engineering proceedings

MDPI

Article

An Analysis of an Open Source Binomial Random Variate Generation Algorithm †

Vincent A. Cicirello

Computer Science, Stockton University, 101 Vera King Farris Dr, Galloway, NJ 08205, USA; cicirelv@stockton.edu

† Presented at the 4th International Electronic Conference on Applied Sciences, 27 October–10 November 2023; Available online: https://asec2023.sciforum.net/.

Abstract: The binomial distribution is the probability distribution of the number of successes for a sequence of n independent trials with success probability p. Efficiently generating binomial random variates is important in many modeling and simulation applications, such as in medicine, risk management, fraud and anomaly detection, among others. A variety of algorithms exist for generating binomial random variates. This paper concerns the algorithm chosen for $\rho\mu$, an open source Java library for efficient randomization, which uses a hybrid of two existing binomial random variate algorithms: the BTPE Algorithm (Binomial, Triangle, Parallelogram, Exponential), and the inverse transform for cases that BTPE cannot handle. BTPE uses rejection sampling, and BTPE's authors originally provided an analytical formula for the expected number of iterations in terms of n and p. That expression is complicated to interpret in practical contexts. I explore BTPE by instrumenting $\rho\mu$'s implementation to empirically analyze its acceptance/rejection behavior to gain further insight into its runtime performance. Although the number of iterations depends upon n and p, my experiments show that the average number of iterations is always under 2, and that the average number of random uniform variates required to generate a single random binomial is under 4 (2 per iteration). Thus, when analyzing the runtime of a simulation algorithm that includes steps generating random binomials, one can consider such steps to have a constant runtime.

Keywords: binomial; btpe; inverse transform; modeling; open source; random variate; simulation

1. Introduction

The binomial distribution is the probability distribution of the number of successes for a sequence of n independent Bernoulli trials with success probability p [1]. Binomial random variates are important in many modeling and simulation [2] applications, such as in medicine [3–6], risk management [7,8], fraud and anomaly detection [9], among others [10]; and many algorithms exist for their efficient generation [2,11–15].

The focus of this paper is on the algorithm chosen for generating binomial random variates for the $\rho\mu$ library [16]. The open source Java library $\rho\mu$ [16] provides enhanced random number generation atop what the Java API itself includes. Java 17 introduced a hierarchy of random number generator interfaces, several new random number generators, among other new randomization features [17]. The core functionality of $\rho\mu$ is provided through a hierarchy of wrapper classes, which corresponds with the hierarchy of random number generator interfaces introduced in Java 17. In some cases, $\rho\mu$'s classes override the behavior of Java's random number generators with faster algorithms, such as for random integers subject to a bound or generating random Gaussians; while in other cases, $\rho\mu$ adds functionality not built into the Java API's classes, such as additional distributions such as the binomial, among others [16]. The $\rho\mu$ library provides efficient random number generation to other libraries, such as JavaPermutationTools [18] and Chips-n-Salsa [19].

Motivating research question: What is the computational cost to generate a random value from binomial distribution $B(n, p)$? Answering this question is important for analysis

check for updates

Citation: Cicirello, V.A. An Analysis of an Open Source Binomial Random Variate Generation Algorithm. *Eng. Proc.* **2023**, *52*, 0. https://doi.org/

Academic Editor:

Published:

Copyright: © 2023 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

of algorithms that rely upon binomial random variates. To generate binomial random variates, $\rho\mu$ [16] utilizes a combination of the BTPE Algorithm (Binomial, Triangle, Parallelogram, Exponential) [11] and the inverse transform [11,15] for cases that can't be handled by BTPE. The runtime of the inverse transform is $O(np)$ [11,15]. However, BTPE's runtime doesn't appear to grow in the same way, if at all. BTPE uses acceptance-rejection sampling [20]. BTPE's authors originally provided an analytical formula for the expected number of acceptance-rejection iterations in terms of n and p. Interpreting that expression is less than practical. In order to further understand the computational efficiency of binomial random variate generation, I instrumented $\rho\mu$'s implementation of BTPE to empirically analyze its acceptance-rejection behavior to gain further insight into its runtime performance. Although the number of iterations depends upon n and p, my experiments show that the average number of iterations is always under 2, and that the average number of uniform random variates required to generate a single random binomial is under 4. Thus, when analyzing the runtime of a simulation algorithm that includes steps generating random binomials, one can consider such steps to have a constant runtime.

I explain the experimental methodology in Section 2, and I present the results in Section 3. The source code of the experiments, the raw and processed data, and analysis is available on GitHub at https://github.com/cicirello/btpe-iterations (accessed on 8 August 2023). The source code for $\rho\mu$ is also on GitHub at https://github.com/cicirello/rho-mu (accessed on 8 August 2023). I conclude with a discussion in Section 4.

2. Methods

2.1. Binomial Random Variate Generation

The $\rho\mu$ library [16] generates binomial random variates primarily using BTPE [11], falling back on the inverse transform [11,15] when np is small. BTPE divides the distribution into four parts, using triangular functions in the middle, and exponential functions in the tails; and uses acceptance-rejection sampling [20]. For complete details of BTPE, which are beyond the scope of this paper, I refer the reader to the article that introduced it [11].

2.2. Expected Acceptance-Rejection Iterations

To generate a random value from binomial distribution $B(n, p)$, each acceptance-rejection iteration of BTPE generates two random values from $U(0, 1)$, i.e., uniformly distributed over the interval $[0.0, 1.0)$. When they introduced BTPE, Kachitvichyanukul and Schmeiser determined that the expected number of iterations of BTPE is [11]:

$$\binom{n}{M} r^M (1-r)^{n-M} \int_{-\infty}^{\infty} t(x)\, dx, \quad (1)$$

where $r = \min(p, 1-p)$, $M = \lfloor nr + r \rfloor$, and $t(x)$ is BTPE's majorizing function (see [11] for details of $t(x)$). Since each iteration generates two random uniform values from $U(0, 1)$, the expected number of uniform variates required by BTPE is thus:

$$2\binom{n}{M} r^M (1-r)^{n-M} \int_{-\infty}^{\infty} t(x)\, dx. \quad (2)$$

2.3. Empirical Methodology

It is not obvious whether Equations (1) and (2) grow with n, or grow with np, or grow with nr, etc? And if so, how quickly? BTPE is fast. Despite being a 35 year old algorithm, it is one of the best available for all but the smallest np. I set out to empirically explore the runtime behavior of BTPE to provide a practical perspective to Equations (1) and (2).

To accomplish this, I wrapped an instance of Java's SPLITTABLERANDOM class, which implements the splitmix [21] pseudorandom number generator, in order to instrument it to count the number of calls to its NEXTDOUBLE() method, which generates uniform random floating-point values in the interval $[0.0, 1.0)$. This wrapped random number generator is then used as the source of randomness for $\rho\mu$'s implementation of BTPE.

I consider values of $n \in \{2^5, 2^6, \ldots, 2^{20}\}$. BTPE is only relevant for: $nr \geq 10$. Thus, $p = \frac{10}{n}$ is the minimum applicable value of p. For a given n, consider $p \in \{\frac{10}{n}, \frac{16}{n}, \frac{32}{n}, \ldots, \frac{1}{2}, \ldots, \frac{n-32}{n}, \frac{n-16}{n}, \frac{n-10}{n}\}$. For each combination of n and p, I use BTPE to generate 10,000 binomial random variates. During which, I compute the average number of uniform variates per binomial, with 95% confidence intervals. I use Equation (2) to predict the number of uniform variates for each case, and test significance with a t-test.

I used OpenJDK 17 on a Windows 10 PC with a 3.4 GHz AMD A10-5700 CPU and 8 GB RAM. The experiments used $\rho\mu$ 3.1.1. The source code for the experiments is on GitHub at https://github.com/cicirello/btpe-iterations (accessed on 8 August 2023), as well as for $\rho\mu$ at https://github.com/cicirello/rho-mu (accessed on 8 August 2023).

3. Results

Tables 1–4 show the results for $n \in \{2^5, 2^{10}, 2^{15}, 2^{20}\}$. These were chosen as representative cases. The raw and processed data for all cases are available on GitHub at https://github.com/cicirello/btpe-iterations (accessed on 8 August 2023). The empirical results confirm the analytical prediction of Equation (2). For all cases, there is no significant difference between the analytical prediction and the empirically computed means. T-test p-values are above 0.05 in almost all cases (well above in most cases). The small number of cases where the t-test p-values are less than 0.05 are explained by random chance. Due to random chance alone, at level 0.05, we should expect this for approximately 5% of cases. This occurred in 3 of the 72 cases represented in the tables (approximately 4% of cases).

Table 1. Average number of calls to $U(0,1)$ by $\rho\mu$'s BTPE implementation for n = 1,048,576.

p	$\rho\mu$ Mean	Predicted	T-Test p-Value
0.0000095367	3.81 ± 0.051	3.80	0.74
0.0000152588	3.42 ± 0.043	3.45	0.21
0.0000305176	2.99 ± 0.034	2.94	0.01
0.0000610352	2.60 ± 0.025	2.63	0.06
0.0001220703	2.48 ± 0.021	2.49	0.54
0.0002441406	2.35 ± 0.018	2.33	0.17
0.0004882812	2.29 ± 0.016	2.30	0.41
0.0009765625	2.26 ± 0.015	2.26	0.95
0.001953125	2.25 ± 0.015	2.26	0.48
0.00390625	2.27 ± 0.015	2.27	0.90
0.0078125	2.27 ± 0.015	2.28	0.27
0.015625	2.29 ± 0.016	2.29	0.61
0.03125	2.30 ± 0.016	2.30	0.65
0.0625	2.29 ± 0.016	2.30	0.17
0.125	2.29 ± 0.016	2.31	0.06
0.25	2.30 ± 0.016	2.31	0.24
0.5	2.31 ± 0.017	2.31	0.58
0.75	2.31 ± 0.016	2.31	0.60
0.875	2.31 ± 0.016	2.31	0.87
0.9375	2.30 ± 0.016	2.30	0.75
0.96875	2.29 ± 0.016	2.30	0.57
0.984375	2.28 ± 0.016	2.29	0.10
0.9921875	2.27 ± 0.015	2.28	0.29
0.99609375	2.27 ± 0.015	2.27	0.94
0.998046875	2.26 ± 0.015	2.26	0.58
0.9990234375	2.25 ± 0.015	2.26	0.47
0.9995117188	2.29 ± 0.016	2.30	0.63
0.9997558594	2.33 ± 0.017	2.33	0.62
0.9998779297	2.48 ± 0.021	2.49	0.28
0.9999389648	2.63 ± 0.025	2.63	0.73
0.9999694824	2.95 ± 0.033	2.94	0.78
0.9999847412	3.44 ± 0.043	3.45	0.75
0.9999904633	3.84 ± 0.053	3.80	0.17

Across all cases, the analytical prediction from Equation (2) indicates a maximum expected number of uniform variates approximately 3.84 ($n = 32$ and $p = 0.3125$). The empirical maximum mean was 3.84 and the minimum was 2.25. So although the average number of uniform variates needed by BTPE to generate one binomial variate fluctuates with n and p, it remains less than 4 even for very large n.

Table 2. Average number of calls to $U(0,1)$ by $\rho\mu$'s BTPE implementation for n = 32,768.

p	$\rho\mu$ Mean	Predicted	T-Test p-Value
0.0003051758	3.81 ± 0.052	3.80	0.73
0.0004882812	3.48 ± 0.044	3.45	0.13
0.0009765625	2.96 ± 0.033	2.94	0.50
0.001953125	2.62 ± 0.025	2.63	0.51
0.00390625	2.48 ± 0.021	2.49	0.15
0.0078125	2.34 ± 0.017	2.34	0.81
0.015625	2.27 ± 0.015	2.27	0.55
0.03125	2.26 ± 0.015	2.26	0.92
0.0625	2.25 ± 0.015	2.26	0.57
0.125	2.28 ± 0.015	2.28	0.43
0.25	2.29 ± 0.016	2.28	0.59
0.5	2.29 ± 0.016	2.30	0.08
0.75	2.30 ± 0.016	2.28	0.16
0.875	2.27 ± 0.016	2.28	0.62
0.9375	2.26 ± 0.015	2.26	0.76
0.96875	2.26 ± 0.015	2.26	0.53
0.984375	2.28 ± 0.015	2.27	0.15
0.9921875	2.32 ± 0.017	2.34	0.04
0.99609375	2.48 ± 0.021	2.49	0.22
0.998046875	2.62 ± 0.025	2.63	0.69
0.9990234375	2.98 ± 0.034	2.94	0.08
0.9995117188	3.45 ± 0.044	3.45	0.81
0.9996948242	3.78 ± 0.051	3.80	0.53

Table 3. Average number of calls to $U(0,1)$ by $\rho\mu$'s BTPE implementation for $n = 1024$.

p	$\rho\mu$ Mean	Predicted	T-Test p-Value
0.009765625	3.82 ± 0.051	3.80	0.42
0.015625	3.49 ± 0.045	3.46	0.16
0.03125	2.94 ± 0.033	2.97	0.06
0.0625	2.70 ± 0.027	2.69	0.53
0.125	2.52 ± 0.022	2.52	0.38
0.25	2.36 ± 0.018	2.34	0.09
0.5	2.30 ± 0.016	2.32	0.11
0.75	2.35 ± 0.018	2.34	0.36
0.875	2.54 ± 0.022	2.52	0.30
0.9375	2.67 ± 0.026	2.69	0.14
0.96875	3.00 ± 0.034	2.97	0.05
0.984375	3.41 ± 0.043	3.46	0.05
0.990234375	3.79 ± 0.051	3.80	0.74

Table 4. Average number of calls to $U(0,1)$ by $\rho\mu$'s BTPE implementation for $n = 32$.

p	$\rho\mu$ Mean	Predicted	T-Test p-Value
0.3125	3.83 ± 0.052	3.84	0.76
0.5	3.58 ± 0.046	3.60	0.46
0.6875	3.78 ± 0.050	3.84	0.03

4. Discussion and Conclusions

Modeling and simulation applications in many domains require efficiently generating binomial random variates. The runtime of some algorithms for such generation grows with n or with np. For example, the average runtime of the inverse transform approach is $O(np)$. Other algorithms are quite fast even for large np, such as BTPE. BTPE's runtime does vary based on n and p, as analyzed by its authors. However, in the empirical investigation in this paper, I complement the existing analytical result by showing that the average number of acceptance-rejection iterations is always less than 2, even for large n and np, and that the average number of uniform variates needed to generate a single binomial is less than 4. Thus, if generating a binomial random variate is a step of another algorithm, such steps can be treated as $O(1)$ in average case runtime analysis.

One limitation of this empirical study, as well as in the analytical expression of Equation (2), is that it considers the average case. The acceptance-rejection sampling cycle of BTPE can potentially run longer. For example, during the experiments, the maximum number of uniform variates generated while producing a single binomial was 38 (19 iterations), compared to the average of less than 4. Longer runs of BTPE are not common. For example, during this study, 2.88 million binomial random variates were generated, and longer runs of BTPE were relatively rare occurrences. However, if you are analyzing the algorithmic complexity of an algorithm that uses BTPE as a subroutine, this result limits you to an average case analysis of that algorithm, rather than a worst case analysis. We may explore in the future whether it may be possible to compute an upper bound on the number of acceptance-rejection sampling iterations to resolve this limitation.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All experiment data (raw and post-processed) are available on GitHub: https://github.com/cicirello/btpe-iterations (accessed on 8 August 2023), which also includes all source code of our experiments, as well as instructions for compiling and running the experiments.

Conflicts of Interest: The author declares no conflict of interest.

Abbreviation

The following abbreviation is used in this manuscript:

BTPE	Binomial, Triangle, Parallelogram, Exponential

References

1. Larson, H.J. *Introduction to Probability Theory and Statistical Inference*, 3rd ed.; Wiley: Hoboken, NJ, USA, 1982.
2. Kuhl, M.E. History of Random Variate Generation. In Proceedings of the 2017 Winter Simulation Conference (WSC), Las Vegas, NV, USA, 3–6 December 2017; pp. 231–242. https://doi.org/10.1109/WSC.2017.8247791.
3. Arshad, H.; Khan, M.A.; Sharif, M.; Yasmin, M.; Javed, M.Y. Multi-level features fusion and selection for human gait recognition: an optimized framework of Bayesian model and binomial distribution. *Int. J. Mach. Learn. Cybern.* **2019**, *10*, 3601–3618. https://doi.org/10.1007/s13042-019-00947-0.
4. Khan, M.; Olivier, J. Regression to the mean for the bivariate binomial distribution. *Stat. Med.* **2019**, *38*, 2391–2412. https://doi.org/10.1002/sim.8115.
5. Naimi, A.I.; Whitcomb, B.W. Estimating Risk Ratios and Risk Differences Using Regression. *Am. J. Epidemiol.* **2020**, *189*, 508–510. https://doi.org/10.1093/aje/kwaa044.
6. Singh, S.; Chawla, M.; Prasad, D.; Anand, D.; Alharbi, A.; Alosaimi, W. An Improved Binomial Distribution-Based Trust Management Algorithm for Remote Patient Monitoring in WBANs. *Sustainability* **2022**, *14*, 2141. https://doi.org/10.3390/su14042141.
7. Wang, G.; Pei, J. Macro Risk: A Versatile and Universal Strategy for Measuring the Overall Safety of Hazardous Industrial Installations in China. *Int. J. Environ. Res. Public Health* **2019**, *16*, 1680. https://doi.org/10.3390/ijerph16101680.
8. Zhang, X.; Lin, Z. Hormesis-induced gap between the guidelines and reality in ecological risk assessment. *Chemosphere* **2020**, *243*, 125348. https://doi.org/10.1016/j.chemosphere.2019.125348.

9. Shah, S.Q.A.; Khan, F.Z.; Ahmad, M. Mitigating TCP SYN flooding based EDOS attack in cloud computing environment using binomial distribution in SDN. *Comput. Commun.* **2022**, *182*, 198–211. https://doi.org/10.1016/j.comcom.2021.11.008.
10. García-García, J.I.; Fernández Coronado, N.A.; Arredondo, E.H.; Imilpán Rivera, I.A. The Binomial Distribution: Historical Origin and Evolution of Its Problem Situations. *Mathematics* **2022**, *10*, 2680. https://doi.org/10.3390/math10152680.
11. Kachitvichyanukul, V.; Schmeiser, B.W. Binomial Random Variate Generation. *Commun. ACM* **1988**, *31*, 216–222. https://doi.org/10.1145/42372.42381.
12. Kachitvichyanukul, V.; Schmeiser, B.W. Algorithm 678: BTPEC: Sampling from the Binomial Distribution. *ACM Trans. Math. Softw.* **1989**, *15*, 394–397. https://doi.org/10.1145/76909.76916.
13. Relles, D.A. A Simple Algorithm for Generating Binomial Random Variables When N is Large. *J. Am. Stat. Assoc.* **1972**, *67*, 612–613. https://doi.org/10.1080/01621459.1972.10481259.
14. Ahrens, J.H.; Dieter, U. Computer methods for sampling from gamma, beta, poisson and bionomial distributions. *Computing* **1974**, *12*, 223–246. https://doi.org/10.1007/BF02293108.
15. Knuth, D.E. *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*, 3rd ed.; Addison Wesley: Hoboken, NJ, USA, 1998.
16. Cicirello, V.A. $\rho\mu$: A Java library of randomization enhancements and other math utilities. *J. Open Source Softw.* **2022**, *7*, 4663.1–4663.4. https://doi.org/10.21105/joss.04663.
17. Steele, G. JEP 356: Enhanced Pseudo-Random Number Generators. OpenJDK. 2017. Available online: https://openjdk.org/jeps/356 (accessed on).
18. Cicirello, V.A. JavaPermutationTools: A Java Library of Permutation Distance Metrics. *J. Open Source Softw.* **2018**, *3*, 950.1–950.4. https://doi.org/10.21105/joss.00950.
19. Cicirello, V.A. Chips-n-Salsa: A Java Library of Customizable, Hybridizable, Iterative, Parallel, Stochastic, and Self-Adaptive Local Search Algorithms. *J. Open Source Softw.* **2020**, *5*, 2448.1–2448.4. https://doi.org/10.21105/joss.02448.
20. Flury, B.D. Acceptance–Rejection Sampling Made Easy. *SIAM Rev.* **1990**, *32*, 474–476. https://doi.org/10.1137/1032082.
21. Steele, G.L.; Lea, D.; Flood, C.H. Fast Splittable Pseudorandom Number Generators. In Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications, New York, NY, USA, 20–24 Octobe 2014; pp. 453–472. https://doi.org/10.1145/2660193.2660195.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.