The Fast Loaded Dice Roller: A Near-Optimal Exact Sampler for Discrete Probability Distributions

Feras A. Saad MIT EECS
Cameron E. Freer MIT BCS
Martin C. Rinard MIT EECS
Vikash K. Mansinghka MIT BCS

Abstract

This paper introduces a new algorithm for the fundamental problem of generating a random integer from a discrete probability distribution using a source of independent and unbiased random coin flips. We prove that this algorithm, which we call the Fast Loaded Dice Roller (FLDR), is highly efficient in both space and time: (i) the size of the sampler is guaranteed to be linear in the number of bits needed to encode the input distribution; and (ii) the expected number of bits of entropy it consumes per sample is at most 6 bits more than the information-theoretically optimal rate. We present fast implementations of the linear-time preprocessing and near-optimal sampling algorithms using unsigned integer arithmetic. Empirical evaluations on a broad set of probability distributions establish that FLDR is 2x–10x faster in both preprocessing and sampling than multiple baseline algorithms, including the widely-used alias and interval samplers. It also uses up to 10000x less space than the information-theoretically optimal sampler, at the expense of less than 1.5x runtime overhead.

1 INTRODUCTION

The problem of generating a discrete random variable is as follows: given a probability distribution $p := (p_1, \dots, p_n)$ and access to a random source that outputs an independent stream of fair bits, return integer i with probability p_i. A classic theorem from Knuth and Yao (1976) states that the most efficient sampler, in terms of the expected number of random bits consumed from the source, uses between $H(p)$ and $H(p) + 2$ bits in expectation, where $H(p) := \sum_{i=1}^{n} p_i \log(1/p_i)$ is the Shannon entropy of p. This entropy-optimal sampler is obtained by building a decision tree using the binary expansions of the p_i.

Despite the fact that the Knuth and Yao algorithm provides the most time-efficient sampler for any probability distribution, this paper shows that its construction can require exponentially larger space than the number of bits needed to encode the input instance p and may thus be infeasible to construct in practice. In light of this negative result, we aim to develop a sampling algorithm whose entropy consumption is close to the optimal rate and whose space scales polynomially.

This paper presents a new sampling algorithm where, instead of using an entropy-optimal sampler to simulate $(p_1, \dots, p_n)$ directly, we define a proposal distribution $(q_1, \dots, q_n, q_{n+1})$ on an extended domain whose probabilities q_i are dyadic rationals that are "close" to the probabilities p_i and then simulate the proposal with an entropy-optimal sampler followed by an accept/reject step. We prove that this sampling algorithm, which we call the Fast Loaded Dice Roller (FLDR), is efficient in both space and time: its size scales linearly in the number of bits needed to encode the input instance p and it consumes between $H(p)$ and $H(p)+6$ bits in expectation, which is near the optimal rate and does not require exponential memory.

We present an implementation of FLDR using fast integer arithmetic and show empirically that it is 2x–10x faster than several exact baseline samplers, and uses up to 10000x less space than the entropy-optimal sampler of Knuth and Yao. To the best of our knowledge, this paper presents the first theoretical characterization and practical implementation of using entropy-optimal proposal distributions for accept-reject sampling, as well as benchmark measurements that highlight the space and runtime benefits of FLDR over multiple existing exact sampling algorithms. A prototype implementation in C is released with the paper.

The remainder of this paper is structured as follows.

Proceedings of the 23rdInternational Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108. Copyright 2020 by the author(s).

Section 2 formally introduces the random bit model of computation for studying the sampling algorithms used throughout the paper. Section 3 establishes the worst-case exponential space of the entropy-optimal Knuth and Yao sampler. Section 4 presents a systematic study of the space–time complexity of three common baseline rejection algorithms. Section 5 presents FLDR and establishes its linear memory and near-optimal entropy consumption. Section 6 presents measurements of the preprocessing time, sampling time, and memory consumption of FLDR and demonstrates improvements over existing exact samplers.

2 PRELIMINARIES

Algebraic model Many algorithms for sampling discrete random variables (Walker, 1977; Vose, 1991; Smith, 2002; Bringmann and Panagiotou, 2017) operate in a model of computation where the space–time complexity of both preprocessing and sampling are analyzed assuming a real RAM model (Blum et al., 1998) (i.e., storing and arithmetically manipulating infinitely precise numbers can be done in constant time (Devroye, 1986, Assumptions I, III)). Algorithms in this model apply a sequence of transformations to a uniform random variable $U \in [0, 1]$, which forms the basic unit of randomness (Devroye, 1986, Assumption II). While often useful in practice, this model does not permit a rigorous study of either the complexity, entropy consumption, or sampling error of different samplers. More specifically, real RAM sampling algorithms typically generate random variates which are only approximately distributed according to the target distribution when implemented on physically-existing machines due to limited numerical precision, e.g., IEEE double-precision floating-point (Bringmann and Friedrich, 2013). This sampling error is challenging to quantify in practice (Devroye, 1982; Monahan, 1985). In addition, the real RAM model does not account for the complexity of drawing and manipulating the random variable U from the underlying source (a single uniform random variate has the same amount of entropy as countably infinitely many such variates) and thus ignores a key design constraint for samplers.

Random bit model This paper focuses on exact sampling (i.e., with zero sampling error) in a word RAM model of computation where the basic unit of randomness is an independent, unbiased bit $B \in \{0, 1\}$ returned from a primitive operation Flip. The random bit model is widely used, both in information theory (Han and Verdú, 1993) and in formal descriptions of sampling algorithms for discrete distributions that use finite precision arithmetic and random fair bits. Examples include the uniform (Lumbroso, 2013), discrete Gaussian (Folláth, 2014), geometric (Bringmann and Panagiotou, 2017), random graph (Blanca and Mihail, 2012), and general categorical (Knuth and Yao, 1976; Uyematsu and Li, 2003) distributions. The model has also been generalized to the setting of using a biased or non-i.i.d. source of coin flips for sampling (von Neumann, 1951; Elias, 1972; Blum, 1986; Roche, 1991; Peres, 1992; Abrahams, 1996; Pae and Loui, 2006; Kozen and Soloviev, 2018).

Problem Formulation Given a list $(a_1, \dots, a_n)$ of n positive integers which sum to m and access to a stream of independent fair bits (i.e., Flip), sample integer i with probability a_i/m $(i = 1, \dots, n)$.

Designing algorithms and data structures for this problem of "dice rolling" has received widespread attention in the computer science literature; see Schwarz (2011) for a survey. We next describe a framework for describing the computational behavior of any sampling algorithm implemented in the random bit model.

Discrete distribution generating trees Knuth and Yao (1976) present a computational framework for expressing any sampling algorithm in the random bit model in terms of a (possibly infinite) rooted binary tree T, called a *discrete distribution generating* (DDG) tree, which has the following properties: (i) each internal node has exactly two children (i.e., T is full); and (ii) each leaf node is labeled with one outcome from the set $\{1, 2, \dots, n\}$. The algorithm is as follows: starting at the root, obtain a random bit $B \sim \text{Flip}$. Proceed to the left child if $B = 0$ and proceed to the right child if $B = 1$. If the child node is a leaf, return the label assigned to that leaf and halt. Otherwise, draw a new random bit B and repeat the process. For any node $x \in T$, let $l(x)$ denote its label and $d(x)$ its level (by convention, the root is at level 0 and all internal nodes are labeled 0). Since Flip returns fair bits, the output probability distribution $(p_1, \dots, p_n)$ is

$$p_i \coloneqq \mathbb{P}[T \text{ returns } i] = \sum_{x \mid l(x)=i} 2^{-d(x)} \quad (i = 1, \dots, n).$$

The number of coin flips L_T used when simulating T is, in expectation, the average depth of the leaves, i.e.,

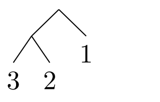
$$\mathbb{E}[L_T] = \sum_{x \mid l(x)>0} d(x) 2^{-d(x)}.$$

The operators $\mathbb{P}$ and $\mathbb{E}$ are defined over the sequence $\mathbf{b} \in \{0, 1\}^\infty$ of bits from the random source, finitely many of which are consumed during a halting execution (which occurs with probability one). The following classic theorem establishes tight bounds on the minimal expected number of bits consumed by any sampling algorithm for a given distribution p, and provides an explicit construction of an optimal DDG tree.

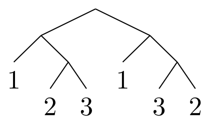
Theorem 2.1 (Knuth and Yao (1976))**.** *Let $p := (p_1, \dots, p_n)$, where $n > 1$. Any sampling algorithm with DDG tree T and output distribution p whose expected number of input bits is minimal (among all trees T' whose output distribution equals p) satisfies $H(p) \le \mathbb{E}[L_T] < H(p) + 2$. These bounds are the tightest possible. In addition, T contains exactly 1 leaf node labeled i at level j if and only if $p_{ij} = 1$, where $(0.p_{i1}p_{i2}\dots)_2$ denotes the binary expansion of each p_i (which ends in $\bar{0}$ whenever p_i is dyadic).*

We now present examples of DDG trees.

Example 2.2. Let $p := (1/2, 1/4, 1/4)$. By Thm. 2.1, an entropy-optimal DDG tree for p can be constructed directly from the binary expansions of the p_i, where p_{ij} corresponds to the jth bit in the binary expansion of p_i ($i = 1, 2, 3; j \ge 0$). Since $p_1 = (0.10)_2$ and $p_2 = p_3 = (0.01)_2$ are all dyadic, the entropy-optimal tree has three levels, and the sampler always halts after consuming at most 2 bits. Also shown is an entropy-suboptimal tree for p, which always halts after consuming at most 3 bits.

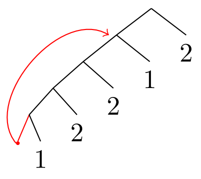


Optimal DDG tree

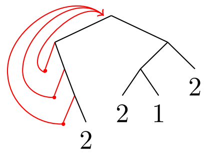


Suboptimal DDG tree

Example 2.3. Let $p := (3/10, 7/10)$. Although p_1 and p_2 have infinite binary expansions, they are rational numbers which can be encoded using a finite prefix and a bar atop a finite repeating suffix; i.e., $p_1 = (0.0\overline{1001})_2, p_2 = (0.1\overline{0110})_2$. While any DDG tree for p has infinitely many levels, it can be finitely encoded by using back-edges (shown in red). The entropy-optimal tree has five levels and a back-edge from level 4 to level 1, corresponding to the binary expansions of the p_i, where the suffixes have four digits and prefixes have one digit.



Optimal DDG tree



Suboptimal DDG tree

Definition 2.4 (Depth of a DDG tree)**.** Let T be a DDG tree over $\{1, \dots, n\}$ with output distribution $(p_1, \dots, p_n)$, where each $p_i \in \mathbb{Q}$. We say that T has depth k if the longest path from the root node to any leaf node in the shortest finite tree encoding of T (using back-edges, as in Example 2.2) consists of k edges.

In this paper, we do not consider distributions with irrational entries, as their DDG trees are infinite and cannot be finitely encoded. Thm. 2.1 settles the problem of constructing the most "efficient" sampler for a target distribution, when efficiency is measured by the expected number of bits consumed.

However, designing an entropy-efficient sampler that is also space-efficient remains an open problem. In particular, as we show in Section 3, the size of the optimal DDG tree T is exponentially larger than the number of bits needed to encode p and is therefore often infeasible to construct in practice. Knuth and Yao (1976) allude to this issue, saying "most of the algorithms which achieve these optimum bounds are very complex, requiring a tremendous amount of space".

3 COMPLEXITY OF ENTROPY-OPTIMAL SAMPLING

This section recounts background results from Saad et al. (2020, Section 3) about the class of entropy-optimal samplers given in Thm. 2.1. These results establish the worst-case exponential space of entropy-sampling and formally motivate the need for space-efficient and near-optimal samplers developed in Section 5. For completeness, the proofs are presented in Appendix A.

For entropy-optimal DDG trees that have depth $k \ge 1$ (Definition 2.4), the output probabilities are described by a fixed-point k-bit number. The fixed-point k-bit numbers x are those such that for some integer l satisfying $0 \le l \le k$, there is an element $(x_1, \dots, x_k) \in \{0,1\}^l \times \{0,1\}^{k-l}$, where the first l bits correspond to a finite prefix and the final $k - l$ bits correspond to an infinitely repeating suffix, i.e., $x = (0.x_1 \dots x_l\overline{x_{l+1} \dots x_k})_2$. Write $\mathbb{B}_{kl}$ for the set of rationals in $[0, 1]$ describable in this way.

Proposition 3.1. *For integers k and l with $0 \le l \le k$, define $Z_{kl} := 2^k - 2^l \mathbf{1}_{l<k}$. Then*

$$\mathbb{B}_{kl} = \left\{ \frac{0}{Z_{kl}}, \frac{1}{Z_{kl}}, \dots, \frac{Z_{kl}-1}{Z_{kl}}, \frac{Z_{kl}}{Z_{kl}} \mathbf{1}_{l<k} \right\}.$$

The next result establishes that the number systems $\mathbb{B}_{kl}$ ($k \in \mathbb{N}$, $0 \le l \le k$) from Prop. 3.1 describe the output probabilities of optimal DDG trees with depth-k.

Theorem 3.2. *Let T be an entropy-optimal DDG tree with a non-degenerate output distribution $(p_i)_{i=1}^n$ for $n > 1$. The depth of T is the smallest integer k such that there exists an integer $l \in \{0, \dots, k\}$ for which all the p_i are integer multiples of $1/Z_{kl}$ (hence in $\mathbb{B}_{kl}$).*

Corollary 3.3. *Every back-edge in an entropy-optimal depth-k DDG tree originates at level $k-1$ and ends at the same level l, where $0 \le l < k - 1$.*

The next result, Thm. 3.4, implies that an entropy-optimal DDG tree for a coin with weight $1/m$ has depth at most $m-1$. Thm. 3.5 shows that this bound is tight for many m, and Rem. 3.6 notes that it is likely tight for infinitely many m.

Theorem 3.4. *Suppose p is defined by $p_i = a_i/m$ $(i = 1, \dots, n)$, where $\sum_{i=1}^{n} a_i = m$. The depth of any entropy-optimal sampler for p is at most $m-1$.*

Theorem 3.5. *Let p be as in Thm. 3.4. If m is prime and 2 is a primitive root modulo m, then the depth of an entropy-optimal DDG tree for p is $m-1$.*

Remark 3.6. The bound in Thm. 3.4 is likely the tightest possible for infinitely many m. Assuming Artin's conjecture, there are infinitely many primes m for which 2 is a primitive root, which by Thm. 3.5 implies any entropy-optimal DDG tree has depth m.

Holding n fixed, the tight upper bound m on the depth of an entropy-optimal DDG tree for any distribution having an entry $1/m$ is thus exponentially larger (in m) than the $n\log(m)$ bits needed to encode the input instance (each of $a_1, \dots, a_n$ requires a word of size at least $\log(m)$ bits). Fig. 2 shows a plot of the scaling characteristics from Thm. 3.4 and provides evidence for the tightness conjectured in Rem. 3.6.

4 REJECTION SAMPLING

We now present several alternative algorithms for exact sampling based on the rejection method (Devroye, 1986, II.3), which lead to the Fast Loaded Dice Roller presented in Section 5. Rejection sampling operates as follows: given a *target distribution* $p := (p_1, \dots, p_n)$ and *proposal distribution* $q := (q_1, \dots, q_l)$ (with $n \le l$), first find a *rejection bound* $A > 0$ such that $p_i \le Aq_i$ $(i = 1, \dots, n)$. Next, sample $Y \sim q$ and flip a coin with weight $p_Y/(Aq_Y)$ (where $p_{n+1} = \dots = p_l = 0$): if the outcome is heads accept Y, otherwise repeat. The probability of halting in any given round is:

$$\Pr\left[\mathsf{Bernoulli}\left(\frac{p_Y}{Aq_Y}\right) = 1\right] = \sum_{i=1}^{n} p_i/(Aq_i)\, q_i = 1/A.$$

The number of trials thus follows a geometric distribution with rate $1/A$, whose mean is A. We next review common implementations of random-bit rejection samplers and their space–time characteristics. All algorithms take n positive integers $(a_1, \dots, a_n)$ and the sum m as input, and return i with probability a_i/m.

Uniform Proposal Consider the uniform proposal distribution $q := (1/n, \dots, 1/n)$. Set $D := \max_i(a_i)$ and set $A := Dn/m$, which gives a tight rejection bound since $p_i \le \max_i(p_i) = D/m = (Dn/m)(1/n) = Aq_i$, so that i is accepted with probability a_i/D $(i = 1, \dots, n)$. Alg. 1 presents an implementation where (i) simulating the uniform proposal (line 3), and (ii) accepting/rejecting the proposed sample (line 4), are both achieved using the two entropy-optimal samplers in Lumbroso (2013) for uniform and Bernoulli generation. The only extra storage needed by Alg. 1 is in computing the maximum D during preprocessing (line 1). For runtime, $A = nD$ trials occur on average; each trial uses $\log n$ bits for sampling $\mathsf{Uniform}(n)$ and 2 bits for sampling $\mathsf{Bernoulli}(a_i/D)$ on average. The entropy is therefore order $n(m-n)\log n \ge n\log n \gg \log n$ bits. Thus, despite its excellent space and preprocessing characteristics, the method can be exponentially wasteful of bits.

Algorithm 1 Rejection sampler (uniform)

```
   // PREPROCESS
1: Let $D \leftarrow \max(a_1, \dots, a_n)$;
   // SAMPLE
2: while true do
3:     $i \sim$ FastDiceRoller($n$); (Lumbroso (2013, p. 4))
4:     $x \sim$ Bernoulli($a_i, D$); (Lumbroso (2013, p. 21))
5:     if ($x = 1$) then return $i$;
```

Dyadic Proposal Consider the following proposal distribution. Let $k \in \mathbb{N}$ be such that $2^{k-1} < m \le 2^k$ (i.e., $k - 1 < \log(m) \le k$ so that $k = \lceil \log m \rceil$) and set

$$q := (a_1/2^k, \dots, a_n/2^k, 1 - m/2^k). \qquad (1)$$

The tightest rejection bound $A = 2^k/m$, since $p_i = a_i/m = (2^k/m)a_i/2^k = Aq_i$ $(i = 1, \dots, n)$ and $p_{n+1} = 0 \le (2^k - m)/2^k = q_{n+1}$. Thus, i is always accepted when $1 \le i \le n$ and always rejected when $i = n + 1$.

Lookup-table Implementation. Devroye (1986) implements the rejection sampler with proposal Eq. (1) using a length-m lookup table T, which has exactly a_i elements labeled i $(i = 1, \dots, n)$, shown in Alg. 2. The sampler draws k random bits $(b_1, \dots, b_k)$, forms an integer $W := \sum_{i=1}^{k} b_i 2^{i-1}$, and returns $T[W]$ if $0 \le W < m$ or repeats if $m \le W \le 2^k - 1$. For fixed n, the $m \log m$ space required by T is exponentially larger (in m) than the $n \log m$ bits needed to encode the input. Further, the number of bits per trial is always k, so $k2^k/m \ge k \approx \log m$ bits are used on average, which (whenever $n \ll m$) can be much higher than the optimal rate, which is at most $H(p) + 2 \le \log n + 2$.

Algorithm 2 Rejection sampler (dyadic + lookup table)

```
   // PREPROCESS
1: Let $k \leftarrow \lceil \log m \rceil$;
2: Make size-$m$ table $T$ with $a_i$ entries $i$ $(i = 1, \dots, n)$;
   // SAMPLE
3: while true do
4:     Draw $k$ bits, forming integer $W \in \{0, \dots, 2^{k-1}\}$;
5:     if ($W < m$) then return $T[W]$;
```

Binary Search Implementation. The exponential memory of the lookup table in Alg. 2 can be eliminated by inversion sampling the proposal Eq. (1) using binary search on the cumulative frequencies, as shown in Alg. 3. This algorithm consumes the same number of bits k as Alg. 2. Its exponential improvement in space from $m \log m$ to $n \log m$ introduces a logarithmic runtime factor the inner loop of the sampler, i.e., line 5 of Alg. 3 sometimes uses $\Omega(\log n)$ time as opposed to the constant indexing time from line 5 of Alg. 2, representing a typical space–runtime tradeoff.

Algorithm 3 Rejection sampler (dyadic + binary search)

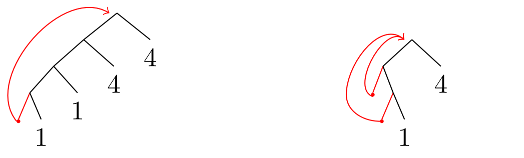
```
   // PREPROCESS
1: Let $k \leftarrow \lceil \log m \rceil$;
2: Define array $T$, where $T[j] \coloneqq \sum_{i=1}^{j} a_i$ $(j = 1, \dots, n)$
   // SAMPLE
3: while true do
4:     Draw $k$ bits, forming integer $W \in \{0, \dots, 2^{k-1}\}$;
5:     if $(W < m)$ then return $\min\{j \mid W < T[j]\}$;
```

5 FAST LOADED DICE ROLLER

Section 4 shows that for rejection sampling using the dyadic proposal Eq. (1), a lookup table requires exponential memory and constant lookup time, whereas binary search uses linear memory but $\log n$ lookup time. Moreover, these methods use k bits/sample, which is highly wasteful for low-entropy distributions. The key idea of the Fast Loaded Dice Roller (FLDR) presented in this section is to eliminate these memory, runtime, and entropy inefficiencies by simulating the proposal distribution q using an entropy-optimal sampler.

Algorithm 4 Fast Loaded Dice Roller (sketch)

1. Let $k \coloneqq \lceil \log m \rceil$ and define the proposal distribution $q \coloneqq (a_1/2^k, \dots, a_n/2^k, 1 - m/2^k)$.
2. Simulate $X \sim q$, using an entropy-optimal sampler as described in Thm. 2.1.
3. If $X \le n$, then return X, else go to Step 2.



(a) Optimal DDG tree (b) FLDR DDG tree

Figure 1: Comparison of DDG trees for $p = (1/5, 4/5)$.

Fig. 1 shows a comparison of an entropy-optimal DDG tree and a FLDR DDG tree. We next establish the linear space and near-optimal entropy of Alg. 4.

Theorem 5.1. *The DDG tree T of FLDR in Alg. 4 has at most $2(n+1)\lceil \log m \rceil$ nodes.*

Proof. Suppose the DDG tree T_q of the entropy-optimal sampler for q in Step 2 of Alg. 4 has N total nodes, $s < N$ leaf nodes, and depth k. Since T_q is a full binary tree it has $N-1$ edges. Moreover, the root has degree two, the s leaves have degree one, and the $N - s - 1$ internal nodes have degree three. Equating the degrees and solving $2(N-1) = 2+s+3(N-s-1)$ gives $N = 2s - 1$. Next, since q is a dyadic distribution over $\{1, \dots, n+1\}$ with base $\lceil \log m \rceil$, T_q has depth $k = \lceil \log m \rceil$ (Thm. 3.2). From the entropy-optimality of the depth-k tree T_q over $\{1, \dots, n+1\}$, we have $s \le (n+1)k$, since each of the k levels has at most 1 leaf node labeled i $(i = 1, \dots, n+1)$ (Thm. 2.1). Thus $N = 2s - 1 \le 2(n+1)k - 1 \le 2(n+1)\lceil \log m \rceil$. Finally, the DDG tree T of FLDR is identical to T_q, except for additional back-edges from each leaf node labeled $n+1$ to the root (i.e., the rejection branch when $X = n+1$ in Step 3). $\square$

Theorem 5.2. *The DDG tree T of FLDR in Alg. 4 satisfies*

$$0 \le \mathbb{E}[L_T] - H(p) < 6. \tag{2}$$

Proof. Let T_q be an entropy-optimal DDG tree for the proposal distribution q defined in Step 1, so that $\mathbb{E}[L_{T_q}] = H(q) + t_q$ for some t_q satisfying $0 \le t_q < 2$ (by Thm. 2.1). Since the expected number of trials of Alg. 4 is $2^k/m$ and the number of trials is independent of the bits consumed in each round, we have $\mathbb{E}[L_T] = (2^k/m)\mathbb{E}[L_{T_q}]$.

If $m = 2^k$ then $p = q$, and we have $\mathbb{E}[L_T] - H(p) = t_q$, so Eq. (2) holds. Now suppose $m < 2^k$. Then

$$\begin{aligned}
&\mathbb{E}[L_T] - H(p) \\
&= (2^k/m)(H(q) + t_q) - H(p) \\
&= (2^k/m)H(q) - H(p) + 2^k t_q/m \\
&= 2^k/m\Big[\textstyle\sum_{i=1}^{n} a_i/2^k \log(2^k/a_i) \\
&\qquad\qquad + (2^k - m)/2^k \log(2^k/(2^k - m))\Big] \\
&\quad - \textstyle\sum_{i=1}^{n} a_i/m \log(m/a_i) + 2^k t_q/m \\
&= \textstyle\sum_{i=1}^{n} a_i/m[\log(2^k/a_i) - \log(m/a_i)] \\
&\quad + (2^k - m)/m \log(2^k/(2^k - m)) + 2^k t_q/m \\
&= \log(2^k/m) + (2^k - m)/m \log(2^k/(2^k - m)) \qquad (3) \\
&\quad + 2^k t_q/m.
\end{aligned}$$

We now bound Eq. (3) under our restriction $2^{k-1} < m < 2^k$. All three terms are monotonically decreasing in $m \in \{2^{k-1}, \dots, 2^k - 1\}$, hence maximized when $m = 2^{k-1}+1$, achieving a value less than that for $m = 2^{k-1}$.

Hence the first term is less than $\log(2^k/2^{k-1}) = 1$, the second term less than

$$\frac{(2^k - 2^{k-1})}{2^{k-1}} \log\left(\frac{2^k}{2^k - (2^{k-1})}\right) = \log\left(\frac{2^k}{2^{k-1}}\right) = 1,$$

and the third term less than $2t_q < 4$. All three terms are positive, thus establishing bound Eq. (2). $\square$

Thms. 5.1 and 5.2 together imply that Alg. 4 uses $O(n \log m)$ space on a size $n \log m$ input instance and guarantees an entropy gap of at most 6 bits sample, for any target distribution p. Fig. 2 compares the asymptotic scaling of the size of the FLDR DDG tree from Thm. 5.1 with that of the entropy-optimal sampler, and Fig. 3 decomposes the entropy gap from Thm. 5.2 according to the three terms in Eq. (3).

Alg. 5 provides one of many possible implementations of FLDR (sketched in Alg. 4) that uses unsigned integer arithmetic to preprocess and sample an encoding of the underlying DDG tree. This algorithm uses two data structures to eliminate the $O(n)$ inner-loop of the DDG tree sampler in Roy et al. (2013, Alg. 1) (at the expense of more memory), where array h stores the number of leaf nodes at each level and matrix H stores their labels in increasing order. (A sparse matrix can often be used for H, as most of its entries are zero.) Alternative DDG tree preprocessing and sampling algorithms that operate on an explicit tree data structure can be found in Saad et al. (2020, Section 5).

Algorithm 5 Implementation of the Fast Loaded Dice Roller using unsigned integer arithmetic

Input: Positive integers $(a_1, \dots, a_n)$, $m := \sum_{i=1}^n a_i$.
Output: Random integer i with probability a_i/m.

```
    // PREPROCESS
 1: $k \leftarrow \lceil \log(m) \rceil$;
 2: $a_{n+1} \leftarrow 2^k - m$;
 3: initialize $h$ int$[k]$;
 4: initialize $H$ int$[n+1][k]$;
 5: for $j = 0, \dots, k-1$ do
 6:     $d \leftarrow 0$;
 7:     for $i = 1, \dots, n+1$ do
 8:         bool $w \leftarrow (a_i >> (k-1) - j))$ & $1$;
 9:         $h[j] \leftarrow h[j] + w$;
10:         if $w$ then
11:             $H[d, j] \leftarrow i$;
12:             $d \leftarrow d + 1$;
    // SAMPLE
13: $d \leftarrow 0, c \leftarrow 0$;
14: while true do
15:     $b \sim$ FLIP();
16:     $d \leftarrow 2 \cdot d + (1 - b)$;
17:     if $d < h[c]$ then
18:         if $H[d, c] \leq n$ then
19:             return $H[d, c]$;
20:         else {$d \leftarrow 0$; $c \leftarrow 0$;}
21:     else {$d \leftarrow d - h[c]$; $c \leftarrow c + 1$;}
```

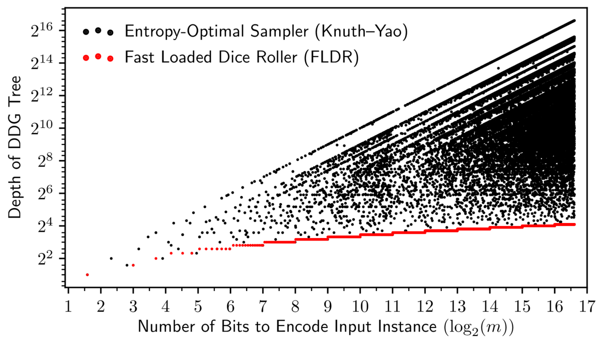


Figure 2: Depth of DDG tree for a distribution having an entry $1/m$, using the Knuth and Yao entropy-optimal sampler (black) and FLDR (red) for $m = 3, \dots, 10^5$ (computed analytically). The y-axis is on a logarithmic scale: the entropy-optimal sampler scales exponentially (Thm. 3.5) and FLDR scales linearly (Thm. 5.1).

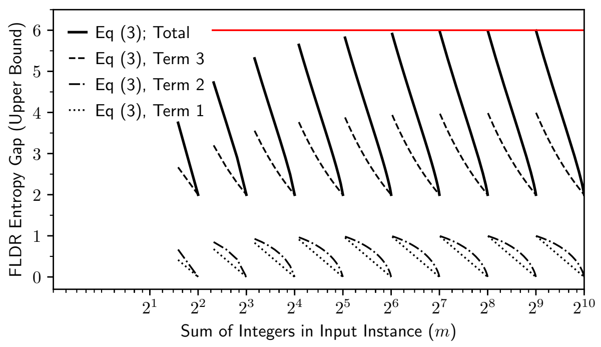


Figure 3: Plot of the three terms in Eq. (3) in the entropy gap (y-axis) from Thm. 5.2, for varying m (x-axis).

6 EMPIRICAL EVALUATION

We next empirically evaluate the memory, runtime, preprocessing, and entropy properties of the Fast Loaded Dice Roller from Section 5 and compare them to the following six baseline algorithms which, like FLDR, all produce exact samples from the target distribution and operate in the random bit model:

(i) entropy-optimal sampler (Knuth and Yao, 1976), using a variant of Alg. 5 (lines 13–21);

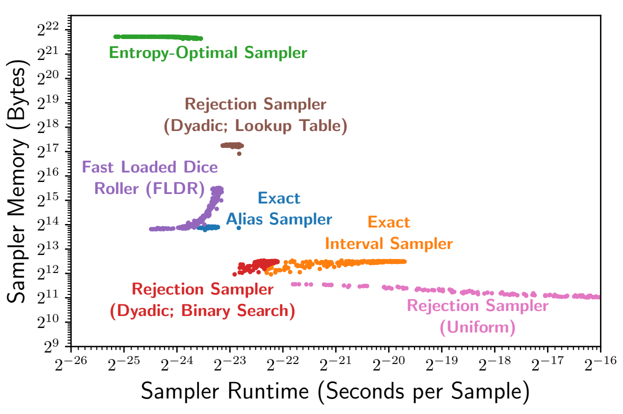
(ii) rejection sampler with uniform proposal (Alg. 1);

(iii) rejection sampler with dyadic proposal (Devroye, 1986), using a lookup table (Alg. 2);

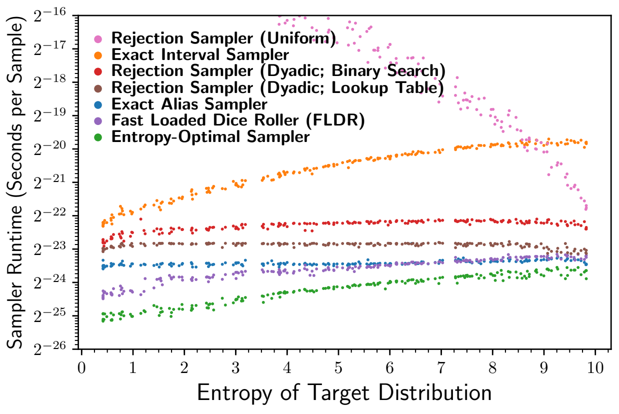
(iv) rejection sampler with dyadic proposal (Devroye, 1986), using binary search (Alg. 3);

(v) exact interval sampler (Han and Hoshi, 1997), using Alg. 1 of Devroye and Gravel (2015);

(vi) exact alias sampler (Walker, 1977), using entropy-optimal uniform and Bernoulli sampling (Lumbroso, 2013) and the one-table implementation (Vose, 1991).



(a)



(b)

Figure 4: Comparison of memory and runtime performance for sampling 500 random frequency distributions over $n=1000$ dimensions with sum $m=40000$, using FLDR and six baseline exact samplers. (a) shows a scatter plot of the sampler runtime (x-axis; seconds per sample) versus sampler memory (y-axis; bytes); and (b) shows how the sampler runtime varies with the entropy of the target distribution, for each method and each of the 500 distributions.

All algorithms were implemented in C and compiled with `gcc` level 3 optimizations, using Ubuntu 16.04 on AMD Opteron 6376 1.4GHz processors.[1]

6.1 Sampler Memory and Runtime

We defined 100 frequency distributions $(a_1, \dots, a_n)$ over $n = 100$ dimensions which sum to $m = 40000$, randomly chosen with entropies equally spaced from 0.034 to $6.64 \approx \log 100$ bits. For each sampling algorithm and each distribution, we measured (i) the size of the data structure created during preprocessing; and (ii) the wall-clock time taken to generate one million random samples. Fig. 4a shows a scatter plot of the sampler memory (y-axis, in bytes) and sampler runtime (x-axis, in seconds per sample) for each algorithm and for each of the 100 distributions in the benchmark set, and Fig. 4b shows a scatter plot of the sampler runtime (y-axis, in seconds per sample) with the entropy of that target distribution (x-axis, in bits).

The runtime of FLDR (purple) most closely follows the runtime of the optimal sampler (green), while using up to 16000x less memory—the memory improvement of FLDR grows at an exponential rate as m increases (Fig. 2.4). In addition, for low-entropy distributions (bottom-left part of purple curve), FLDR uses even less memory than the linear bound from Thm. 5.1.

The lookup table rejection sampler (brown) uses up to 256x more memory and is up to 4x slower than FLDR, since it draws a constant $k = 16$ bits/sample and uses a large size-m table—the memory improvement of FLDR again grows at an exponential rate as m increases. The binary search rejection sampler (red) uses up to 32x less than FLDR since it only stores running sums, but has up to 16x slower runtime due to the cost of binary search—this runtime factor grows at a logarithmic rate as n increases. Rejection sampling with a uniform proposal (pink) performs poorly at low-entropy distributions (many rejections) and moderately at higher entropies where the target distribution is more uniform.

It is worthwhile to note that the Han and Hoshi interval sampler (orange) has a tighter theoretical upper bound on entropy gap than FLDR (3 bits versus 6 bits). However, FLDR is up to 16x faster in our experiments, since we can directly simulate the underlying DDG tree using Alg. 5. In contrast, implementations of the interval sampler in the literature for unbiased sources do not sample the underlying DDG tree, instead using expensive integer divisions and binary search in the main sampling loop (Han and Hoshi, 1997; Uyematsu and Li, 2003; Devroye and Gravel, 2015). In addition, the array on which binary search is performed changes dynamically over the course of sampling. To the best of our knowledge, unlike with FLDR, there is no existing implementation of interval sampling that directly simulates the underlying DDG tree so as to fully leverage its entropy efficiency.

The alias method (blue) is the most competitive baseline, which is up to 2x slower than FLDR (at low entropies) while using between 1x (at low-entropy distributions) and 8x less memory (at high entropies) to store the alias table. While the alias method is commonly said to require constant runtime, this analysis

[1]All experiments in this section use target distributions with integer weights. We note that the reference implementations of FLDR in C and Python additionally contain preprocessing algorithms for exact sampling given IEEE 754 floating-point weights. All samplers and experiments are at https://github.com/probcomp/fast-loaded-dice-roller.

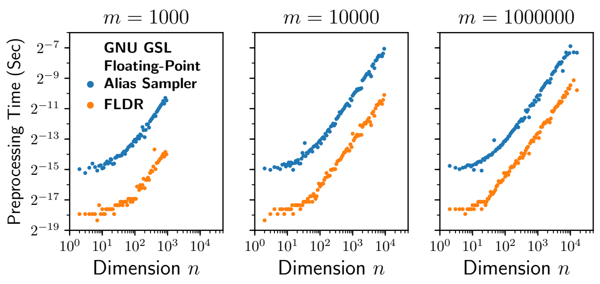


Figure 5: Comparison of the preprocessing times (y-axes; wall-clock seconds) of FLDR with those of the alias sampler, for distributions with dimension ranging from $n = 10^0, \dots 2 \times 10^4$ (x-axes) and normalizers $m = 1000$, 10000, and 1000000 (left, center, and right panels, respectively).

only holds in the real RAM model and typical floating-point implementations of the alias method have non-zero sampling error. For producing exact samples in the random bit model, the alias method requires (on average) between $\log n$ and $\log n + 1$ bits to sample a uniform over $\{1, \dots, n\}$ and two bits to sample a Bernoulli, which gives a total of $\log n + 3$ bits/sample, independently of $H(p)$ (horizontal blue line in Fig. 4b). In contrast, FLDR requires at most $H(p) + 6$ bits on average, which is less than alias sampling whenever $H(p) \ll \log n$. For fixed n, the constant rate of the alias sampler corresponds to the "worst-case" runtime of FLDR: in Fig. 4b, the gap between purple (FLDR) and blue (alias) curves is largest at lower entropies and narrows as $H(p)$ increases.

6.2 Preprocessing Time

We next compared the preprocessing time of FLDR (Alg. 5, lines 1–12) for varying (n, m) with that of the alias sampler (Walker, 1977), which is the most competitive baseline method. To measure the preprocessing time of the alias method, we used the open-source implementation in the C GNU Scientific Library (GSL)[2]. Fig. 5 shows a log-log plot of the preprocessing time (y-axis; wall-clock seconds) and dimensions (x-axis; n) for distributions with $m = 1000$, 10000, 1000000 (panels left to right). Our C implementation of FLDR (orange) has a lower preprocessing time than the GSL alias sampler (blue) in all these regimes. Since the matrix H constructed during FLDR preprocessing has $n + 1$ rows and $\log m$ columns, the gap between the two curves narrows (at a logarithmic rate) as m increases. On a 64-bit architecture we may assume that $m < 2^{64}$ (i.e., `unsigned long long` in C) and so the $n \log m \approx 64n$ preprocessing time of FLDR is highly scalable, growing linearly in n.

[2]The `gsl_ran_discrete_preproc` function from the `gsl_randist` GSL library implements the $O(n)$ alias table preprocessing algorithm from Vose (1991).

Table 1: Number of PRNG calls and wall-clock time when drawing 10^6 samples from $n = 1000$ dimensional distributions, using FLDR & approximate floating-point samplers.

Method	Entropy (bits)	Number of PRNG Calls	PRNG Wall Time (ms)
FLDR	1	123,607	3.69
	3	182,839	4.27
	5	258,786	5.66
	7	325,781	7.90
	9	383,138	8.68
Floating Point	all	1,000,000	21.51

6.3 Calls to Random Number Generator

This paper has emphasized exact sampling in the random bit model, where the sampling algorithm lazily draws a random bit $B \sim \textsc{Flip}$ on demand. As discussed in Section 2, most sampling algorithms in existing software libraries operate under the real RAM model and approximate an ideal uniform variate $U \sim \mathsf{Uniform}([0, 1])$ using a high-precision floating-point number. Floating-point samplers produce non-exact samples—both as U is not exactly uniform and as arithmetic operations involving U (such as division) are non-exact. Further, these implementations can be highly wasteful of computation. (As an illustrative example, sampling a fair coin requires only one random bit, but comparing $U < 0.5$ in floating-point consumes a full machine word, e.g., 64 pseudo-random bits, to generate U.) Following Lumbroso (2013), our implementation of $\textsc{Flip}$ maintains a buffer of 64 pseudo-random bits. Table 1 shows a comparison of the number of calls to the pseudo-random number generator (PRNG) and wall-clock time for generating 10^6 samples from 1000-dimensional distributions with various entropies, using FLDR and floating-point samplers (we have conservatively assumed that the latter makes exactly one PRNG call per sample). The results in Table 1 highlight that, by calling the PRNG nearly as many times as is information-theoretically optimal (Thm. 5.2), FLDR spends significantly less time calling the PRNG than do floating-point samplers (with the added benefit of producing exact samples).

7 CONCLUSION

This paper has presented the Fast Loaded Dice Roller, a new method for generating discrete random variates. The sampler has near-optimal entropy consumption, uses a linear amount of storage, and requires linear setup time. Due to its theoretical efficiency, ease-of-implementation using fast integer arithmetic, guarantee of generating exact samples, and high performance in practice, we expect FLDR to be a valuable addition to the suite of existing sampling algorithms.

References

Julia Abrahams. 1996. Generation of Discrete Distributions from Biased Coins. *IEEE Trans. Inf. Theory* 42, 5 (Sept. 1996), 1541–1546. https://doi.org/10.1109/18.532895

Antonio Blanca and Milena Mihail. 2012. Efficient Generation ϵ-close to $G(n, p)$ and Generalizations. (April 2012). arXiv:1204.5834

Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. 1998. *Complexity and Real Computation*. Springer-Verlag, New York.

Manuel Blum. 1986. Independent Unbiased Coin Flips from a Correlated Biased Source: A Finite State Markov Chain. *Combinatorica* 6, 2 (June 1986), 97–108. https://doi.org/10.1007/BF02579167

Karl Bringmann and Tobias Friedrich. 2013. Exact and Efficient Generation of Geometric Random Variates and Random Graphs. In *ICALP 2013: Proceedings of the 40th International Colloquium on Automata, Languages and Programming* (Riga, Latvia). Lecture Notes in Computer Science, Vol. 7965. Springer, Heidelberg, 267–278. https://doi.org/10.1007/978-3-642-39206-1_23

Karl Bringmann and Konstantinos Panagiotou. 2017. Efficient Sampling Methods for Discrete Distributions. *Algorithmica* 79, 2 (Oct. 2017), 484–508. https://doi.org/10.1007/s00453-016-0205-0

Luc Devroye. 1982. A Note on Approximations in Random Variate Generation. *J. Stat. Comput. Simul.* 14, 2 (1982), 149–158.

Luc Devroye. 1986. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York.

Luc Devroye and Claude Gravel. 2015. Sampling with Arbitrary Precision. (Feb. 2015). arXiv:1502.02539

Peter Elias. 1972. The Efficient Construction of an Unbiased Random Sequence. *Ann. Math. Stat.* 43, 3 (June 1972), 865–870. https://doi.org/10.1214/aoms/1177692552

János Folláth. 2014. Gaussian Sampling in Lattice Based Cryptography. *Tatra Mount. Math. Pub.* 60, 1 (Sept. 2014), 1–23. https://doi.org/10.2478/tmmp-2014-0022

Te Sun Han and Mamoru Hoshi. 1997. Interval Algorithm for Random Number Generation. *IEEE Trans. Inf. Theory* 43, 2 (March 1997), 599–611. https://doi.org/10.1109/18.556116

Te Sun Han and Sergio Verdú. 1993. Approximation Theory of Output Statistics. *IEEE Trans. Inf. Theory* 39, 3 (May 1993), 752–772. https://doi.org/10.1109/18.256486

Donald E. Knuth and Andrew C. Yao. 1976. The Complexity of Nonuniform Random Number Generation. In *Algorithms and Complexity: New Directions and Recent Results*, Joseph F. Traub (Ed.). Academic Press, Inc., Orlando, FL, 357–428.

Dexter Kozen and Matvey Soloviev. 2018. Coalgebraic Tools for Randomness-Conserving Protocols. In *RAMiCS 2018: Proceedings of the 17th International Conference on Relational and Algebraic Methods in Computer Science* (Groningen, The Netherlands). Lecture Notes in Computer Science, Vol. 11194. Springer, Cham, 298–313. https://doi.org/10.1007/978-3-030-02149-8_18

Jérmie Lumbroso. 2013. Optimal Discrete Uniform Generation from Coin Flips, and Applications. (April 2013). arXiv:1304.1916

John F. Monahan. 1985. Accuracy in Random Number Generation. *Math. Comput.* 45, 172 (Oct. 1985), 559–568. https://doi.org/10.2307/2008146

Sung-il Pae and Michael C Loui. 2006. Randomizing Functions: Simulation of a Discrete Probability Distribution Using a Source of Unknown Distribution. *IEEE Trans. Inf. Theory* 52, 11 (Nov. 2006), 4965–4976. https://doi.org/10.1109/TIT.2006.883555

Yuval Peres. 1992. Iterating von Neumann's Procedure for Extracting Random Bits. *Ann. Stat.* 20, 1 (March 1992), 590–597. https://doi.org/10.1214/aos/1176348543

James R. Roche. 1991. Efficient Generation of Random Variables from Biased Coins. In *ISIT 1991: Proceedings of the IEEE International Symposium on Information Theory* (Budapest, Hungary). IEEE Press, Piscataway, 169–169. https://doi.org/10.1109/ISIT.1991.695225

Sinha S. Roy, Frederik Vercauteren, and Ingrid Verbauwhede. 2013. High Precision Discrete Gaussian Sampling on FPGAs. In *SAC 2013: Proceedings of the 20th International Conference on Selected Areas in Cryptography* (Burnaby, Canada). Lecture Notes in Computer Science, Vol. 8282. Springer, Berlin, 383–401. https://doi.org/10.1007/978-3-662-43414-7_19

Feras A. Saad, Cameron E. Freer, Martin C. Rinard, and Vikash K. Mansinghka. 2020. Optimal Approximate Sampling from Discrete Probability Distributions. *Proc. ACM Program. Lang.* 4, POPL, Article 36 (Dec. 2020), 31 pages. https://doi.org/10.1145/3371104

Keith Schwarz. 2011. Darts, Dice, and Coins. Retrieved Oct 5, 2019 from http://www.keithschwarz.com/darts-dice-coins/

Warren D. Smith. 2002. *How To Sample from a Probability Distribution.* Technical Report DocNumber17. NEC Research.

Tomohiko Uyematsu and Yuan Li. 2003. Two Algorithms for Random Number Generation Implemented by Using Arithmetic of Limited Precision. *IEICE Trans. Fund. Elec. Comm. Comp. Sci* 86, 10 (Oct. 2003), 2542–2551.

John von Neumann. 1951. Various Techniques Used in Connection with Random Digits. In *Monte Carlo Method*, A. S. Householder, G. E. Forsythe, and H. H. Germond (Eds.). National Bureau of Standards Applied Mathematics Series, Vol. 12. U.S. Government Printing Office, Washington, DC, Chapter 13, 36–38.

Michael D. Vose. 1991. A Linear Algorithm for Generating Random Numbers with a Given Distribution. *IEEE Trans. Softw. Eng.* 17, 9 (Sept. 1991), 972–975. https://doi.org/10.1109/32.92917

Alastair J. Walker. 1977. An Efficient Method for Generating Discrete Random Variables with General Distributions. *ACM Trans. Math. Softw.* 3, 3 (Sept. 1977), 253–256. https://doi.org/10.1145/355744.355749