Exact and Efficient Generation of Geometric Random Variates and Random Graphs

Karl Bringmann[1,⋆] and Tobias Friedrich[2]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany
[2] Friedrich-Schiller-Universität Jena, Germany

Abstract. The standard algorithm for fast generation of Erdős-Rényi random graphs only works in the Real RAM model. The critical point is the generation of geometric random variates $\mathrm{Geo}(p)$, for which there is no algorithm that is both exact and efficient in any bounded precision machine model. For a RAM model with word size $w = \Omega(\log\log(1/p))$, we show that this is possible and present an exact algorithm for sampling $\mathrm{Geo}(p)$ in optimal expected time $\mathcal{O}(1 + \log(1/p)/w)$. We also give an exact algorithm for sampling $\min\{n, \mathrm{Geo}(p)\}$ in optimal expected time $\mathcal{O}(1 + \log(\min\{1/p, n\})/w)$. This yields a new exact algorithm for sampling Erdős-Rényi and Chung-Lu random graphs of n vertices and m (expected) edges in optimal expected runtime $\mathcal{O}(n + m)$ on a RAM with word size $w = \Theta(\log n)$.

1 Introduction

Random Graph Generation. A large fraction of empirical research on graph algorithms is performed on random graphs. Random graph generation is also commonly used for simulating networking protocols on the Internet topology and the spread of epidemics (or rumors) on social networks (e.g. [16]) It is also an important tool in real world applications such as detecting motifs in biological networks (e.g. [21]). We focus on homogenous and inhomogenous random graphs and consider Erdős-Rényi [9] and Chung-Lu graphs [5]. The key ingredient for generating such graphs with n vertices faster than the obvious $\Theta(n^2)$ algorithm is an efficient algorithm for exact sampling of geometric random variates.

Efficient Random Variate Generation. Non-uniform random variates are typically generated from random variates that are uniformly distributed on $[0, 1]$. With the introduction of Intel's Ivy Bridge microarchitecture with built-in hardware digital random number generation, we can assume that we have fast access to a stream of high quality random bits. However, most non-uniform random variate generation algorithms [8, 26] assume a Real RAM, which can manipulate real numbers. This assumption is highly problematic as real numbers are

⋆ Karl Bringmann is a recipient of the *Google Europe Fellowship in Randomized Algorithms*, and this research is supported in part by this Google Fellowship.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 267–278, 2013.

infinite objects and all physical computers can only handle finite portions of these objects. Typical implementations, with e.g. double floating point precision, are efficient, but *not exact* (e.g. in C++11); that is, some outcomes might not be reachable and others might become more likely than they should.

Exact Random Variate Generation. Knuth and Yao [18] initiated the study of exact nonuniform random number generation. They discuss the power of various restricted machine models. Several authors [10, 11, 18] provided additional results along these lines, but only presented efficient algorithms for single distributions. There also exist implementations of exact and efficient random number generators for exponential and normal distributions [15]. For parameterized families of distributions such as the geometric distribution, one can study the expected asymptotic runtime in the parameter. However, in this regard there are no exact and efficient algorithms known on any bounded precision machine model. For sampling $\mathrm{Geo}(p)$, the trivial algorithm of repeatedly sampling a coin with Bernoulli distribution $\mathrm{Ber}(p)$ until it falls heads up has expected runtime $\mathcal{O}(1/p)$, which is *not efficient* for p close to 0.

Exact and Efficient Random Variate Generation. Our aim is the design of exact and efficient algorithms for random variate generation. We show that this is possible in many cases and give a particularly fast algorithm for geometric random variates. This allows exact and efficient generation of Erdős-Rényi and Chung-Lu random graphs. It also allows exact and efficient generation of very large non-uniform random variates (e.g. for cryptographic applications [13]), which has been open so far.

Related Work on Random Graph Generation. There is a large body of work on generating random regular graphs (e.g. [17]), graphs with a prescribed degree distribution (e.g. [3]), and graphs with a prescribed joint degree distribution (e.g. [23]). All these algorithms converge to the desired distribution for $n \to \infty$. Note that this typically implies for finite n that only an approximation of the true distribution is reached.

The most studied random graph model is certainly the Erdős-Rényi [9] random graph $\mathcal{G}(n,p)$, where each edge of a graph of n vertices is present independently and uniformly with probability $p \in [0,1]$. Many experimental papers use algorithms with runtime $\Theta(n^2)$ to draw from $\mathcal{G}(n,p)$. The reason for this is probably that most graph algorithm software libraries such as JUNG, LEDA, BGL, and JDSL also do not contain efficient random graph generators. However, there are several algorithms which can sample from $\mathcal{G}(n,p)$ in expected time $\mathcal{O}(m+n)$ on a Real RAM, where $m = \Theta(pn^2)$ is the expected number of edges [1, 20]. This is done by using the fact that in an ordered list of all $\Theta(n^2)$ pairs of vertices the distance between two consecutive edges is geometrically distributed. The resulting distribution is *not exact* if the algorithm is run on a physical computer, which can only handle bounded precision, as it ignores the bias introduced by fixed-length number representations. The available implementation in the library NetworkX [14] therefore also does not return the desired distribution exactly. It is not obvious how to get an exact implementation even by using

algebraic real numbers [19] and/or some high accuracy floating-point representation. The problem of sampling $\mathcal{G}(n,p)$ on a bounded precision model has been studied by Blanca and Mihail [2]. They showed how to achieve an approximation of the desired distribution efficiently. Our aim is an *exact and efficient* generation on a bounded precision model instead.

Generating Random Geometric Distributions. As discussed above, all previous algorithms to sample a geometric random variate which are efficient on a Real RAM become inexact when implemented on a bounded precision machine. We assume the more realistic model of a Word RAM with word size w that can sample a random word in constant time; for a definition of the machine models see Section 2. We first observe the following lower bound for any exact sampling algorithm.

Theorem 1. *On a RAM with word size w, any algorithm sampling a geometric random variate* $\mathrm{Geo}(p)$ *with parameter $p \in (0,1)$ needs at least expected runtime $\Omega(1 + \log(1/p)/w)$.*

Theorem 1 follows from Lemma 1, which shows that the expected output size is $\Omega(\log(1/p))$ bits. To get a first upper bound we translate the well-known inversion method (typically used on a Real RAM [8]) to our bounded precision model by using multi-precision arithmetic, obtaining the following result.

Theorem 2. *On a RAM with word size $w = \Omega(\log\log(1/p))$, a geometric random variate* $\mathrm{Geo}(p)$ *with parameter $p \in (0,1)$ can be sampled in expected runtime $\mathcal{O}(1 + \log(1/p)\,\mathrm{poly}\log\log(1/p)/w)$.*

To the best of our knowledge, this observation is not discussed in the literature so far. However, as the following Theorem 3 is strictly stronger, we defer the presentation of the inversion method and Theorem 2 to the full version of this paper. It not only applies to geometric distributions, but to all distributions where the inverse of the cumulative distribution is efficiently computable on a Word RAM. The assumption on w is needed to handle pointers to an array as large as the expected output size in constant time. This result is independent of the rest of the paper and demonstrates that the classical inversion method *does not give an optimal runtime* matching Theorem 1, since this algorithm, as well as many other approaches, *does not avoid taking logarithms*. Note that it is a long-standing open problem in analytic number theory and computational complexity whether the logarithm can be computed in linear time.

Our aim is a Word RAM algorithm which returns the exact geometric distribution in optimal runtime. In Section 3 we give a simple algorithm for this and prove the following theorem. Note that our algorithm also works for *bitstreams* p, see Section 2.

Theorem 3. *On a RAM with word size $w = \Omega(\log\log(1/p))$, a geometric random variate* $\mathrm{Geo}(p)$ *with parameter $p \in (0,1)$ can be sampled in expected runtime $\mathcal{O}(1 + \log(1/p)/w)$, which is optimal.*

Observe that, as a sample of a geometric random variate can be arbitrarily large, the aforementioned sampling algorithm cannot work in bounded worst-case time

or space. Also note that on a parallel machine with P Word RAM processors the runtime decreases to $\mathcal{O}(1 + \log(1/p)/(wP))$.

Generating Bounded Random Geometric Distributions. In the full version of this paper we extend this to sampling bounded geometric random distributions $\mathrm{Geo}(p, n) = \min\{n, \mathrm{Geo}(p)\}$ and observe the following lower bound.

Theorem 4. *On a RAM with word size w, any algorithm sampling a bounded geometric random variate $\mathrm{Geo}(p, n) = \min\{n, \mathrm{Geo}(p)\}$ with parameters $n \in \mathbb{N}$ and $p \in (0, 1)$ needs at least expected runtime $\Omega(1 + \log(\min\{1/p, n\})/w)$.*

We present an algorithm which achieves this optimal runtime bound and prove the following theorem.

Theorem 5. *On a RAM with word size $w = \Omega(\log\log(1/p))$, a bounded geometric random variate $\mathrm{Geo}(p, n) = \min\{n, \mathrm{Geo}(p)\}$ with parameters $n \in \mathbb{N}$ and $p \in (0, 1)$ can be sampled in expected runtime $\mathcal{O}(1 + \log(\min\{1/p, n\})/w)$, which is optimal.*

If p is a rational number with numerator and denominator fitting in $\mathcal{O}(1)$ words, then this algorithm needs $\mathcal{O}(n)$ space in the worst case.

If p is a bitstream, we cannot bound the worst-case space usage of a sampling algorithm for $\mathrm{Geo}(n, p)$ in general. However, if p is a rational with numerator and denominator fitting in a constant number of words of the Word RAM, Theorem 5 shows that this is indeed possible.

Random Graph Generation. We believe our new exact and efficient sampling algorithms for bounded and unbounded geometric distributions are of independent interest, but also present in Section 4 one particular application, which is the generation of random graphs. For generating graphs with n vertices it is natural to assume $w = \Omega(\log n)$.

Theorem 6. *On a RAM with word size $w = \Omega(\log n)$, the random graph $\mathcal{G}(n, p)$ can be sampled in expected time $\Theta(n + m)$, where $m = \Theta(pn^2)$ is the expected number of edges. This is optimal if $w = \mathcal{O}(\log n)$. If p is a rational number with numerator and denominator fitting in $\mathcal{O}(1)$ words, then the worst-case space complexity of the algorithm is asymptotically equivalent to the size of the output graph, which is optimal.*

A similar algorithm achieves optimal runtime for the more general Chung-Lu random graphs $\mathcal{G}(n, W)$ [5], generating a random graph with a given sequence of expected degrees.

Theorem 7. *Let $W = (W_1, \ldots, W_n)$ be rationals with common denominator, where each numerator and the common denominator fit in $\mathcal{O}(1)$ words. Then on a RAM with word size $w = \Omega(\log n)$, the random graph $\mathcal{G}(n, W)$ can be sampled in expected time $\Theta(n + m)$, where $m = \Theta(\sum_{i=1}^{n} W_i)$ is the expected number of edges. This is optimal if $w = \mathcal{O}(\log n)$. The worst-case space complexity of the algorithm is asymptotically equivalent to the size of the output graph, which is optimal.*

Both theorems follow from plugging our algorithm for sampling geometric random variables into the best algorithm known for sampling the respective graph class.

Note that due to tight space constraints many proofs, in particular for Theorems 2, 4, 5, and 7, are defered to the full version of this paper.

2 Preliminaries

Machine Models. We discuss two variants of random access machines (RAMs). Both are abstract computational machine models with an arbitrary number of registers that can be indirectly addressed. In the classic RAM, each register can contain an arbitrarily large natural number $\mathbb{N}_0$ and all basic mathematical functions can be performed in constant time.

The two models relevant for this paper are the Real RAM, which allows computation even with real numbers, and the Word RAM, which only allows computation with bounded precision. In addition to the standard definitions, we assume that a uniform random number can be sampled in constant time. As we are dealing with randomized algorithms and a sample of a geometric random variate can be arbitrarily large, we also allow potentially unbounded space usage[1].

The *Real RAM* is the main model of computability in computational geometry and is also used in numerical analysis. Here, each register can contain a real number in the mathematical sense. All basic mathematical functions including the logarithm of a real number can be computed in constant time. The disadvantage of the model is that real numbers are infinite objects and all physical computers can only handle finite portions of these objects.

The *Word RAM* is a more realistic model of computation. It is parameterized by a parameter w which determines the word length. The registers are called words and contain integers in the range $\{0, \ldots, 2^w - 1\}$. The execution of basic arithmetic instructions on words takes constant time; our algorithms only need constant time addition, subtraction and comparison, as well as constant time generation of random words. Long integers are represented by a string of words. Floating point numbers are represented by an exponent (a string of words of some length k) and a mantissa (a string of words of some length ℓ). Addition and multiplication can then be done in time $\mathcal{O}(\text{poly}(\ell, k))$ and with error $2^{-w\ell}$. Note that the Word RAM offers an intrinsic parallelism where, in constant time, an operation on w bits can be performed in parallel.

Random Graph Models. In the *Erdős-Rényi* [9] random graph model $G(n, p)$, each edge of an n vertex graph is independently present with probability p. This yields a binomial degree distribution and approaches a Poisson distribution in the

[1] We assume that accessing the i-th memory cell costs $\mathcal{O}(1)$ although it might make more sense to assume cost proportional to the length of a pointer to i (which is $\Theta(1 + \log(i)/w)$) or larger. However, our results remain valid as long as this cost is $\mathcal{O}((2-\varepsilon)^{i/2^w})$ for some $\varepsilon > 0$.

limit. As many real-world networks have power-law degree distributions, we also study inhomogenous random graphs. We consider *Chung-Lu* [5] graphs $G(n, W)$ with n vertices and weights $W = (W_1, W_2, \dots, W_n) \in \mathbb{R}^n_{\geqslant 0}$. In this model, an edge between two vertices i and j is independently present with probability $p_{i,j} := \min\{W_i W_j / \sum_k W_k, 1\}$. For sufficiently large graphs, the expected degree of i converges to W_i. The related definitions of generalized random graph [25] with $p_{ij} = W_i W_j / (\sum_k W_k + W_i W_j)$ and Norros-Reittu random graphs [22] with $p_{i,j} = 1 - \exp(-W_i W_j / \sum_k W_k)$ can be handled in a similar way. However, we will focus on Chung-Lu random graphs.

Probability Distributions. Let $p \in (0, 1)$. The *Bernoulli* distribution $\mathrm{Ber}(p)$ takes values in $\{0, 1\}$ such that $\Pr[\mathrm{Ber}(p) = 1] = 1 - \Pr[\mathrm{Ber}(p) = 0] = p$. The *geometric* distribution $\mathrm{Geo}(p)$ takes values in $\mathbb{N}_0$ such that for any $i \in \mathbb{N}_0$, we have $\Pr[\mathrm{Geo}(p) = i] = p(1-p)^i$. For $n \in \mathbb{N}_0$, we define the *bounded geometric* distribution $\mathrm{Geo}(p, n)$ to be $\min\{n, \mathrm{Geo}(p)\}$. This means that $\mathrm{Geo}(p, n)$ takes values in $\{0, \dots, n\}$ such that for any $i \in \mathbb{N}_0$, $i < n$, we have $\Pr[\mathrm{Geo}(p, n) = i] = p(1-p)^i$, and $\Pr[\mathrm{Geo}(p, n) = n] = (1-p)^n$. The *uniform* distribution $\mathrm{Uni}[0, 1]$ takes values in $[0, 1]$ with uniform probability. For $n \in \mathbb{N}$, we define the uniform distribution $\mathrm{Uni}(n)$ to be the uniform distribution over $\{0, \dots, n-1\}$.

Input Model. We assume the input p to be given in the following form: We are given a number $k \in \mathbb{N}_0$ such that $2^{-k} \geqslant p > c2^{-k}$ for some fixed constant[2] $c > 0$. Moreover, for any $i \in \mathbb{N}$ we are able to compute a number $p_i \leqslant 1$ such that $|p_i - 2^k p| \leqslant 2^{-i}$. We can assume that p_i has at most $i + 1$ bits (otherwise take the first $i + 1$ bits of p_{i+1}, which are a 2^{-i}-approximation of $2^k p$). Since we assumed $w = \Omega(\log \log(1/p))$, k fits into $\mathcal{O}(1)$ words; this resembles the usual assumption that we can compute with numbers as large as the input/output size in constant time. Furthermore, we want to assume that p_i can be computed in time $\mathrm{poly}(i)$. This means that p can be approximated efficiently. However, it is sufficient even if the runtime is $\mathcal{O}((2-\varepsilon)^i)$ for some constant $\varepsilon > 0$. All numbers other than the input parameter p will be encoded as simple strings of words or floating point numbers, as discussed in the paragraph "machine models".

Notations. The base of all logarithms is 2. For integer division we use $a \operatorname{div} b := \lfloor a/b \rfloor$ for $a, b \in \mathbb{Z}$. We typically use x_i to denote the i-th bit (approximation) of x. We denote the set $\{1, \dots, n\}$ by $[n]$.

3 Sampling Geometric Random Variates

In this section we show a Word RAM algorithm for generating a geometric random variate $\mathrm{Geo}(p)$ in optimal expected runtime. We assume that the parameter p is given by a bitstream, i.e., we are given $k \in \mathbb{N}_0$ and can approximate p_* such that $p = 2^{-k} p_*$ and $c < p_* \leqslant 1$ for some $c > 0$. Approximating p_* with precision i needs time $\mathcal{O}((2-\varepsilon)^i)$ for some $\varepsilon > 0$. We first prove that the expected

[2] One could set $c = 1/2$, but our results hold more generally, in the case where we cannot compute such a good approximation of p.

output size is $\Theta(\log(1/p))$, which gives a lower bound of $\Omega(1 + \log(1/p)/w)$ for the expected runtime of any algorithm sampling $\mathrm{Geo}(p)$ on the Word RAM as (at most) w bits can be processed in parallel.

Lemma 1. *For any $p \in (0,1)$, we have $\mathbb{E}[\log(1+\mathrm{Geo}(p))] = \Theta(\log(1/p))$, where the lower bound holds for $1/p$ large enough.*

We now present an algorithm achieving this optimal expected runtime. The main trick is that we split up $\mathrm{Geo}(p)$ into $\mathrm{Geo}(p) \operatorname{div} 2^k$ and $\mathrm{Geo}(p) \bmod 2^k$. It is easy to see that both parts are independent random variables. Now, $\mathrm{Geo}(p) \operatorname{div} 2^k$ has constant expected value, so we can iteratively check whether it equals $0, 1, 2, \ldots$. On the other hand, $\mathrm{Geo}(p) \bmod 2^k$ is sufficiently well approximated by the uniform distribution over $\{0, \ldots, 2^k - 1\}$; the rejection method suffices for fast sampling. These ideas are brought together in Algorithm 1.

Algorithm 1. GenGeo(p) samples $\mathrm{Geo}(p)$ given a bitstream $p = 2^{-k} p_*$.

```
D ← 0
while Ber((1 − p)^{2^k}) do
    D ← D + 1
repeat
    M ←(lazy)— Uni(2^k)
until Ber((1 − p)^M)
fill up M with random bits
return 2^k D + M
```

Here, D represents $\mathrm{Geo}(p) \operatorname{div} 2^k$, initialized to 0. It is increased by 1 as long as a Bernoulli random variate $\mathrm{Ber}((1-p)^{2^k})$ turns out to be 1. Then M, corresponding to $\mathrm{Geo}(p) \bmod 2^k$, is chosen uniformly from the interval $\{0, \ldots, 2^k - 1\}$, but rejected with probability $(1-p)^M$. We sample M lazily, i.e., a bit of M is sampled only if needed by the test $\mathrm{Ber}((1-p)^M)$. After we leave the loop, M is filled up with random bits, so that we return the same value as if we had sampled M completely inside of the second loop. The result is, naturally, $2^k D + M$.

We will next discuss correctness of this algorithm, describe the details of how to implement it efficiently, and analyze its runtime. We postpone the issue of how to sample $\mathrm{Ber}((1-p)^n)$ to the end of this section. For the moment we will just assume that this can be done in expected constant time, looking at the first expected constant many bits of p and n.

Correctness. Let $n \geqslant 0$. The probability of outputting $n = 2^k D + M$ should be $p(1-p)^n$, i.e., it should be proportional to $(1-p)^n$. Following the algorithm step by step we see that the probability is

$$\underbrace{\left((1-p)^{2^k}\right)^D \cdot \left(1 - (1-p)^{2^k}\right)}_{\text{first loop}} \cdot \underbrace{\sum_{t \geqslant 0} \left(1 - \sum_{i=0}^{2^k-1} 2^{-k}(1-p)^i\right)^t 2^{-k}(1-p)^M}_{\text{second loop}},$$

where t is the number of iterations of the second loop; note that $2^{-k}(1-p)^i$ is the probability of outputting i in the first iteration of the second loop, so that $\sum_{i=0}^{2^k-1} 2^{-k}(1-p)^i$ is the probability of leaving the second loop after the first iteration. Collecting the factors dependent on D and M we see that this probability is proportional to $(1-p)^{2^k D+M} = (1-p)^n$, showing correctness of the algorithm.

Runtime. We show that the expected runtime of Algorithm 1 is $\mathcal{O}(1 + \log(1/p)/w)$. Again, assume that we can sample $\mathrm{Ber}((1-p)^n)$ in expected constant time. By the last section, incrementing the counter D can be done in amortized constant time, and we only need an expected constant number of bits of M during the second loop, after which we fill up M with random bits in time $\mathcal{O}(1 + \log(1/p)/w)$. Hence, if we show that the two loops run in expected constant time, then Algorithm 1 runs in expected time $\mathcal{O}(1 + \log(1/p)/w)$.

We consider the probabilities of dropping out of the two loops. Since $2^{-k} \geqslant p > c2^{-k}$, for the first loop this is

$$1 - (1-p)^{2^k} \geqslant 1 - (1-p)^{c/p} \geqslant 1 - e^{-c}, \tag{1}$$

so we have constant probability to drop out of this loop in every iteration. Moreover, the second loop terminates immediately if $k = 0$; otherwise we have

$$(1-p)^M \geqslant (1-p)^{2^k} \geqslant (1-2^{-k})^{2^k} \geqslant (1-1/2)^2 = 1/4, \tag{2}$$

so for the second loop we also have constant probability of dropping out.

To show that each loop runs in expected constant time, let T be a random variable denoting the number of iterations of the loop; note that $\mathbb{E}[T] = \mathcal{O}(1)$, since the probability of dropping out of each loop is $\Omega(1)$. Furthermore, let X_i be the runtime of the i-th iteration of the loop; note that by assumption we can sample $\mathrm{Ber}((1-p)^n)$ in expected constant time, so that $\mathbb{E}[X_i \mid T \geqslant i] = \mathcal{O}(1)$. The total runtime of the loop is $X_1 + \ldots + X_T$. Thus, the following lemma shows that the expected runtime of the loop is $\mathcal{O}(1)$. This finishes the proof of Theorem 3, aside from sampling $\mathrm{Ber}((1-p)^n)$.

Lemma 2. *Let T be a random variable with values in $\mathbb{N}_0$ and X_i, $i \in \mathbb{N}$, be random variables with values in $\mathbb{R}$; we assume no independence. Let $\alpha \in \mathbb{R}$ with $\mathbb{E}[X_i \mid T \geqslant i] \leqslant \alpha$ for all $i \in \mathbb{N}$. Then we have $\mathbb{E}[X_1 + \ldots + X_T] \leqslant \alpha \cdot \mathbb{E}[T]$.*

We remark that the above lemma is an easy special case of Wald's equation.

Note that the only points where this algorithm is using the Word RAM parallelism are when we fill up M and when we compute with exponents. The generation of $\mathrm{Ber}((1-p)^n)$, discussed in the remainder of this section, will use Word RAM parallelism only for working with exponents. The filling of M can be done in time $\mathcal{O}(1 + \log(1/p)/w)$ as we assumed that we can generate random words in unit time. Also note that given P processors, each one capable of performing Word RAM operations, we can trivially further parallelize this algorithm to run in expected time $\mathcal{O}\big(1 + \frac{\log(1/p)}{wP}\big)$.

Sampling Ber$((1-p)^n)$. It is left to show how to sample a Bernoulli random variable with parameter $(1-p)^n$. We can use the fact that we know k with $2^{-k} \geqslant p > c2^{-k}$ and can approximate $2^k p$ by p_i, and that $n \in \mathbb{N}$, $n \leqslant 2^k$. Note that we can easily get an approximation n_i of n of the form $|2^{-k}n - n_i| \leqslant 2^{-i}$ in the situation of Algorithm 1: In the first loop we have $n = 2^k$, then simply pick $n_i = 1$; in the second loop $n = M$ is uniform in $\{0, \ldots, 2^k - 1\}$, so that we get n_i by determining (i.e. flipping) the highest i bits of n. In this situation we can show the following lemma.

Lemma 3. *Given bitstream p with $2^{-k} \geqslant p = \Omega(2^{-k})$, for $n = 2^k$ or for uniformly random n in $\{0, \ldots, 2^k - 1\}$ we can sample* Ber$((1-p)^n)$ *in expected constant time.*

In the full version of this paper we discuss a method to sample Ber(q) in expected constant time which is closely related to Flajolet and Saheb [10].

Lemma 4. *A Bernoulli random variate* Ber(q) *with parameter $q \in (0,1)$ (given as a bitstream) can be sampled in constant expected runtime on a Word RAM.*

The only thing we need to efficiently sample Ber(q) is to be able to compute an approximation q_i of q with $|q - q_i| \leqslant 2^{-i}$ in time $\mathcal{O}((2-\varepsilon)^i)$. To get such an approximation for $(1-p)^n$, we make use of the binomial theorem $(1-p)^n = \sum_{j=0}^{n} \binom{n}{j}(-p)^j$. Noting that $\binom{n}{j} \leqslant \frac{n^j}{j!}$ and $n \leqslant 1/p$, we see that the j-th summand is absolutely bounded by $1/j!$. Moreover, the absolute value of the summands is monotonically decreasing in j, and their sign is $(-1)^j$, implying $\left|\sum_{j=i+2}^{n} \binom{n}{j}(-p)^j\right| \leqslant 1/(i+2)! \leqslant 2^{-i-1}$. Thus, by summing up only the first $i+2$ summands we get a good approximation of $(1-p)^n$.

Moreover, we have

$$\sum_{j=0}^{i+1} \binom{n}{j}(-p)^j = \frac{1}{(i+1)!} \sum_{j=0}^{i+1} (-p)^j \left(\prod_{h=j+1}^{i+1} h \right) \prod_{h=0}^{j-1} (n-h). \tag{3}$$

We will compute the right-hand side of this with working precision r. This means that we work with floating point numbers, with an exact exponent encoded by a string of words, and a mantissa which is a string of $\lceil r/w \rceil$ words. We get p and n up to working precision r by plugging in $2^{-k}p_r$ and $2^k n_r$. Then we calculate the numerator and denominator of the right-hand side independently with working precision r. Note that adding or multiplying the floating point numbers takes time $\mathcal{O}(\text{poly}(r))$ for adding/multiplying the mantissas (even using the school method for multiplication is fine for this), and $\mathcal{O}(1 + \log(i))$ for subtracting/adding the exponents, as all exponents in equation (3) are absolutely bounded by $\mathcal{O}(\text{poly}(i) \cdot k)$ and k fits in $\mathcal{O}(1)$ words.

Regarding runtime, noting that there are $\mathcal{O}(\text{poly}(i))$ operations to carry out in computing the right-hand side of equation (3), we see that we can compute the latter with working precision r in time $\mathcal{O}(\text{poly}(r, i))$. If we choose r large enough so that this yields an approximation of equation (3) with absolute error

at most 2^{-i-1}, then combined with the error analysis from using only the first $i+2$ terms, we get a runtime of $\mathcal{O}(\text{poly}(r,i))$ to compute an approximation of $(1-p)^n$ with absolute error 2^{-i}. Now, as long as we can choose $r = \text{poly}(i)$, this runtime is small enough to use Lemma 4, since we only needed an approximation of $(1-p)^n$ with absolute error 2^{-i} in time $\mathcal{O}((2-\varepsilon)^i)$ for some $\varepsilon > 0$. Under this assumption on r, we are done proving Lemma 3. The following lemma shows that $r = \text{poly}(i)$ is indeed sufficient.

Lemma 5. *The absolute error of computing equation* (3) *with working precision* $r = i + \alpha(1 + \log(i))$ *is at most* 2^{-i-1}*, for a large enough constant* α*.*

4 Generating Random Graphs

In this section we show that Erdős-Rényi and Chung-Lu random graphs can be efficiently generated. For this we simply take the efficient generation on Real RAMs from [1, 20] and replace the generation of bounded geometric variables by our algorithm from the last section. In the following we discuss why this is sufficient and leads to the runtimes claimed in Theorems 6 and 7.

Consider the original efficient generation algorithm of Erdős-Rényi random graphs described in [1], which is essentially the following. For each vertex $u \in [n]$ we want to sample its neighbors $v \in [u-1]$ in decreasing order. Defining $v_0 := u$, the first neighbor v_1 of u is distributed as $v_1 \sim v_0 - 1 - \text{Geo}(p, v_0 - 1)$, where the event $v_1 = 0$ represents that u has no neighbor. Then the next neighbor is distributed as $v_2 \sim v_1 - 1 - \text{Geo}(p, v_1 - 1)$ and so on. Sampling the graph in this way, we use $m + n$ bounded geometric variables, where m is the number of edges in the final graph (which is a random variable).

In this algorithm we have to cope with indices of vertices, thus, it is natural to assume $w = \Omega(\log n)$. Under this assumption, all single operations of the original algorithm can be performed in worst-case constant time on a Word RAM, except for the generation of bounded geometric variables $\text{Geo}(p,k)$, with $k \leqslant n$. The latter, however, can be done in expected time $\mathcal{O}(1+\log(\min\{n, 1/p\})/w) = \mathcal{O}(1)$ using our algorithm from Theorem 5. Hence, the expected runtime of the modified algorithm (with replaced sampling of bounded geometric variables) should be the same as that of the original algorithm. To prove this, consider the runtime the modified algorithm spends on sampling bounded geometric variables. This random variable can be written as $X_1 + \ldots + X_T$, where T is a random variable denoting the number of bounded geometric variables sampled by the algorithm, and X_i is the time spent on sampling the i-th such variable. Note that $\mathbb{E}[X_i \mid T \geqslant i] = \mathcal{O}(1)$. Thus, by Lemma 2 we can bound $\mathbb{E}[X_1 + \ldots + X_T]$ by $\mathcal{O}(\mathbb{E}[T])$. Since the original algorithm spends time $\Omega(T)$, the total expected runtime of the modified algorithm is asymptotically the same as the expected runtime of the original algorithm, namely $\mathcal{O}(n + pn^2)$. This runtime is optimal, as writing down a graph takes time $\Omega(n+m)$ (each index needs $\Theta(1)$ words; this depends, however, on the representation of the graph). Noting that the space requirements of the algorithm are met by Theorem 5, this proves Theorem 6.

A similar result applies to the more general Chung-Lu random graphs $\mathcal{G}(n, W)$. Again we assume $w = \Omega(\log n)$. Let us further assume, for simplicity, that all given weights W_u, $u \in V$ are rational numbers with the same denominator, with each numerator and the common denominator fitting in $\mathcal{O}(1)$ words. In this case, the sum $S = \sum_{u \in V} W_u$ has the same denominator as all W_u and numerator bounded by n times the numerator of the largest W_u. Since $w = \Omega(\log n)$, the numerator of S fits in $\mathcal{O}(1)$ more words than used for the largest W_u. Hence, numerator and denominator of S fit in $\mathcal{O}(1)$ words and can be computed in $\mathcal{O}(n)$ time. Moreover, the edge probabilities $p_{u,v} = \min\{W_u W_v / S, 1\}$ are also rationals with numerator and denominator fitting in $\mathcal{O}(1)$ words that can be computed in constant time if S is available.

Carefully examining the efficient sampling algorithm for Chung-Lu random graphs, Algorithm 2 of Miller and Hagberg [20], we see that now every step can be performed in the same deterministic time bound as on a Real RAM, except for the generation of bounded geometric variables and Bernoulli variables. Note that for any $p \in (0, 1)$ we have $\mathrm{Ber}(p) \sim \mathrm{Geo}(1 - p, 1)$, so Theorem 5 shows that the bounded geometric as well as the Bernoulli random variables can be sampled in expected constant time and bounded space (for $w = \Omega(\log n)$). Thus, we can bound the expected runtime of the modified generation for Chung-Lu graphs analogously to the Erdős-Rényi case, proving Theorem 7.

5 Conclusions and Future Work

We have presented new exact algorithms which can sample $\mathrm{Geo}(p)$ and $\min\{n, \mathrm{Geo}(p)\}$ in optimal time and space on a Word RAM. It remains open to find similar algorithms for other non-uniform random variates besides exponential and normal distributions. Moreover, it would be interesting to see whether our theoretically optimal algorithms are also practical, e.g., for generating very large geometric random variates for cryptographic applications [13].

Regarding our new exact algorithm for sampling Erdős-Rényi and Chung-Lu random graphs in optimal time and space on a Word RAM, we believe that similar results can be proven for the more general case where the weights W_u are given as bitstreams.

References

[1] Batagelj, V., Brandes, U.: Efficient generation of large random networks. Phys. Rev. E 71, 036113 (2005)

[2] Blanca, A., Mihail, M.: Efficient generation ϵ-close to G(n,p) and generalizations (2012), `arxiv.org/abs/1204.5834`

[3] Blitzstein, J.K., Diaconis, P.: A sequential importance sampling algorithm for generating random graphs with prescribed degrees. Internet Mathematics 6, 489–522 (2011)

[4] Brent, R., Zimmermann, P.: Modern Computer Arithmetic. Cambridge University Press (2010)

[5] Chung, F., Lu, L.: Connected components in random graphs with given expected degree sequences. Annals of Combinatorics 6, 125–145 (2002)
[6] Cook, S.A., Reckhow, R.A.: Time bounded random access machines. J. Comput. Syst. Sci. 7, 354–375 (1973)
[7] De, A., Kurur, P.P., Saha, C., Saptharishi, R.: Fast integer multiplication using modular arithmetic. In: 40th Symp. Theory of Computing (STOC), pp. 499–506 (2008)
[8] Devroye, L.: Non-uniform random variate generation. Springer (1986)
[9] Erdős, P., Rényi, A.: On random graphs. Publ. Math. Debrecen 6, 290–297 (1959)
[10] Flajolet, P., Saheb, N.: The complexity of generating an exponentially distributed variate. J. Algorithms 7, 463–488 (1986)
[11] Flajolet, P., Pelletier, M., Soria, M.: On Buffon machines and numbers. In: 22nd Symp. Discrete Algorithms (SODA), pp. 172–183 (2011)
[12] Fürer, M.: Faster integer multiplication. SIAM J. Comput. 39, 979–1005 (2009)
[13] Ghosh, A., Roughgarden, T., Sundararajan, M.: Universally utility-maximizing privacy mechanisms. In: 41st Symp. Theory of Computing (STOC), pp. 351–360 (2009)
[14] Hagberg, A.A., Schult, D.A., Swart, P.J.: Exploring network structure, dynamics, and function using NetworkX. In: 7th Conf. Python in Science, pp. 11–15 (2008)
[15] Karney, C.: Random number library, version 1.5 (2012), http://sourceforge.net/projects/randomlib
[16] Keeling, M.J., Eames, K.T.: Networks and epidemic models. Journal of The Royal Society Interface 2, 295–307 (2005)
[17] Kim, J.H., Vu, V.H.: Generating random regular graphs. In: 35th Symp. Theory of Computing (STOC), pp. 213–222. ACM (2003)
[18] Knuth, D.E., Yao, A.C.-C.: The complexity of nonuniform random number generation. In: Traub, J.F. (ed.) Symposium on Algorithms and Complexity: New Directions and Recent Results, pp. 357–428 (1976)
[19] Mehlhorn, K., Näher, S.: LEDA: A Platform for Combinatorial and Geometric Computing. Cambridge University Press (1999)
[20] 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)
[21] Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., Alon, U.: Network motifs: Simple building blocks of complex networks. Science 298, 824–827 (2002)
[22] Norros, I., Reittu, H.: On a conditionally Poissonian graph process. Advances in Applied Probability 38, 59–75 (2006)
[23] Ray, J., Pinar, A., Seshadhri, C.: Are we there yet? When to stop a markov chain while generating random graphs. In: Bonato, A., Janssen, J. (eds.) WAW 2012. LNCS, vol. 7323, pp. 153–164. Springer, Heidelberg (2012)
[24] Solovay, R., Strassen, V.: A fast Monte-Carlo test for primality. SIAM J. Comput. 6, 84–85 (1977)
[25] van der Hofstad, R.: Random graphs and complex networks (2009), http://www.win.tue.nl/~rhofstad/NotesRGCN.pdf
[26] von Neumann, J.: Various techniques used in connection with random digits. In: Householder, A.S., et al. (eds.) The Monte Carlo Method. National Bureau of Standards, Applied Mathematics Series, vol. 12, pp. 36–38 (1951)