Fast Random Integer Generation in an Interval

Lukas Geis ✉
Goethe Universität Frankfurt am Main

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent convallis orci arcu, eu mollis dolor. Aliquam eleifend suscipit lacinia. Maecenas quam mi, porta ut lacinia sed, convallis ac dui. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse potenti.

2012 ACM Subject Classification Mathematics of computing → Random number generation

Keywords and phrases Random Number Generation, Rejection Method, Randomized Algorithms, Algorithm Engineering

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Supplementary Material Sources of implementations and experiments can be accessed at `https://github.com/lukasgeis/FRIGiaI-Review`

1 Introduction

1.1 Motivation

The study of randomized algorithms goes as far back as the 1910s [6]. While initially small, this study is now scientifically pursued in almost all parts of computer science. Not only in theory, but also in practice, randomized algorithms gain even more importance today to not only solve specific algorithmic problems but also to create large, scalable, and unbiased data sets that can be used in a variety of fields.
The core of such algorithms is most often the pseudo-random number generator (PRNG). There are many efficient PRNGs such as XorShift [9], MersenneTwister [10], linear congruential generators [8, 12, 13], and younger generators such as the PCG-Family [11] and ChaCha [4]. These generators produce 32-bit or 64-bit words which we can interpret as unsigned integers in $[0, 2^{32})$ and $[0, 2^{64})$ respectively. However, for most applications we do not want to have such a large range of possible random values but instead want to confine the range to a given interval $[a, b]$. This can not only be used to uniformly sample an element from an array but also for more complex algorithms

- The Fisher-Yates random shuffle [7] (see Algorithm 1) uniformly permutes an array of n elements in time $\Theta(n)$. We have to draw a random integer in an interval for every shuffled element.
- A random graph model describes a distribution over a (parametrized) family of graphs. There exist an uncountable number of such models and almost all of them require some sort of integer sampling in an interval. For example, the $\mathcal{G}(n, m)$ model [5] uniformly samples a (directed) graph with n nodes and m edges. A possible algorithm for this problem is Reservoir-Sampling [14](see Algorithm 2) which is a general technique that can sample k elements without replacement from an input stream. Another model is the BA model [1, 2] which iteratively builds the graph by adding nodes with edges to previously added nodes (see Algorithm 3). Both models require sampling of an integer in an interval for each of its edges (or more).
- An Alias-Table [15] can be used to simulate static discrete distributions in practice. After creating and initializing the table, we can sample in (expected) time $\Theta(1)$ by drawing a uniform random row and offset in each round. Hence, we need to sample an integer uniformly at random in $[0, n)$ where n is the number of rows in the table.

Algorithm 1 Fisher-Yates-Shuffle: uniformly shuffle an array of size n

Require: source of uniformly-distrbuted random integers in $[0, i]$ for parameter i
Require: array A of size n
for $i = n-1, \ldots, 1$ **do**
  $j \leftarrow$ random integer in $[0, i]$
  swap $A[i]$ and $A[j]$

Algorithm 2 Sampling of (directed) $\mathcal{G}(n, m)$ graphs using Reservoir Sampling

Require: source of uniformly-distrbuted random integers in $[0, i]$ for parameter i
Require: number of nodes n and number of edges $m \leq n^2$
$E \leftarrow$ array (of edges) of size m
for $i = 0, \ldots, m-1$ **do**
  $E[i] \leftarrow (i \div n, i \mod n)$
for $i = m, \ldots, n^2$ **do**
  $j \leftarrow$ random integer in $[0, i]$
  if $j < m$ **then**
    $E[j] \leftarrow (i \div n, i \mod n)$
return E

Algorithm 3 Preferential-Attachment-Generator [3]: sample BA-graphs in linear time

Require: source of uniformly-distrbuted random integers in $[0, i]$ for parameter i
Require: inital edge-list E_0 with n_0 nodes and $m_0 := |E_0|$ edges, number of additional nodes N, neighbor-parameter ν
$M \leftarrow$ array of size $2(m_0 + \nu N)$
for $i = 0, \ldots, m_0 - 1$ **do**
  $(M[2i], M[2i+1]) \leftarrow E_0[i]$
for $i = 0, \ldots, N-1$ **do**
  for $j = 1, \ldots, \nu$ **do**
    $k \leftarrow$ random integer in $[0, 2(m_0 + i\nu) - 1]$
    $M[2(m_0 + i\nu)] \leftarrow M[k]$
    $M[2(m_0 + i\nu) + 1] \leftarrow n_0 + i$
$E \leftarrow$ array (of edges) of size $(m_0 + \nu N)$
for $i = 0, \ldots, m_0 + \nu N - 1$ **do**
  $E[i] \leftarrow (M[2i], M[2i+1])$
return E

1.2 Preliminaries

We denote by $x \div y$ the integer division operation, namely $\lfloor x/y \rfloor$, and the remainder operation by $x \bmod y \equiv x - (x \div y)y$. Divisions by a power of two can be efficiently implemented by Bit-RightShift denoted by $\gg$: $x \div 2^W \equiv x \gg W$. Similarly, we can efficiently compute the remainder of a division py a power of two by a logical And-operation, denoted by &:

$x \bmod 2^W \equiv x \ \& \ (2^W - 1)$. Typically, PRNGs generate uniform random numbers in an interval $[0, 2^W)$ where $W \in \{32, 64\}$, namely a uniform random W-bit number.

Since we can map every integer in an interval $[a, b)$ to the interval $[0, b-a)$ by subtracting a, our goal is to generate a uniform random integer in an interval $[0, s)$ for a given s. If we map all integers x in $[0, 2^W)$ with the function $x \bmod s$, we get the following sequence:

$$\overbrace{\underbrace{0, 1, \ldots, s-1}_{s \text{ values}}, \underbrace{0, 1, \ldots, s-1}_{s \text{ values}}, \ldots, \underbrace{0, 1, \ldots, s-1}_{s \text{ values}}}^{\left(2^W \div s\right) \cdot s \text{ values}}, \underbrace{0, 1, \ldots, \left(2^W \bmod s\right) - 1}_{2^W \bmod s \text{ values}}}^{2^W \text{ values}}$$

This motivates the following lemma:

▶ **Lemma 1** (Remainder Multiples). *For given integers $a, b, s > 0$ with $a < b$, there exist exactly $(b-a) \div s$ integers $x \in [a, b)$ for every $0 \leq r < s$ such that $x \bmod s \equiv r$ whenever s divides $b - a$.*

1.3 Related Work

1.4 Structure

We present and analyse existing techniques for uniformly sampling random integers in an interval $[0, s)$ in Section 2. In Section 3, we present the main contribution of this paper: a nearly-divisionless algorithm to sampler uniform random integers in an interval $[0, s)$. In Section 4, we provide a short summary.

2 Existing Techniques

2.1 Biased

2.1.1 Modulo Reduction

2.1.2 Multiply-and-Shift

2.1.3 Floating-Point Conversion

2.2 Unbiased

2.2.1 OpenBSD

2.2.2 Java

2.2.3 Flips

2.2.4 Bitmask with Rejection

3 Nearly Divisionless Unbiased Sampling

4 Conclusion

References

1 Albert-László Barabási. *Network Science Book*. Network Science, 2014.

2 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. URL: https://www.science.org/doi/abs/10.1126/science.286.5439.509, arXiv:https://www.science.org/doi/pdf/10.1126/science.286.5439.509, doi:10.1126/science.286.5439.509.

3 Vladimir Batagelj and Ulrik Brandes. Efficient generation of large random networks. *Phys. Rev. E*, 71:036113, Mar 2005. URL: https://link.aps.org/doi/10.1103/PhysRevE.71.036113, doi:10.1103/PhysRevE.71.036113.

4 Daniel J. Bernstein. Chacha, a variant of salsa20. Technical report, The University of Illinois at Chicago, 2008.

5 Paul L. Erdos and Alfréd Rényi. On random graphs. i. *Publicationes Mathematicae Debrecen*, 2022. URL: https://api.semanticscholar.org/CorpusID:253789267.

6 Walter Gautschi, editor. *Mathematics of Computation 1943–1993: a half-century of computational mathematics*, pages 504, "Perhaps Pocklington also deserves credit as the inventor of the randomized algorithm". American Mathematical Society, Providence, 1994.

7 Donald E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, Boston, third edition, 1997.

8 Pierre L'Ecuyer. Tables of linear congruential generators of different sizes and good lattice structure. *Math. Comput.*, 68:249–260, 01 1999. doi:10.1090/S0025-5718-99-00996-5.

9 George Marsaglia. Xorshift rngs. *Journal of Statistical Software*, 8(14):1–6, 2003. URL: https://www.jstatsoft.org/index.php/jss/article/view/v008i14, doi:10.18637/jss.v008.i14.

10 Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, jan 1998. doi:10.1145/272991.272995.

11 Melissa E. O'Neill. Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation. Technical Report HMC-CS-2014-0905, Harvey Mudd College, Claremont, CA, September 2014.

12 S. K. Park and K. W. Miller. Random number generators: Good ones are hard to find. *Commun. ACM*, 31(10):1192–1201, oct 1988. doi:10.1145/63039.63042.

13 W. H. Payne, J. R. Rabung, and T. P. Bogyo. Coding the lehmer pseudo-random number generator. *Commun. ACM*, 12(2):85–86, feb 1969. doi:10.1145/362848.362860.

14 Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, mar 1985. doi:10.1145/3147.3165.

15 Alastair J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Trans. Math. Softw.*, 3(3):253–256, sep 1977. doi:10.1145/355744.355749.