is a square number, $n'_1 \leq k$ and $n'_2 \leq n/k$. It replaces the large table with two small ones that can answer the decomposition of $n' \leq k$ (resp. $\leq n/k$) into four square numbers. Choosing k to be $\sqrt{n}$, the preprocessing for the table drops to $\mathcal{O}(n)$, which can be further reduced to $\mathcal{O}(n^\varepsilon)$ for a small constant $\varepsilon > 0$ by means of a decomposition into more square numbers. If n' is decomposed into $4t$ square numbers, the total preprocessing of the t tables is $\mathcal{O}(tn^{1/t}) = \mathcal{O}(n^\varepsilon)$.

4 Polylogarithmic Preprocessing

Recall the algorithm of Bringmann et al. [6], which can sample $B(n, 1/2)$ in expected constant time. Here, we observe that this bound can be improved to constant time w.h.p. with a small amount of preprocessing. We first show that although one sample may take polylog time w.h.p., a batch of samples can be computed in amortized constant time w.h.p., if the batch is large enough.

Lemma 8 *Given an algorithm whose runtime is t^c with probability $\mathcal{O}(e^{-t})$ for $t > 0$, constant $c \geq 1$, then running the algorithm k times takes $\mathcal{O}(k)$ time with probability $1 - 1/e^{\Omega(k^{1/(c+1)})}$ and in expectation. Specifically, the runtime is $\mathcal{O}(k)$ with probability $1 - 1/n^{\Omega(1)}$ and in expectation if $k = \Omega(\log^{c+1} n)$.*

Proof The average runtime can be bounded by $\mathcal{O}\left(\sum_{t=1}^{\infty} t^c e^{-t}\right) = \mathcal{O}(1)$. By the central limit theorem, when k goes to infinity, the amortized runtime converges to the average $\mathcal{O}(1)$ almost surely. Here we show how small the k can be to make the runtime bounded by $\mathcal{O}(k)$ with probability $1 - 1/n^{\Omega(1)}$.

Let $X_1, X_2, \ldots, X_k$ be the runtime of the k runs. Let $I[E]$ be an indicator variable denoting whether the event E is true. Then, the total runtime of the k executions is

$$\sum_{i=1}^{k} X_i \leq \sum_{t=0}^{\infty} \left((t+1)^c - t^c\right) \sum_{i=1}^{k} I[X_i \geq t^c].$$

We analyze the right-hand side by splitting the summation into two parts, as follows.

$$\begin{aligned}\sum_{i=1}^{k} X_i \leq & \sum_{t \leq \delta \ln k} \left((t+1)^c - t^c\right) \sum_{i=1}^{k} I[X_i \geq t^c] \\ & + \sum_{t > \delta \ln k} \left((t+1)^c - t^c\right) \sum_{i=1}^{k} I[X_i \geq t^c]\end{aligned}$$

for some $\delta \in (0, 1)$. We now separately bound the contribution of each right-hand summation. Intuitively, the first summation is for the jobs that finish quickly, and thus contribute little to the running time, and the second summation is for the few jobs that take a long time.

Consider that the following equality

$$\Pr\left\{\sum_{i=1}^{k} I\left[X_i \geq t^c\right] - \Pr\left[X_i \geq t^c\right] = \Omega(k/e^t)\right\} = 1/e^{\Omega(k/e^t)} \tag{4}$$

holds for every t due to Chernoff bound. Thus, by the union bound the first summation is bounded by $\mathcal{O}(k)$ with probability $1 - 1/e^{\Omega(k^{1-\delta})}$, which is exponentially small in k, as desired.

For the second summation, we consider the probability of events E_1 and E_2,

- E_1 denotes that $X_i \leq k^\delta$ for all $i \leq k$,
- E_2 denotes that $\sum_{i=1}^{k} I[X_i \geq (\delta \ln k)^c] = \mathcal{O}(k^{1-\delta})$.

The failure probabilities of these two events are small. Precisely, we have

$$\Pr\left[\bar{E_1}\right] = 1/e^{\Omega(k^{\delta/c})} \text{ (by the union bound)}$$

and

$$\Pr\left[\bar{E_2}\right] = 1/e^{\Omega(k^{1-\delta})} \text{ (by Equality (4)).}$$

By the union bound, we know that $\Pr\left[\bar{E_1} \vee \bar{E_2}\right]$ is exponentially small in k. In other words, the second summation is bounded by $\mathcal{O}(k^\delta k^{1-\delta}) = \mathcal{O}(k)$ as desired, given $E_1 \wedge E_2$.

We complete the proof by choosing (k, δ) to be $(\log^{c+1} n, c/(c+1))$. This bounds the total runtime of the k executions by $\mathcal{O}(k)$ with polynomially small probability $1 - 1/n^{\Omega(1)}$. □

Therefore, we conclude:

Observation 9 *$B(n, 1/2)$ can be sampled in $\mathcal{O}(1)$ time with probability $1 - 1/n^{\Omega(1)}$ and in expectation, after preprocessing that takes polylogarithmic time w.h.p.*

Proof We change the expected runtime bound to the desired high-probability runtime by double buffering. In Lemma 8, we showed that $r = \log^{c+1} n$ samples can be computed by the algorithm in [6] in $d \log^{c+1} n$ time w.h.p. for some constant d. The preprocessing step is to fill a buffer with r samples.

For each sample removed, the algorithm performs d steps of the refill algorithm, placing the newly computed samples in the second buffer. When the first buffer is empty, the second will be full w.h.p., and the buffers are swapped. Thus, a sample can be returned after $d = \mathcal{O}(1)$ steps, w.h.p.. □

Notice that this preprocessing is specific to a particular n, and therefore it is not quite comparable to the preprocessing described in Theorem 1, since that preprocessing applies to all $n' \leq n$. In order to achieve a similar universal preprocessing for all n' up to n, one can apply the decomposition into sums of square numbers described in Sect. 3. Note that in this case, we can decompose n' into sums of $\mathcal{O}(1)$ square numbers, and thus achieve a preprocessing of $O(n^{\varepsilon})$, for any fixed $\varepsilon > 0$.

5 Empirical Evaluation

We conducted experiments to compare the quality of generated variates and to compare the computation time used for generating variates among the algorithms with and without loss of precision. We compared our proposed algorithm (Theorem 1) with BTPE [15], which is the algorithm for binomial sampling in both R [22] and GNU Scientific Library (GSL) [11]. The implementation in GSL was used for the experiments.

We generated 10^8 random variates from $B(2^{30}, 1/2)$ and plotted a histogram of outputs. BTPE demonstrated a discontinuity and an overestimate in the tail probability, as shown in Fig. 2. In contrast, the histogram produced by our algorithm is smooth through the region of interest.

Our algorithm takes constant time with high probability, requiring the access of a data structure of size $\mathcal{O}(n^{1/2+\varepsilon})$. In contrast, BTPE takes approximately constant time with few memory accesses. As shown in Fig. 3, the computation time of generating 10^8 variates by our algorithm is about 6 times faster than that by BTPE and drops to 1.2 times faster as n increases due to more cache misses. The bottom two lines in Fig. 3 indicate the runtime of just the queries, or the queries plus the preprocessing time. Even if the preprocessing time is included, our algorithm slightly outperforms BTPE in the range of interest. Given that the preprocessing can be universally used for all n and p, it makes sense to compare the query time only, which is 3 to 6 times faster than BTPE for $n \leq 10^9$. Thus, we are competitive in speed and we do not suffer from round-off anomalies.

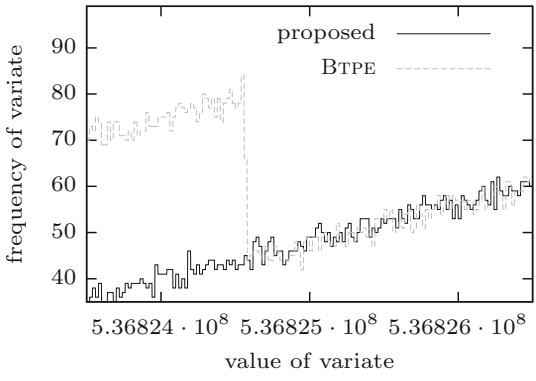

Fig. 2 The histograph of results for BTPE in discontinuous $B(2^{30}, 1/2)$. The histograph is smoothed by window averaging, with a window of size 20. BTPE oversamples the tail probability by a factor of 2.59, that increases the sum of frequencies by 0.74 %

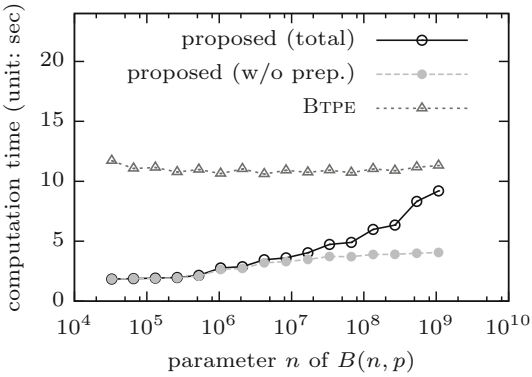

Fig. 3 Compare the computation time used for generating 10^8 variates from $B(n, 1/2)$ by different algorithms. Each data point is an average of 10 experiments. Consider that the preprocessed data structure can be reusable, we compare the running time including/excluding the preprocessing time with that of BTPE

We implemented our algorithm with parameter $s = 2\log_2 n$ in C++ with GNU Scientific Library [11] and GNU Multiple Precision Arithmetic Library [12], compiled it with g++4.63 with optimization flag -O3. The machine we used is equipped with a Celeron G530 2.4GHz CPU and 2GB of 1066MHz RAM. The operating system is Ubuntu 12.04 Desktop. The computation time is measured by the wall time, i.e. the elapsed time.

Acknowledgments We sincerely thank the reviewers for their helpful commments.

References

1. Abe, J., Kamimura, Y.: Do female parasitoid wasps recognize and adjust sex ratios to build cooperative relationships? J. Evol. Biol. **25**(7), 1427–1437 (2012)
2. Ahrens, J.H., Dieter, U.: Sampling from binomial and Poisson distributions: A method with bounded computation times. Computing **25**(3), 193–208 (1980)
3. Batagelj, V., Brandes, U.: Efficient generation of large random networks. Phys. Rev. E **71**(3), 36113 (2005)
4. Blanca, A., Mihail, M.: Efficient generation ε-close to G(n,p) and generalizations. CoRR (2012). arXiv:1204.5834
5. Bringmann, K., Friedrich, T.: Exact and efficient generation of geometric random variates and random graphs. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M.Z., Peleg, D. (eds.) Automata, Languages, and Programming (ICALP), vol. 7965, pp. 267–278. Springer, Berlin, Heidelberg (2013)
6. Bringmann, K., Kuhn, F., Panagiotou, K., Peter, U., Thomas, H.: Internal dla: Efficient simulation of a physical growth model. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) Automata, Languages, and Programming (ICALP), vol. 8572, pp. 247–258. Springer, Berlin, Heidelberg (2014)
7. Devroye, L.: Generating the maximum of independent identically distributed random variables. Comput. Math. Appl. **6**(3), 305–315 (1980)
8. Devroye, L.: Non-Uniform Random Variate Generation. Springer, Berlin (1986)
9. Farach-Colton, M., Tsai, M.T.: Exact sublinear binomial sampling. In: Cai, L., Cheng, S.-W., Lam, T.W. (eds.) Algorithms and Computation (ISAAC), vol. 8283, pp. 240–250. Springer, Berlin, Heidelberg (2013)
10. Fox, J., Weisberg, S.: An R Companion to Applied Regression. Sage Publications, New York (2010)
11. Galassi, M.: Gnu Scientific Library: Reference Manual (2003)
12. Granlund, T.: The GMP Development Team. GNU MP: The GNU Multiple Precision Arithmetic Library (2012)
13. Hörmann, W.: The generation of binomial random variates. J. Stat. Comput. Simul. **46**(1–2), 101–110 (1993)

14. Hörmann, W., Leydold, J., Derflinger, G.: Automatic Nonuniform Random Variate Generation. Springer, Berlin (2004)
15. Kachitvichyanukul, V., Schmeiser, B.W.: Binomial random variate generation. Commun. ACM **31**(2), 216–222 (1988)
16. Karney, C.F.F.: Sampling exactly from the normal distribution. CoRR (2013). arXiv:1303.6257
17. Knuth, D., Yao, A.: Algorithms and Complexity: New Directions and Recent Results, Chap. The Complexity of Nonuniform Random Number Generation. Academic Press, Waltham (1976)
18. Knuth, D.E.: The Art of Computer Programming: Seminumerical Algorithms, vol. 2. Addison-Wesley, Boston (1997)
19. Kronmal, R.A., Peterson, A.V.J.: On the alias method for generating random variables from a discrete distribution. Am. Stat. **33**(4), 214–218 (1979)
20. Miller, J.C., Hagberg, A.: Efficient generation of networks with given expected degrees. In: Frieze, A.M., Horn, P., Pralat, P. (eds.) Algorithms and Models for the Web Graph (WAW), vol. 6732, pp. 115–126. Springer, Berlin, Heidelberg (2011)
21. Patterson, R.S., Louisville, U.: Testing the Effects of Predictors Using Data Generated by Non-identity Link Functions of the Single-Index Model: A Monte Carlo Approach. University of Louisville, Louisville (2008)
22. R Development Core Team: R: A Language and Environment for Statistical Computing (2008)
23. Stadlober, E., Zechner, H.: The patchwork rejection technique for sampling from unimodal distributions. Trans. Model. Comput. Simul. **9**(1), 59–80 (1999)
24. Stillwell, J.: Elements of Number Theory. Springer, Berlin (2003)
25. Tsai, M., Wang, D., Liau, C., Hsu, T.: Heterogeneous subset sampling. In: Thai, M.T., Sahni, S. (eds.) Computing and Combinatorics (COCOON). vol. 6196, pp. 500–509. Springer, Berlin, Heidelberg (2010)