Exact Sublinear Binomial Sampling*

Martín Farach-Colton and Meng-Tsung Tsai

Rutgers University, New Brunswick NJ 08901, USA
{farach,mtsung.tsai}@cs.rutgers.edu

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 and has no precision loss, however, this method is often too slow in many settings. 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 (GSL) [10], however, all known sublinear-time algorithms involve precisions loss, which introduces artifacts into the sampling, such as discontinuities.

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.

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

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

To draw a random variate b from a binomial distribution $B(n,p)$, one can naively realize n Bernoulli trials of success rate p and count how many of them have a positive outcome in $O(n)$ time. In other words, binomial sampling can be used as an alternative for realizing n Bernoulli trials.

Sampling variates from binomial distribution is a common procedure provided by the GNU Scientific Library [10] and the statistical software R [22], both of which use the algorithm BTPE proposed in [14]. These implementations have inaccuracies, such as the discontinuity shown in Figure 1. In particular, we show that BTPE substantially overestimates the probability of the tail of the distribution. In Figure 1, the overestimation is by a factor of 2.59, or 0.74% of the total samples. Thus, the occurence of a rare event in BTPE cannot be trusted.

Many applications use these implementations as part of the procedures, such as the efficient generation of random graphs from $\mathcal{G}(n,p)$ [3,19,4], logistic regression [9], generating virtual data sets in GLM analysis [1], generating random data or parameters [21] and speeding up Monte-Carlo simulation system [27].

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

L. Cai, S.-W. Cheng, and T.-W. Lam (Eds.): ISAAC2013, LNCS 8283, pp. 240–250, 2013.

Recently, both efficient and exact sampling algorithm for some distributions are developed, e.g. for normal and geometric distributions [15,5] but not yet for binomial distribution. In this paper, we present what is, to the best of our knowledge, the **first sublinear-time algorithm** for drawing a sample from a binomial distribution **with no loss of precision**. In particular, we show that:

Theorem 1. *Given a binomial distribution $B(n,p)$ for $n \in \mathbb{N}$, $p \in \mathbb{Q}$, drawing a sample from it takes $\mathcal{O}(\log^2 n)$ time using $\mathcal{O}(n^{1/2+\epsilon})$ space w.h.p., after $\mathcal{O}(n^{1/2+\epsilon})$-time preprocessing for small $\epsilon > 0$. The preprocessing does not depend on p and can be used for any $p' \in \mathbb{Q}$ and for any $n' \leq n$.*

Previous Work. Several sublinear-time algorithms have been described [17,7,14], although all of these trade precision for speed. Algorithms Binv [14] and Bg [6,14] 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 [18,14] requires calculating $\binom{n}{k}$ for all k in $\{0,1,\ldots,n\}$ and constructing an alias table [18] 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 with bounded precision. Algorithm Btpe [14] divides the binomial distribution into parts and approximates each part by an upper-bound 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 skill is known as the accept/reject rule, used in [17,14,7,27]. Btpe runs in sublinear time and is used by default in the statistical software R and GNU Scientific Library [22,10]. 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 [2,7,12,25,13] 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 at least linear. It is not clear how to modify them to be both accurate to full precision and sublinear.

Organization. In Section 2, preliminary definitions and building blocks are introduced. In Section 3, a simple algorithm is devised and further revised to achieve the claimed time complexity. Then, in Section 4, we conducted a set of experiments to compare the default algorithm used in GNU Scientific Library with the proposed one.

2 Preliminaries

To make it easier to understand the proposed algorithms, we sketch the outline of the algorithms and make definitions in this section. Given an input of a positive integer n and a real number $p \in [0,1]$, the output is an integer $b \in \{0,1,\ldots,n\}$ selected so that $\Pr[b=k] = \binom{n}{k} p^k (1-p)^{n-k}$ for all $k \in \{0,1,\ldots,n\}$; that is, a

sample b is drawn from the binomial distribution $B(n,p)$ of n trials and success rate p. We analyze algorithms under the log-cost RAM model [20]. We assume that it takes $\mathcal{O}(1)$ time to do arithmetic calculations on constant number of operands of $\mathcal{O}(\log n)$ bits and to generate a fair binary random bit $u \in \{0,1\}$. We assume that p is rational and thus p can be represented by finite number of digits in base two, possibly repeating. Without loss of generality, let p be $(0.a_1 \cdots a_\ell \overline{a_{\ell+1} \cdots a_{\ell+r}})_2$. If $r > 0$, the part $a_{\ell+1} \cdots a_{\ell+r}$ repeats; otherwise, no part repeats. Formally, if $r > 0$, $a_i = a_{i-r}$ for all $i > \ell + r$; otherwise, $a_i = 0$ for all $i > \ell$.

Consider determining a Bernoulli trial T with success rate $p = (0.a_1 a_2 \cdots)_2$, 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 $u = a_1$, proceed to the next digit and repeat the procedure. Then, expected $\mathcal{O}(1)$ comparisons are needed.

A binomial variate b can be sampled from $B(n,p)$ by simply checking how many of n Bernoulli trials have a positive outcome. We can mimic the single Bernoulli trial procedure above in which we replaced 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 digit 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 digits 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, both in expectation and with high probability, considering that n will be roughly halved ($\leq n/2 + \sqrt{cn \ln n}$) in each round with probability $1 - \mathcal{O}(1/n^c)$.

In Section 3, we show how to construct a structure $S(n,c)$, a variation of *discrete distribution generating tree* [16]. Both construction time and used space are $\mathcal{O}((n \log^3 n)^{1/2})$. Affter which, sampling a variate from $B(n,1/2)$ takes $\mathcal{O}(\log n)$ time (matched the possible optimal bound [16]) with probability $1 - \mathcal{O}(1/n^c)$ for any constant $c \geq 1$. We also show how to construct $S(n,\infty)$ in $\mathcal{O}(n^2)$ time.

Note that drawing a variate from $B(n,p)$ by the above procedure possibly uses $B(n',1/2)$ for all $n' \in \{1,2,\ldots,n\}$. It would be too slow to construct $S(n',c)$ whenever we need to generate a variate from $B(n',1/2)$. A possible solution is to construct $S(n',c(n'))$ for all $n' \in \{1,2,4,\ldots,\lfloor\!\lfloor n \rfloor\!\rfloor\}$ where $c(n') = 4c$ if $n' > n^{1/4}$ or otherwise $c(n') = \infty$ and $\lfloor\!\lfloor x \rfloor\!\rfloor$ is the largest power of two no greater than x. To generate a variate b from $B(n',1/2)$ where n' is not a power of two, we decompose n' into $h = \mathcal{O}(\log n)$ powers of two $\omega_1, \omega_2, \ldots, \omega_h$, generate a variate b_i from each $B(\omega_i, 1/2)$, and let $b = \sum_i b_i$. Therefore, if we have the structures $S(n',c(n'))$ for all $n' \in \{1,2,4,\ldots,\lfloor\!\lfloor n \rfloor\!\rfloor\}$, generating a variate from $B(n',1/2)$ for any $n' \leq n$ takes $\mathcal{O}(\log^2 n)$ time with probability $1 - \Pr\left[\bigcup_i \mathcal{E}_i\right] \leq 1 - \sum_i \Pr\left[\mathcal{E}_i\right] = 1 - \mathcal{O}(h/n^c) = 1 - \mathcal{O}(\log n/n^c)$ where $\mathcal{E}_i$ denotes the event that generating b_i takes more than $\mathcal{O}(\log n)$ time. As a result, generating a variate from $B(n,p)$ takes $\mathcal{O}(\log^3 n)$ time with probability higher than $1 - \mathcal{O}(\log^2 n/n^c)$. We call this *simple sampling*.

Our *refined sampling* algorithm is based on the fact that every natural number is sum of four square numbers [26]. However, we are not going to construct $S(n', c(n'))$ for all $n' \in \mathcal{W} = \{k^2 : k \leq n\}$ because it would take too long. Instead, we construct a much smaller set $\mathcal{W}' \subset \mathcal{W}$ such that every natural number is the sum of h numbers in $\mathcal{W}'$, where h is a multiple of 4. Let $h = 16$, for example, $|\mathcal{W}'| \approx |\mathcal{W}|^{1/2}$. Therefore, whenever $S(n', c(n'))$ is needed and $n' \notin \mathcal{W}'$, we decompose n' into square numbers $w_1, w_2, \ldots, w_h \in \mathcal{W}'$, generate a variant b_i from each $B(w_i, 1/2)$, and let $b = \sum_i b_i$. Since h is constant, generating a variate from $B(n, p)$ takes $\mathcal{O}(\log^2 n)$ time with probability higher than $1 - \mathcal{O}(\log n / n^c)$.

3 Exact Binomial Sampling

In Section 2, we described how generating a variate from $B(n, p)$ can be reduced to generating variates from a sequence of $\mathcal{O}(\log n)$ $B(n', 1/2)$ with probability higher than $1 - \mathcal{O}(\log n / n^c)$ no matter what the precision of p is and where $n' \in \{1, 2, \ldots, n\}$. In this section, we begin by showing how to construct a structure $S(n, c)$ for generating a variate from $B(n, 1/2)$ in $\mathcal{O}(\log n)$ time with probability higher than $1 - \mathcal{O}(1/n^c)$. Then, we show that constructing $S(n', c)$ for each n' in a small subset of $\{1, 2, \ldots, n\}$ suffices to generate $B(n', 1/2)$ for each $n' \in \{1, 2, \ldots, n\}$.

In order to sample from $B(n, 1/2)$, we need to be careful of how many bits we use. To see why, consider that a generated variate b has value k with probability $P[b = k] = \binom{n}{k}/2^n$. The value of these probabilities can vary from 2^{-n} to $\Theta(1/\sqrt{n})$. Thus, if we are not careful, we end up manipulating probabilities that take n bits to represent.

The main idea will be to construct a structure that uses fewer bits to represent the needed probabilities. In most cases, this will be enough to correctly compute the variates. In order to compute the sample with exactly the correct probabilities, our structure $S(n, c)$ will change from time to time, but with low probability, expand the number of bits it uses to calculate the sample. Therefore, with high probability, $S(n, c)$ will be small and fast, but occasionally we might expand it.

Consider the event space for sampling. In order to generate a variate from $B(n, 1/2)$, one must pick an event e_i from $E = \{e_0, e_1, \ldots, e_n\}$ with probability $p_i = \binom{n}{i}/2^n$ and output the chosen i as the generated variate. We decompose each $p_i = \binom{n}{i}/2^n$ into ℓ_i powers of two which sum to p_i and replace each $e_i \in E$ with $e_{i1}, e_{i2}, \ldots, e_{i\ell_i}$. If e_{ij} is picked, then the generated variate is i. Each event e_{ij} from $\hat{E} = \{e_{ij} : 0 \leq i \leq n, 1 \leq j \leq \ell_i\}$ is selected with probability p_{ij}, each of which is a power of two. Note that there are many different ways to decompose a probability p_i into powers of two, though in only one of them does each power of two occur only once. We will consider decompositions in which some powers of two might appear twice so that it leaves the flexibility of computing p_i incrementally without worrying about carries in arithmetic operations. The distribution on the p_{ij} is heavily biased towards a few high-probability events, which correspond to the most significant bits of the binomial coefficients in base two.

We construct a *power tree* on $\hat{E}$ in order to draw an event e_{ij} from $\hat{E}$ with probability p_{ij}. Let a power tree be a binary tree in which each node of depth d is associated with a probability 2^{-d}. The set of probabilities on the leaf nodes of $T_{\hat{E}}$ are exactly $\hat{P} = \{p_{ij} : e_{ij} \in \hat{E}\}$, in one-to-one correspondence, in an arbitrary order. A variate can be generated according to $\hat{P}$ by taking a random, fair root-to-leaf path in $T_{\hat{E}}$.

To construct a power tree, we start with a root node associated with probability 1 and make it an unlabelled leaf. We process each $e_{ij} \in \hat{E}$ in an arbitrary order as follows. When processing e_{ij}, find a candidate unlabeled leaf with smallest probability p no smaller than p_{ij}. If $p = p_{ij}$, label the leaf with e_{ij}. Otherwise, replace the candidate leaf with two children, each with half the probability, and make one leaf the new candidate. Proceed until a leaf has been labelled with e_{ij}.

Note that after processing each e_{ij}, no two unlabelled leaves have the same probability. Therefore, if $Q \subseteq \hat{E}$ is the set of events we have processed so far and $p_Q = \min\{p_{ij} : e_{ij} \in Q\}$, there are $|Q|$ labelled leaves and at most $1 + \log(1/p_Q)$ unlabelled leaves, so the tree has size $\mathcal{O}(|Q| + \log(1/p_Q))$ in total.

Lemma 1. *Given an event set $\hat{E}$, the power tree on $\hat{E}$ can be constructed in $\mathcal{O}(|\hat{E}|)$ time using $\mathcal{O}(|\hat{E}|)$ space. The construction is incremental, adding one event at a time. During the construction, if some $Q \subseteq \hat{E}$ has processed, both running time and used space are $\mathcal{O}(|Q| + \log(1/p_Q))$.*

Proof. We maintain an array of pointers, $\rho_0, \rho_1, \ldots, \rho_{\log(1/p_Q)}$, where ρ_i points to the unique unlabelled leaf with probability 2^{-i}, if such a leaf exists. Then, it takes $\mathcal{O}(1)$ time to process an event e_{ij} if $\rho_{\log(1/p_{ij})}$ points to an unlabelled node; otherwise, we have to find the unlabelled leaf with smallest p no smaller than p_{ij} by traversing the pointer array in $\mathcal{O}(\log(1/p_{ij}) - \log(1/p))$ time. Simple charging scheme, in which the event that consumes a leaf is charged for creating the leaf, yields the bound.

We show that the construction never fails as follows. Let U be the set of probabilities of unlabelled leaves. For each coming e_{ij}, the associated probability $p_{ij} \leq \sum_{p \in U} p$. If the procedure fails as e_{ij} comes, then it means we cannot find $p \in U, p \geq p_{ij}$. Since each $p \in U$ is an unique power of two, if each $p \in U, p < p_{ij}$, then $\sum_{p \in U} p < p_{ij}$, a contradiction. □

The set $\hat{E}$ is as large as $\mathcal{O}(n^2)$ for the power tree of $B(n, 1/2)$ because each p_i requires $O(n)$ bits, as noted above. Therefore, fully constructing power tree for $B(n, 1/2)$, i.e. $S(n, \infty)$, takes $\mathcal{O}(n^2)$ time. We will construct the power tree of a set Q chosen so that it has $o(n)$ events and its power tree has depth $O(\log n)$, but so that the total probability of $\hat{E} - Q$ is polynomially small. Lemma 2 establishes that such a set always exists. If we sample using the power tree of Q, we will almost always reach a labelled leaf. If we reach an unlabelled leaf, we complete the construct of the power tree of $\hat{E}$ to finish the sampling procedure. That is, once a random walk reaches an unlabeled leaf, we postpone the on-going random walk, complete the unfinished part of the power tree construction, and then resume the random walk at the point where it is postponed. Thus, with very high probability, we will use small space and time to sample from $\hat{E}$.

Since the size of Q depends on how we decompose p_i of $e_i \in \hat{E}$, we describe the decomposition of p_i first to complete the claim that Q has small size and then describe the motivation of p_i's decomposition. For each p_i, we find two positive numbers $\mathcal{H}_s(p_i)$ and $\mathcal{L}_s(p_i)$ which add to p_i. The first is roughly the high-order $(c+1)\log n$ bits of p_i and the second is roughly the remaining (low order) bits. Ideally, they would be exactly the high and lower order bits, but it would be computationally expensive to find the high-order bits of $\binom{n}{i}$, considering the difficulty to determine certain bit of the product of two integers [23]. As the computation efficiency is concerned, to make $\mathcal{H}_s(p_i)$ to be of short length and $\mathcal{L}_s(p_i)$ to be of small value, we let $\mathcal{H}_s(p_i)$ and $\mathcal{L}_s(p_i)$ have an overlap as follows.

The value $\mathcal{L}_s(p_i) = p_i - \mathcal{H}_s(p_i)$, so we only need to worry about $\mathcal{H}_s(p_i)$. We defer the discussion of exactly how to compute $\mathcal{H}_s(p_i)$ until later and here note that $\mathcal{H}_s(p_i) = k \cdot 2^s$ for non-negative integer k and $p_i - \mathcal{H}_s(p_i) < 2 \cdot 2^s$. When we drop the subscript s, the default value of $s = -(c+1)\log n$. Note that, if $\mathcal{H}_r(p_i)$ for $r < s$ is given, one can get $\mathcal{H}_s(p_i)$ by truncating the trailing $s - r$ bits of $\mathcal{H}_r(p_i)$ because $\mathcal{L}_r(p_i) < 2 \cdot 2^r \leq 2^s$ and trailing $s - r$ bits of $\mathcal{H}_r(p_i)$ are less than 2^s, which add up to a value less than $2 \cdot 2^s$. We will show that such an $\mathcal{H}_s(p_i)$ can always be selected. And the only probability that a power of two potentially gets repeated in such a decomposition of $\mathcal{H}_s(p_i)$ and $\mathcal{L}_s(p_i)$ is 2^s.

The high order bits of many events are all zeros because the binomial coefficient are strongly concentrated. We define the major range where the events of non-zero high order bits located to be $\mathcal{C}(n) = [n/2 \pm (cn \ln n)^{1/2}]$ (C for center). Trivially, the number of bits is $(c+1)\log n * 2(cn \ln n)^{1/2}$, so picking this for Q is small. Lemma 2 shows that the probability is high. Because every $\mathcal{H}(p_i)$ is a multiple of $2^{-(c+1)\log n}$, this bounds the depth of the power tree to be $\mathcal{O}(c \log n)$.

To obtain Q, we decompose $\mathcal{H}(p_i)$ into its binary representation for $i \in \mathcal{C}(n)$. The following lemma completes the claim that $|Q|$ is small. Let $\mathrm{bit}(p)$ be the set of powers of two in the binary representation of p.

Lemma 2. *Let $\mathcal{P}(E)$ be the sum of probabilities associated with the events in E. Let Q be the set of events associated with the probability in*

$$\bigcup_{i \in \mathcal{C}(n)} \mathrm{bit}(\mathcal{H}(p_i)). \tag{2}$$

Then, $|Q|$ is $\mathcal{O}((c^3 n \log^3 n)^{1/2})$ and $\mathcal{P}(Q) \geq 1 - \mathcal{O}(1/n^c)$.

Proof. Let Q_1 and Q_2 be the set of events associated with the probability in, respectively,

$$\bigcup_{i \notin \mathcal{C}(n)} \mathrm{bit}(\mathcal{H}(p_i)) \cup \mathrm{bit}(\mathcal{L}(p_i)) \text{ and } \bigcup_{i \in [0,n]} \mathrm{bit}(\mathcal{L}(p_i)). \tag{3}$$

We have $\mathcal{P}(Q_1) \leq 2/n^c$ by Chernoff bound [8] and $\mathcal{P}(Q_2) \leq 2(n+1)/n^{c+1}$ by the definition of $\mathcal{L}(p_i)$. Because $\overline{Q} = Q_1 \cup Q_2$,

$$\mathcal{P}(Q) \geq 1 - \mathcal{P}(Q_1) - \mathcal{P}(Q_2) = 1 - \mathcal{O}(1/n^c) \tag{4}$$

as desired. Then, $|Q| = (cn \log n)^{1/2}(c+1)\log n = \mathcal{O}((c^3 n \log^3 n)^{1/2})$. □

We claim that $\mathcal{H}(p_i)$ for all $i \in \mathcal{C}(n)$ can be efficiently computed. We first show how to compute $\mathcal{H}(p_i)$ for $i \in \mathcal{C}(n)$ from $\mathcal{H}(p_{\lfloor n/2 \rfloor})$ and then show how to compute $\mathcal{H}(p_{\lfloor n/2 \rfloor})$ itself. Given the $\mathcal{H}_s(p_{\lfloor n/2 \rfloor})$ for $s = (c+2)\log n$, Lemma 3 shows how we can compute $\mathcal{H}(p_i)$ for $i \in \mathcal{C}(n)$ in $\mathcal{O}((c^3 n \log n)^{1/2})$ time. In other words, if we have the central probability of the binomial computed to more precision, we can use that to compute the surrounding values.

Lemma 3. *Given $\mathcal{H}_s(p_{\lfloor n/2 \rfloor})$ for $s \leq -(c+2)\log n$, it takes $\mathcal{O}((c^3 n \log n)^{1/2})$ time to calculate $\mathcal{H}(p_i)$ for all $i \in \mathcal{C}(n)$.*

Proof. Let $\lfloor a \rfloor_s \equiv \lfloor a/2^s \rfloor 2^s$. Given $\mathcal{H}_s(p_{\lfloor n/2 \rfloor})$, we claim that $\mathcal{H}(p_{\lfloor n/2 \rfloor - 1})$ can be obtained by truncating some trailing bits in $\ell_1 \equiv \lfloor \mathcal{H}_s(p_{\lfloor n/2 \rfloor}) * r_1 \rfloor_s$ where $r_1 = \binom{n}{\lfloor n/2 \rfloor - 1} / \binom{n}{\lfloor n/2 \rfloor} = \lfloor n/2 \rfloor / (\lfloor n/2 \rfloor + 1) < 1$ because

$$\begin{aligned} p_{\lfloor n/2 \rfloor - 1} - \ell_1 &= p_{\lfloor n/2 \rfloor} * r_1 - \ell_1 \\ &= (\mathcal{H}_s(p_{\lfloor n/2 \rfloor}) + \mathcal{L}_s(p_{\lfloor n/2 \rfloor})) * r_1 - \ell_1 \\ &= (\mathcal{H}_s(p_{\lfloor n/2 \rfloor}) * r_1 - \lfloor \mathcal{H}_s(p_{\lfloor n/2 \rfloor}) * r_1 \rfloor_s) + \mathcal{L}_s(p_{\lfloor n/2 \rfloor}) * r_1 \\ &< 2^s + 2 \cdot 2^s < 3 \cdot 2^s. \end{aligned}$$

Then, ℓ_1 has $-s$ bits and $p_{\lfloor n/2 \rfloor - 1} - \ell_1 < 3 \cdot 2^s$. If we get ℓ_1' by truncating the last two bits in ℓ_1, then ℓ_1' has $-s-2$ bits and $p_{\lfloor n/2 \rfloor - 1} - \ell_1' < 3 \cdot 2^s + 3 \cdot 2^s < 2 \cdot 2^{s+2}$. Hence, ℓ_1' is a valid $\mathcal{H}_{s+2}(p_{\lfloor n/2 \rfloor - 1})$ and we have $\mathcal{H}(p_{\lfloor n/2 \rfloor - 1})$ if $s \leq -(c+2)\log n$. Similarly, we claim that $\mathcal{H}(p_{\lfloor n/2 \rfloor - 2})$ can be obtained by truncating some trailing bits in $\ell_2 \equiv \lfloor \ell_1 * r_2 \rfloor_s$ where $r_2 = (\lfloor n/2 \rfloor - 1)/(\lfloor n/2 \rfloor + 2) < 1$ because

$$\begin{aligned} p_{\lfloor n/2 \rfloor - 2} - \ell_2 &= p_{\lfloor n/2 \rfloor - 1} * r_2 - \ell_2 \\ &= (\ell_1 + (p_{\lfloor n/2 \rfloor - 1} - \ell_1)) * r_2 - \ell_2 \\ &= (\ell_1 * r_2 - \lfloor \ell_1 * r_2 \rfloor_s) + (p_{\lfloor n/2 \rfloor - 1} - \ell_1) \\ &< 2^s + 3 \cdot 2^s < 4 \cdot 2^s \end{aligned}$$

Again, we get ℓ_2' by truncating the last two bits in ℓ_2, then ℓ_2' has $-s-2$ bits and $p_{\lfloor n/2 \rfloor - 2} - \ell_2' < 3 \cdot 2^s + 4 \cdot 2^s < 2 \cdot 2^{s+2}$ as desired. Clearly, $p_{\lfloor n/2 \rfloor - k} - \ell_k < (k+2) \cdot 2^s$ and we can calculate $\mathcal{H}(p_i)$ for all $i \in \mathcal{C}(n)$ if $s = -(c+2)\log n$. Since each ℓ_i' can be calculated by $\mathcal{O}(1)$ arithmetic calculations on operands of $\mathcal{O}(c \log n)$ bits, we need $\mathcal{O}(c)$ time for each ℓ_i' and thus $\mathcal{O}((c^3 n \log n)^{1/2})$ in total. □

To compute $\mathcal{H}_s(p_{\lfloor n/2 \rfloor})$ for $s = -(c+2)\log n$, we exploit the idea of computing Q; that is, consider summing up the terms of $i \in \mathcal{C}(n)$ in the equality

$$\binom{2k}{k} = \sum_i \binom{k}{i}^2 \approx \sum_{i \in \mathcal{C}(n)} \mathcal{H}\left(\binom{k}{i}\right)^2. \tag{5}$$

Suppose $\binom{k}{\lfloor k/2 \rfloor}$ has been computed, we use it to compute $\binom{k}{i}$ for $i \in \mathcal{C}(k)$ in $\mathcal{O}((c^3 k \log k)^{1/2})$ time by Lemma 3 and use all of them to compute $\binom{2k}{k}$ by Equation 5. To compute $\mathcal{H}_s\left(\binom{2k}{k}\right)$ with $s = 2k - (c+2)\log 2k$ (i.e. $\mathcal{H}_s(\binom{2k}{k})/2^{2k}$) with

$s = -(c+2)\log 2k)$, we need $\mathcal{H}_s\left(\binom{k}{\lfloor k/2 \rfloor}\right)$ with $s = k - (c+4+1/2)\log k$ implied by Lemma 4 if $k \geq 2^{(c+7)/2}$. Otherwise $k < 2^{(c+7)/2}$, the binomial coefficient $\binom{k}{\lfloor k/2 \rfloor}$ can be computed in $\mathcal{O}(1)$ time because c is a constant. Consequently, for even n, the computation of $\binom{n}{n/2}$ can be reduced to the computation of $\binom{n/2}{\lfloor n/2 \rfloor}$; for odd n, the computation of $\binom{n}{\lfloor n/2 \rfloor}$ could simply add $\binom{n-1}{\lfloor n/2 \rfloor}$ and $\binom{n-1}{\lfloor n/2 \rfloor - 1}$, each of which is a case of even n. We apply this procedure to compute $\binom{n}{\lfloor n/2 \rfloor}$ recursively and the time complexity is

$$\sum_{k=0}^{\log n} \mathcal{O}(((c + 5k/2 + 2)^3 (n/2^k)\log(n/2^k))^{1/2}) = \mathcal{O}((n \log^2 n)^{1/2}), \tag{6}$$

dominated by the construction time of the power tree, remarked in Theorem 2.

Lemma 4. *Let* $p_k = \mathcal{H}_s^2\left(\binom{n}{k}\right)/\binom{2n}{n}$ *with* $s \leq n - (2c + 1/2)\log n$. *Then,*

$$\sum_{k \in \mathcal{C}(n)} p_k \geq 1 - 18/n^{2c}.$$

Proof.

$$\sum_{k \in [0,n]} \binom{n}{k}^2 - \sum_{k \in \mathcal{C}(n)} \mathcal{H}_s^2\left(\binom{n}{k}\right) = \sum_{k \notin \mathcal{C}(n)} \binom{n}{k}^2 \tag{7}$$

$$+ \sum_{k \in \mathcal{C}(n)} 2\binom{n}{k}\mathcal{L}_s\left(\binom{n}{k}\right) - \mathcal{L}_s^2\left(\binom{n}{k}\right) \tag{8}$$

Consider the sum of sampled n values from a pool of n 0's and 1's without replacement. The probability of the sum being k is $q_k = \binom{n}{k}^2/\binom{2n}{n}$. By Corollary 1.1 in [24], we have $\sum_{k \notin \mathcal{C}(n)} q_k \leq 2\exp[-4(c \ln n)/(1 + 1/n)] \leq 2/n^{2c}$. Then, $(7)/\binom{2n}{n} \leq 2/n^{2c}$.

By definition, $\mathcal{L}_s(\binom{n}{k}) < 2 \cdot 2^s$ and therefore $(8) \leq 4 \cdot 2^s \cdot 2^n = 2^{n+s+2}$. Because $\binom{2n}{n} \geq 2^{2n-2}/n^{1/2}$ by Stirling's approximation, $(8)/\binom{2n}{n} \leq 16/n^{2c}$. Putting the results together, we have

$$\sum_{k \in \mathcal{C}(n)} \mathcal{H}_s^2\left(\binom{n}{k}\right)/\binom{2n}{n} \geq 1 - 18/n^{2c}. \qquad \square$$

Theorem 2. *To compute* $S(n, c)$, *it takes* $\mathcal{O}((n \log^3 n)^{1/2})$ *time, after which, it is stored in* $\mathcal{O}((n \log^3 n)^{1/2})$ *space. Given* $S(n, c)$, *it takes* $\mathcal{O}(\log n)$ *time to generate a variate from* $B(n, 1/2)$ *with probability* $1 - \mathcal{O}(1/n^c)$.

Now we have $S(n, c)$ for generating variates from $B(n, 1/2)$ and show how to use it for $B(n, p)$ as follows; that is, building a set of $S(n', c(n'))$. We suppress the term $c(n')$ for convience without changing the time complexity.

Simple Sampling. generates a variate from $B(n,p)$ for $n \in \mathbb{N}, p \in \mathbb{Q}$ denoted by $(0.a_1a_2\cdots)_2$ using the structure $S(n)$ for all $n' \in \{1,2,4,\ldots,\llbracket n \rrbracket\}$.

In Section 2, we have shown how to use $B(n',1/2)$ for $n' \in [n]$ to generate a variate from $B(n,p)$. Because $S(n')$ might not be contained in the constructed structure $\{S(1),S(2),S(4),\ldots,S(n)\}$, to generate a variate from $B(n',1/2)$, we generate a variate from each of $B(\omega_1,1/2),\ldots,B(\omega_h,1/2)$ and add the variates, where ω_i are powers of two added to n'. Clearly, h can be $\mathcal{O}(\log n')$.

There are $\mathcal{O}(\log n)$ steps in the reduction from $B(n,p)$ to $B(n',1/2)$ for $n' \in [n]$. Each step requires to generate a variate from $B(n',1/2)$, where n' can be decomposed into $\mathcal{O}(\log n')$ powers of two. Thus, it takes $\mathcal{O}(\log^2 n)$ time to generate a variate from $B(n',1/2)$ with probability $1-\mathcal{O}(\log n/n^c)$ by union bound. Considering that the number of steps is $\mathcal{O}(\log n)$ with probability $1-\mathcal{O}(1/n^c)$, the total running time for generating $B(n,p)$ is therefore $\mathcal{O}(\log^3 n)$ with probability $1-\mathcal{O}(\log^2 n/n^c)$.

Refined Sampling. is as the simple sampling but selects an integer set R such that every positive integer $n' \le n$ is a sum of j elements in R, where j is a constant. In this way, to generate a variate from $S(n')$, one can generate a variate from each of $S(w_1),S(w_2),\ldots,S(w_j)$ and add the generated variates, where $n' = w_1+w_2+\cdots+w_j$. $R' = \{k^2 \le n : k \in \mathbb{N}\}$ is a possible candidate for R because each positive integer is a sum of at most four square numbers [26]. However, we do not build the data structure $S(n')$ for all $n' \in R'$ because the time complexity $\sum_{k\in\{1,4,9,\ldots,n\}} \mathcal{O}((n\log^3 n)^{1/2}) = \mathcal{O}((n^2\log^3 n)^{1/2})$ is too much. A better candidate for R could be

$$R_1 \cup R_2 = \{k^2 \le n : k \in \mathbb{N}, k^2 \equiv 0 \bmod t\} \cup \{k^2 \le n : k \in \mathbb{N}, k^2 < t\}, \qquad (9)$$

where t is a chosen square number. Let n' be represented as w_1t+w_2. Because w_1t (resp. w_2) is a sum of at most 4 square numbers in R_1 (resp. R_2), every integer $n' \le n$ is a sum of at most 4×2 numbers in $R_1 \cup R_2$. Thus, the time complexity is reduced to $\mathcal{O}((t^2\log^3 t)^{1/2}) + \mathcal{O}((t(n/t)^2\log^3 n)^{1/2})$ or $\mathcal{O}((n^{4/3}\log^3 n)^{1/2})$ by letting $t = n^{2/3}$. Similarly, let $n' = w_1t_1+w_2t_2+\cdots+w_h$, the time complexity can be furtherly reduced to $\mathcal{O}((n^{2^h/(2^h-1)}\log^3 n)^{1/2}) = \mathcal{O}(n^{1/2+\epsilon})$ for some constant h and small $\epsilon > 0$. To decompose each w_it_i into four square numbers in the corresponding set R_i in $\mathcal{O}(1)$ time, we preprocess a small table for lookup. The small table is used to decompose integers no more than $n^{1/8+\epsilon}$ into four square numbers, whose construction time is bounded by $\mathcal{O}((n^{1/8+\epsilon})^4)$ dominated by the main procedure. Therefore, there is no problem to decompose $w_1,w_2,\ldots,w_{h-2}$ because all of them are no more than $n^{1/8+\epsilon}$. For w_{h-1} bounded by $n^{1/4+\epsilon}$, one can first decompose w_{h-1} as $x+y$, where x is the largest square number smaller than w_{h-1} and thus $y = \mathcal{O}(n^{1/8+\epsilon})$ so that it can be decomposed by table-lookup in $\mathcal{O}(1)$ time. Similar decomposition is applied to w_h. As a result, the mentioned constant $j = 4h+3$.

As the arguments used for the simple sampling, generating a variate from $B(n,p)$ takes $\mathcal{O}(\log^2 n)$ time with probability $1-\mathcal{O}(\log n/n^c)$ after $\mathcal{O}(n^{1/2+\epsilon})$-time preprocessing.

4 Empirical Evaluation

In this section, 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 compare our proposed algorithm with BTPE [14], which is the default algorithm for binomial sampling in both R [22] and GNU Scientific Library (GSL) [10]. The implementation of GSL is used to conduct the experiments.

We generated 10^8 random variates from $B(2^{30}, 1/2)$ and plotted a histogram of outputs. BTPE demonstrated a discontinuity, as shown in Figure 1.

Our algorithm takes poly-logarithmic time. In contrast, BTPE takes approximately constant time. As shown in Figure 2, the computation time is about the same for $n \leq 2^{15}$ and increases to 3.5 times more for $n = 2^{30}$. When we stored the power tree in pre-order, the running time of our algorithm improved, suggesting that cache misses were to blame. Therefore, our algorithm could benefit from more tuning.

We implemented our algorithm in C++ with GNU Scientific Library [10] and GNU Multiple Precision Arithmetic Library [11], 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 wall time, i.e., the elapsed time.

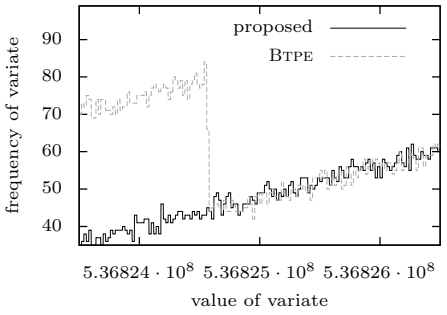

Fig. 1. The histograph of results for BTPE in discontinuous $B(2^{30}, 1/2)$. The histograph is smoothed by window averaging, with a window of size 20. BTPE oversamples the tail 0.74% of the time, that is, by a factor of 2.59.

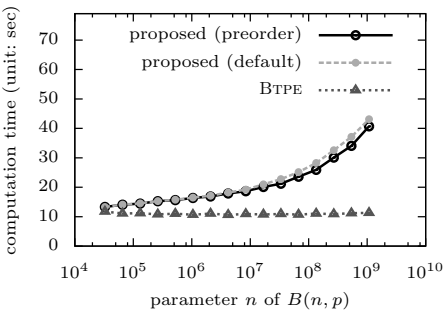

Fig. 2. Compare the computation time used for generating 10^8 variates from $B(n, 1/2)$ by different algorithms. Each data point is an average of 10 experiments.

References

1. Abe, J., Kamimura, Y.: Do female parasitoid wasps recognize and adjust sex ratios to build cooperative relationships? Journal of Evolutionary Biology 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. Physical Review E 71(3), 36113 (2005)

4. Blanca, A., Mihail, M.: Efficient generation - close to G(n,p) and generalizations. CoRR abs/1204.5834 (2012)
5. Bringmann, K., Friedrich, T.: Exact and efficient generation of geometric random variates and random graphs. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part I. LNCS, vol. 7965, pp. 267–278. Springer, Heidelberg (2013)
6. Devroye, L.: Generating the maximum of independent identically distributed random variables. Computers and Mathematics with Applications 6(3), 305–315 (1980)
7. Devroye, L.: Non-Uniform Random Variate Generation. Springer (1986)
8. Feller, W.: An Introduction to Probability Theory and Its Applications, vol. 1. Wiley (January 1968)
9. Fox, J., Weisberg, S.: An R Companion to Applied Regression. SAGE Publications (2010)
10. Galassi, M., et al.: Gnu Scientific Library: Reference Manual. Network Theory Ltd. (2003)
11. Granlund, T.: The GMP development team: GNU MP: The GNU Multiple Precision Arithmetic Library, 5.0.5 edn. (2012), `http://gmplib.org/`
12. Hörmann, W.: The generation of binomial random variates. Journal of Statistical Computation and Simulation 46, 101–110 (1993)
13. Hörmann, W., Leydold, J., Derflinger, G.: Automatic nonuniform random variate generation. Springer (2004)
14. Kachitvichyanukul, V., Schmeiser, B.W.: Binomial random variate generation. Commun. ACM 31, 216–222 (1988)
15. Karney, C.F.F.: Sampling exactly from the normal distribution. CoRR abs/1303.6257 (2013)
16. Knuth, D., Yao, A.: The complexity of nonuniform random number generation. In: Algorithms and Complexity: New Directions and Recent Results. Academic Press (1976)
17. Knuth, D.E.: The art of computer programming. Seminumerical algorithms, vol. 2. Addison-Wesley Longman Publishing Co., Inc. (1997)
18. Kronmal, R.A., Peterson, A.V.J.: On the alias method for generating random variables from a discrete distribution. The American Statistician 33(4), 214–218 (1979)
19. Miller, J.C., Hagberg, A.: Efficient generation of networks with given expected degrees. In: Frieze, A., Horn, P., Prałat, P. (eds.) WAW 2011. LNCS, vol. 6732, pp. 115–126. Springer, Heidelberg (2011)
20. Motwani, R., Raghavan, P.: Randomized algorithms. Cambridge University Press (1995)
21. Patterson, R.S.: of 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 (2008)
22. R Development Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing (2008)
23. Sauerhoff, M., Woelfel, P.: Time-space tradeoff lower bounds for integer multiplication and graphs of arithmetic functions. In: STOC, pp. 186–195. ACM (2003)
24. Serfling, R.J.: Probability Inequalities for the Sum in Sampling without Replacement. The Annals of Statistics 2(1), 39–48 (1974)
25. Stadlober, E., Zechner, H.: The patchwork rejection technique for sampling from unimodal distributions. ACM Trans. Model. Comput. Simul. 9(1), 59–80 (1999)
26. Stillwell, J.: Elements of Number Theory. Springer (2003)
27. Tsai, M.-T., Wang, D.-W., Liau, C.-J., Hsu, T.-S.: Heterogeneous subset sampling. In: Thai, M.T., Sahni, S. (eds.) COCOON 2010. LNCS, vol. 6196, pp. 500–509. Springer, Heidelberg (2010)