Combinatorics, Probability and Computing (2016) 25, 324–351. © Cambridge University Press 2016
doi:10.1017/S0963548315000358

Probabilistic Divide-and-Conquer: A New Exact Simulation Method, With Integer Partitions as an Example

RICHARD ARRATIA[1] and STEPHEN DeSALVO[2†]

[1]Department of Mathematics, University of Southern California, Los Angeles, CA 90089, USA
(e-mail: rarratia@math.usc.edu)

[2]Department of Mathematics, University of California Los Angeles, Los Angeles, CA 90024, USA
(e-mail: stephendesalvo@math.ucla.edu)

Received 17 October 2011; revised 11 November 2015; first published online 22 January 2016

We propose a new method, probabilistic divide-and-conquer, for improving the success probability in rejection sampling. For the example of integer partitions, there is an ideal recursive scheme which improves the rejection cost from asymptotically order $n^{3/4}$ to a constant. We show other examples for which a non-recursive, one-time application of probabilistic divide-and-conquer removes a substantial fraction of the rejection sampling cost.

We also present a variation of probabilistic divide-and-conquer for generating i.i.d. samples that exploits features of the coupon collector's problem, in order to obtain a cost that is sublinear in the number of samples.

2010 *Mathematics subject classification*: Primary 60C05
Secondary 05A17

1. Introduction

1.1. Exact simulation

The task of simulation is to provide one or more samples from a set according to some given probability distribution. For many combinatorial problems, the given distribution is the uniform choice over all possibilities of a fixed size. A host of combinatorial objects, including assemblies, multisets, and selections, may be expressed (see [7]) in terms of a process of independent random variables, conditional on a weighted sum – the object size – equalling a given target.

The 'Boltzmann sampler' (see [18]) samples this independent process once and is content with an object size that is *close* to the given target. The *rejection sampling* algorithm, on the other hand, samples this independent process *repeatedly* until the condition is satisfied.

† Partially supported by a Dana and David Dornsife final year dissertation fellowship.

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

The main measure of the efficiency of such a rejection sampling algorithm is the expected number of times the independent process must be sampled before the condition is satisfied.

The essence of *probabilistic divide-and-conquer* – PDC – is random sampling of conditional distributions, which, *when appropriately pieced together*, represent a sample from the target distribution. We assume throughout that the target object S may be expressed as a pair (A, B), where[1]

$$A \in \mathcal{A},\ B \in \mathcal{B} \text{ have given distributions,} \tag{1.1}$$

$$A, B \text{ are independent,} \tag{1.2}$$

$$h : \mathcal{A} \times \mathcal{B} \to \{0, 1\} \tag{1.3}$$

$$\text{satisfies } p := \mathbb{E}\, h(A, B) \in (0, 1],$$

where, of course, we also assume that h is measurable, and

$$S \in \mathcal{A} \times \mathcal{B} \text{ has distribution } \mathcal{L}(S) = \mathcal{L}(\,(A, B) \,|\, h(A, B) = 1), \tag{1.4}$$

that is, the law of S is the law of the independent pair (A, B) *conditional on* having $h(A, B) = 1$.

Then rejection sampling may be viewed as sampling $(A_1, B_1), (A_2, B_2), \ldots$ from the law of (A, B) until $h(A_i, B_i) = 1$. PDC can be described as sampling from the law of $(A \,|\, h(A, B) = 1)$ *first*, say with observation x, and *then* sampling from the law of $(B \,|\, h(x, B) = 1)$. A simple comparison is shown in Algorithms 1–3.

Algorithm 1 Rejection sampling

1 Generate sample from $\mathcal{L}(A)$, call it a.
2 Generate sample from $\mathcal{L}(B)$, call it b.
3 Check if $h(a, b) = 1$; if so, return (a, b), otherwise restart.

Algorithm 2 Probabilistic divide-and-conquer

1 Generate sample from $\mathcal{L}(A \,|\, h(A, B) = 1)$, call it x.
2 Generate sample from $\mathcal{L}(B \,|\, h(x, B) = 1)$, call it y.
3 Return (x, y).

Our main tool is von Neumann's acceptance/rejection method, which is not necessary for PDC but which provides a simple and effective means for sampling from the conditional distributions in Algorithm 2. We review this method in Section 2.2. Our primary application of Algorithm 2 with von Neumann's rejection method is Algorithm 3, where the rejection cost is split between step 2 for sampling A, and step 3 for sampling

[1] The requirement that $p > 0$ is a choice we make here, for the sake of simpler exposition; there are $p = 0$ examples where divide-and-conquer is useful, see [14]. In cases where $p = 0$, the conditional distribution, apparently specified by (1.4), needs further specification – this is known as Borel's paradox.

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

B. Furthermore, step 3 in Algorithm 3 can be performed either directly or recursively; we give examples of each in Section 3.

Algorithm 3 Probabilistic divide-and-conquer via von Neumann

1 Generate sample from $\mathcal{L}(A)$, call it a.
2 Accept a with probability $t(a)$, where $t(a)$ is a function of $\mathcal{L}(B)$ and h; otherwise, restart.
3 Generate sample from $\mathcal{L}(B \mid h(a, B) = 1)$, call it y.
4 Return (a, y).

A key class of examples is discussed in Section 3.3, where the calculation for $t(a)$ is very simple, and the speedup is order $\sqrt{n}$ for the specific examples relating to k-cores and set partitions, and of order $n^{1/3}$ for the specific example relating to plane partitions. In these examples, the PDC is not recursive and the programming is very easy. At the opposite extreme, when B can be made into a scaled replica of the original problem, and calculation of $t(a)$ is still easy enough – as in the example of ordinary integer partitions, thanks to results of Hardy and Ramanujan [22], Rademacher [39] and Lehmer [27, 28] – a recursive, self-similar PDC is available, and although the algorithm is more involved, this gives the fastest method for large n.

One progenitor of PDC is by McKay and Wormald [31], where the ability to calculate likelihood ratios is combined with acceptance/rejection sampling to reduce the number of loops and double edges implied by a random configuration; the luck required to get a simple graph is spread over several stages. Another progenitor of PDC, this time in the recursive, self-similar case, is by Alonso [3] on sampling of Motzkin words. In that case, the rejection probabilities of partially completed Motzkin words are given explicitly by quotients of binomial coefficients.

A separate application of our divide-and-conquer paradigm, championed in Section 4, is what we call *mix-and-match*. Our desire is to combine $A_1, A_2, \ldots, A_m$, i.i.d. samples from $\mathcal{L}(A)$, and $B_1, B_2, \ldots, B_m$, i.i.d. samples from $\mathcal{L}(B)$, with m^2 chances of having $h(A_i, B_j) = 1$. However, doing this blindly does not achieve the goal of exact sampling, although it does yield a *consistent* estimator; see [13, Section 4.6]. A particularly practical variation, dubbed *deliberately missing data* [13, Section 4.4.3], fixes some number v of samples in advance, and samples $A_1, A_2, \ldots$ and $B_1, B_2, \ldots$ until exactly v matching pairs have been found; this procedure is notably sublinear in v. Such a comparison of two lists has been exploited previously, as in [17], the meet-in-the-middle attack, and in biological sequence matching [8, 6], where for two independent sequences of i.i.d. letters, we are interested in finding contiguous blocks of letters that appear in both sequences.

Finally, in Section 5 we present practical considerations when implementing PDC for integer partitions on a computer.

1.2. An example: integer partitions

Our first example to illustrate the use of PDC is uniform sampling of integer partitions of a given size n. Additional examples are given in Section 3.3.1. The starting point is a

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

Table 1. Summary of PDC algorithms. The quantities are proved in Section 3

Algorithm	Asymptotic number of rejections
Algorithm 4: Rejection sampling	$2\sqrt[4]{6}\,n^{3/4}$
Algorithm 5: PDC deterministic	$2\pi\sqrt[4]{6}\,n^{1/4}$
Algorithm 6: Self-similar PDC	$2\sqrt{2}$

conditional distribution of the form

$$\left((Z_1, Z_2, \ldots, Z_n) \,\middle|\, \sum_{i=1}^{n} i\, Z_i = n\right),$$

where $Z_1, \ldots, Z_n$ are *independent* geometric distributions with respective parameters $p_1, \ldots, p_n$, with $p_i = 1 - e^{-i\pi/\sqrt{6n}}$, $i = 1, \ldots, n$. The precise description of how they correspond to a random integer partition is described in Section 3.2; for now, we simply wish to emphasize the simple and explicit nature of the following algorithms, in a way accessible to the reader solely interested in implementing such algorithms on a computer. Many combinatorial structures follow a similar pattern; see [7].

In Table 1 we summarize the relevant rejection sampling and PDC algorithms (Algorithms 4–6); we postpone their full explanations until Section 3. One simple measure of an algorithm's performance is the asymptotically expected number of times the n-tuple $(Z_1, \ldots, Z_n)$ must be sampled before the simulation terminates and returns a valid sample, which we call the asymptotic number of rejections in the table.

In what follows, U will denote a uniform random variable in the interval $(0, 1)$, independent of all other random variables, and for use in Algorithm 6 we define, via (3.2) and (3.6),

$$f_n(j) := \mathbb{P}_{x(n)}\left(\sum_{i \leqslant n/2} 2i\, Z_{2i} = 2j\right). \tag{1.5}$$

Algorithm 4 Rejection sampling of integer partitions

1 Generate sample from $\mathcal{L}(Z_1, Z_2, \ldots, Z_n)$, call it $(z_1, \ldots, z_n)$.
2 If $\sum_{i=1}^{n} i\, z_i = n$, return $(z_1, \ldots, z_n)$; else restart.

Algorithm 5 Probabilistic divide-and-conquer deterministic second half for integer partitions

1 Generate sample from $\mathcal{L}(Z_2, \ldots, Z_n)$, call it $(z_2, \ldots, z_n)$.
Set $k := n - \sum_{i=2}^{n} i\, z_i$.
2 If $k \geqslant 0$ and $U < e^{-k\pi/\sqrt{6n}}$, return $(k, z_2, \ldots, z_n)$; else restart.

Remark 1. Note that while Algorithms 4 and 5 are entirely explicit, Algorithm 6 relies on our ability to determine whether $U < f_n(k/2)/\max_\ell f_n(\ell)$. *This is not the same as being*

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

able to compute $f_n(j)$ for various values of the parameters. The task in Algorithm 6 is considerably easier, since we simply need to compare the leading bits of each quantity until there is disagreement. In fact, *on average we only need the leading 2 bits of each side of the inequality in order to determine the rejection in step 4 of Algorithm 6*; see [25] and our discussion in Section 5.2. There are also several other improvements to Algorithm 6 that can be applied at the expense of further complicating the algorithm; these are discussed in Section 3.5.3.

Algorithm 6 Self-similar probabilistic divide-and-conquer for integer partitions

procedure SS_PDC_IP(n)
1 If $n = 1$, return 1; otherwise,
2 Generate sample from $\mathcal{L}(Z_1, Z_3, Z_5, \ldots)$, call it $(z_1, z_3, z_5, \ldots)$.
3 Set $k := n - \sum_{i \text{ odd}} i\, z_i$.
4 If $k < 0$ or k is odd, restart.
5 If $U < f_n(k/2)/\max_\ell f_n(\ell)$,
let $(z_2, z_4, \ldots) =$ SS_PDC_IP($k/2$);
return $(z_1, z_2, \ldots, z_n, 0, 0, \ldots)$.
Else restart.
end procedure

On a personal computer, the RandomPartition function in Mathematica®'s Combinatorica package [29] appears to hit the wall at around $n = 2^{20}$. On the same computer, the recursive divide-and-conquer algorithm in Algorithm 6 can handle n as large[2] as 2^{58}. Relative to the rejection sampling algorithm in Algorithm 4, analysed in Section 3.2, the recursive divide-and-conquer achieves more than a trillion-fold speedup at $n = 2^{58}$.

2. The basic lemma for exact sampling with divide-and-conquer

2.1. Statement of the PDC Lemma

The following lemma is a straightforward application of Bayes' formula.

Lemma 2.1. *Suppose X is a random element of $\mathcal{A}$ with distribution*

$$\mathcal{L}(X) = \mathcal{L}(A \mid h(A, B) = 1), \tag{2.1}$$

and Y is a random element of $\mathcal{B}$ with conditional distribution

$$\mathcal{L}(Y \mid X = a) = \mathcal{L}(B \mid h(a, B) = 1). \tag{2.2}$$

Then $(X, Y) =^d S$, that is, the pair (X, Y) has the same distribution as S, given by (1.4).

[2] In MATLAB [30], we got up to 2^{49}; the 2^{58} is from a C++ implementation. In both cases, memory rather than time is the limiting factor.

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

2.2. Use of acceptance/rejection sampling

Assume that we know how to generate a sample from $\mathcal{L}(A)$ – this is under the distribution in (1.1), where A, B are independent. But instead we need to sample from an alternate distribution, $\mathcal{L}(A \mid h(A,B) = 1)$, denoted above as that of $X \in \mathcal{A}$. The rejection method recipe, for using equation (2.1), may be viewed as having two steps, as follows. Let $p_a := \mathbb{E}\, h(a,B)$.

(a) Find a threshold function $t : \mathcal{A} \to [0,1]$, with $t(a)$ proportional to p_a, and $t(a) \leqslant 1$, for all $a \in \mathcal{A}$.

(b) Propose i.i.d. samples $A_1, A_2, \ldots$ and independently generate uniform (0,1) random variables $U_1, U_2, \ldots$, until $U_i \leqslant t(A_i)$.

Assuming that we can find an a^* where p_a achieves its maximum value, then the optimal function is $t(a) = p_a/p_{a^*}$. However, we note that even when p_{a^*} is difficult to compute exactly, we simply need an upper bound on p_{a^*}, and really only the ability to compute the leading bits of $t(a)$ as needed until a disagreement is reached between the bits of a uniform random number in $(0,1)$ and $t(a)$; see Remark 1 and Section 3.6.

For comparison with hard rejection sampling, we write

$$t(a) = \frac{C}{p}\, p_a,$$

where C is allowed to be *any* positive constant such that $C \leqslant p/p_{a^*}$, with optimal value given by $C = p/p_{a^*}$. See [16] for a thorough treatment of acceptance/rejection methods.

The expected fraction of proposed samples A_i to be accepted will be the average of $t(a)$ with respect to the distribution of A, that is,

$$p_{\text{acc}} := \mathbb{P}(U \leqslant t(A)) = \mathbb{E}\, t(A) = C \times \frac{p_A}{p} = C,$$

and the expected number of proposals needed to get each acceptance is the reciprocal of this, so we define

$$\text{acceptance cost} := \frac{1}{p_{\text{acc}}} = \frac{1}{C} = \frac{p_{a^*}}{p}, \tag{2.3}$$

where the last equality assumes assumes the optimal choice of C. For comparison, if we were using rejection sampling instead of divide-and-conquer, that is, proposing pairs (A_i, B_i) until $h(A_i, B_i) = 1$, with success probability p at each trial and expected number of proposals to get one success equal to $1/p$, then, *ignoring the cost of proposing the* B_i, the ratio of old cost to new might be called a speedup:

$$\text{speedup} = \frac{\text{old cost}}{\text{new cost}} = \frac{1/p}{1/C} = \frac{1}{p_{a^*}}, \tag{2.4}$$

where again the last equality assumes the optimal choice of C. Even though the cost involved in proposing the B is in general *not negligible*, in Section 3.3 we will give several natural examples, similar to Algorithm 5, where it is negligible.

Remark 2. For each proposed value $a = A_i$, we need to be able to compute $t(a)$; this can be either a minor cost, a major cost, or an absolute impediment, making probabilistic

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

divide-and-conquer infeasible. All of these variations occur in the context of integer partitions, and will be discussed in Sections 3.4–3.5, and again in Section 3.6; see also Remark 1.

Remark 3 (hard versus soft rejection). Adopting the language of information theory, where soft-decision decoders are contrasted with hard-decision decoders, by a *hard rejection* function we mean an acceptance/rejection function $t : \mathcal{A} \to [0,1]$ whose image is actually $\{0,1\}$. We may thus consider Algorithm 4 to be more specifically a *hard* rejection sampling algorithm. This is in contrast to Algorithm 5, which may be considered a *soft* rejection sampling algorithm.

3. Algorithms for simulating random integer partitions of n

The computational analysis that follows uses an informal adaptation of *uniform* costing; see Section 3.6. Some elements of the analysis, specifically asymptotics for the acceptance rate, will be given rigorously, for example in theorems 3.1, 3.2 and 3.3.

An integer partition, of size n, is a multiset of positive integers, with sum n. We denote the number of partitions of a given size n by $p(n)$. We typically denote the parts of an integer partition by $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_\ell > 0$, where ℓ is the number of parts in the partition, which varies from 1 to n. Instead of listing the part sizes, we can instead record the multiplicities $c_1, \ldots, c_n$ of each number $1, \ldots, n$ in the integer partition.

For example, the integer partition of size 12 with parts $(5,3,2,1,1)$ can be equally described by the 12-tuple $(c_1, \ldots, c_{12}) = (2,1,1,0,1,0,\ldots,0)$, where c_i denotes the number of parts of size i. All partitions of a given size n satisfy $\sum_{i=1}^{n} i\, c_i = n$, and this motivates the probabilistic formulation, given in Section 3.2, via formulas (3.3) and (3.4).

3.1. For baseline comparison: table methods

A natural algorithm for the generation of a random integer partition is to find the largest part first, then the second largest part, and so on. The main cost associated with this method is the storage of all the distributions. Some details follow.

Let $p(\leqslant k, n)$ denote the number of partitions of n with each part of size $\leqslant k$, so that $p(\leqslant n, n) = p(n)$. These can be quickly calculated from the recurrence

$$p(\leqslant k, n) = p(\leqslant k-1, n) + p(\leqslant k, n-k), \tag{3.1}$$

where the right-hand side counts the number of partitions without any k plus the number of partitions with at least one k. If only a few partitions are desired, the quantities on the right-hand side above can be computed as needed via the recursion given in equation (3.1) and normalized to form a probability distribution. However, rather than computing each quantity as it appears, an $n \times n$ table, whose (i,j) entry is $p(\leqslant i, j)$, can be computed and stored. The generation of random partitions is extremely fast once this table has been created.

An algorithm for simulating a random partition, based on Euler's identity

$$np(n) = \sum_{d,j \geqslant 1} dp(n - d\,j),$$

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

is given in [34, 35], and cited as 'the recursive method'. We have not found a clear-cut complexity analysis in the literature, although [12] comes close. But we believe that this algorithm is less useful than the $p(\leqslant k, n)$ table method. Sampling from the distribution on (d, j) implicit in

$$np(n) = \sum_{d,j \geqslant 1} d\, p(n - d\, j)$$

requires computation of partial sums; if all the partial sums for

$$\sum_{d,j \geqslant 1, dj \leqslant m} d\, p(m - d\, j)$$

with $m \leqslant n$ are stored in a table, the total storage requirement is of order $n^2 \log n$, and if they are not stored, computing the values, as needed, becomes a bottleneck.

3.2. Rejection sampling

Fristedt [21] observed that, for any choice $x \in (0, 1)$, if $Z_i \equiv Z_i(x)$ has the geometric distribution given by

$$\mathbb{P}(Z_i = k) \equiv \mathbb{P}_x(Z_i = k) = (1 - x^i)\ (x^i)^k,\ k = 0, 1, 2, \ldots, \tag{3.2}$$

with $Z_1, Z_2, \ldots$ independent, and T is defined by

$$T = Z_1 + 2Z_2 + \cdots, \tag{3.3}$$

then, conditional on the event $\{T = n\}$, the partition λ having Z_i parts of size i, for $i = 1, 2, \ldots$, is uniformly distributed over the $p(n)$ possible partitions of n.[3]

This extremely useful observation is easily seen to be true, since for any non-negative integers $(c_1, c_2, \ldots)$ with $c_2 + 2c_2 + \cdots = n$, specifying a partition λ of the integer n,

$$\begin{aligned} \mathbb{P}(Z_i = c_i, i = 1, 2, \ldots) &= \prod \mathbb{P}(Z_i = c_i) = \prod \left((1 - x^i)(x^i)^{c_i}\right) \\ &= x^{c_1 + 2c_2 + \cdots} \prod (1 - x^i) = x^n \prod (1 - x^i), \end{aligned} \tag{3.4}$$

which does not vary with the partition λ of n.

The event $\{T = n\}$ is the disjoint union, over all partitions λ of n, of the events whose probabilities are given in (3.4), showing that

$$\mathbb{P}_x(T = n) = p(n)\ x^n \prod (1 - x^i). \tag{3.5}$$

If we are interested in random partitions of n, an especially effective choice for x, used in [21, 37], is

$$x(n) = \exp(-c/\sqrt{n}), \quad \text{where } c = \pi/\sqrt{6}. \tag{3.6}$$

Under this choice, we have, as $n \to \infty$,

$$\frac{1}{n}\, \mathbb{E}_{x(n)} T \to 1, \quad \frac{1}{n^{3/2}} \mathrm{Var}_{x(n)} T \to \frac{2}{c}; \tag{3.7}$$

[3] We write $\mathbb{P}_x$ or $\mathbb{P}$, and Z_i or $Z_i(x)$ interchangeably, depending on whether the choice of $x \in (0, 1)$ needs to be emphasized, or left understood.

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

this is essentially a pair of Riemann sums [7, p. 106]. If we write $\sigma(x)$ for the standard deviation of T, then the second part of (3.7) says: with $x = x(n)$, as $n \to \infty$,

$$\sigma(x) \sim \sqrt{2/c}\; n^{3/4}. \tag{3.8}$$

The *local central limit heuristic* would thus suggest asymptotics for $\mathbb{P}_x(T = n)$, and these simplify as follows, using (3.6) and (3.7):

$$\mathbb{P}_x(T = n) \sim \frac{1}{\sqrt{2\pi}\;\sigma(x)} \sim \frac{1}{2\sqrt[4]{6}\, n^{3/4}}. \tag{3.9}$$

Hardy and Ramanujan [22] proved the asymptotic formula, as $n \to \infty$,

$$p(n) \sim \exp(2c\sqrt{n})/(4\sqrt{3}n), \quad \text{where } c = \pi/\sqrt{6} \doteq 1.282550. \tag{3.10}$$

The Hardy–Ramanujan asymptotics (3.10) and the exact formula (3.5) combine to show that (3.9) does hold.

Theorem 3.1 (analysis of rejection sampling). *Algorithm 4* does *produce one sample from the desired distribution, and the expected number of proposals until the algorithm terminates is asymptotically* $2\sqrt[4]{6}\, n^{3/4}$.

Proof. It is easily seen that $\mathbb{P}_x(Z_i = 0 \text{ for all } i > n) \to 1$. Hence, the asymptotics (3.9), given for the infinite sum T, also serve for the finite sum T_n, in which the number of summands, along with the parameter $x = x(n)$, varies with n. □

Remark 4. We are *not* claiming that the running time of the algorithm grows like $n^{3/4}$, but only that the number of proposals needed to get one acceptable sample grows like $n^{3/4}$. The time to propose a sample also grows with n. Assigning cost 1 to each call to the random number generator, with all other operations being free, the cost of proposing one sample grows like $\sqrt{n}$ rather than n; details are in Section 3.6. Combining with Theorem 3.1, the cost of the rejection sampling algorithm grows like $n^{5/4}$.

3.3. Divide-and-conquer with a deterministic second half

Perhaps surprisingly, the choice $A = (Z_2, \ldots, Z_n)$ and $B = Z_1$ from Algorithm 5 is excellent. Loosely speaking, it reduces the cost of rejection sampling from order $n^{3/4}$ to order $n^{1/4}$.

The analysis of the speedup relative to waiting-to-get-lucky, as defined in (2.4), is easy.

Theorem 3.2. *The speedup, as defined in* (2.4)*, of Algorithm 5, relative to the rejection sampling algorithm of Algorithm 4, is asymptotically* $\sqrt{n}/c$*, with* $c = \pi/\sqrt{6}$*. Equivalently, the acceptance cost is asymptotically* $2\,\pi\,\sqrt[4]{6}\, n^{1/4}$.

Proof. We have $A = (Z_2, \ldots, Z_n)$, $B = Z_1$, so with $a = (z_2, \ldots, z_n)$ and

$$\ell = n - (2z_2 + \cdots + nz_n),$$

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

we have $\mathbb{E}\, h(a, B) = \mathbb{P}(Z_1 = \ell)$. Following (2.4), and recalling that $x = e^{-c/\sqrt{n}}$, where $c = \pi/\sqrt{6}$, we have

$$\text{speedup} = \frac{1}{\max_\ell \mathbb{P}(Z_1 = \ell)} = \frac{1}{\mathbb{P}(Z_1 = 0)} = \frac{1}{1-x} \sim \frac{\sqrt{n}}{c}.$$

Combined with the asymptotic rejection cost for Algorithm 4, as given by Theorem 3.1, this yields the claimed asymptotic acceptance cost for Algorithm 5. □

To review, step 1 of Algorithm 5 is to simulate $(Z_2, Z_3, \ldots, Z_n)$, and accept it with probability *proportional to* the chance that $Z_1 = n - (2Z_2 + \cdots + nZ_n)$. The speedup shows the brilliance of the idea in [42]: conditional on accepting a value $(z_2, \ldots z_n)$, we are done; the distribution

$$\mathcal{L}\left(Z_1 \middle| Z_1 + \sum_{i=2}^{n} i\, z_i = n\right)$$

is trivial, so there is no need to 'wait for a lucky Z_1'. Algorithm 5 is a soft rejection sampling algorithm; it might be more appropriate to call it a *soft acceptance* sampling algorithm. In contrast, the hard rejection in Algorithm 4 can be viewed as simulating $(Z_2, Z_3, \ldots, Z_n)$ and then *simulating* Z_1 to see whether or not $Z_1 = n - (2Z_2 + \cdots + nZ_n)$; if Z_1 does not have the correct value, then we resample $(Z_2, Z_3, \ldots, Z_n)$ (and then Z_1) until we finally get lucky.

3.3.1. Examples: *k*-cores, set partitions, and plane partitions. The PDC using a deterministic second half is remarkably robust and powerful. It is robust in that it applies to a wide variety of simulation tasks and gives a non-trivial speedup.

For a first additional example, there is much recent interest in k-cores [26]. There is an easy-to-calculate bijection between k-cores and integer partitions with $\lambda_1 < k$. The natural way to simulate partitions of n with $\lambda_1 < k$, following Fristedt's method from Section 3.2, is to use independent $Z_1, \ldots, Z_{k-1}$, but, instead of defining $x = x(n)$ as in (3.6), we instead solve numerically for $x = x(k, n)$ which satisfies

$$n = \sum_{1 \leqslant i < k} \mathbb{E}\, iZ_i = \sum_{1 \leqslant i < k} i\, x^i/(1 - x^i).$$

Using $A = (Z_2, \ldots, Z_{k-1})$ and $B = Z_1$, the optimal threshold function is explicitly given by $\mathbb{P}(Z_1 = m)/\max_j \mathbb{P}(Z_1 = j)$. The speedup, relative to rejection sampling, is $1/(1 - x(k, n))$ – exactly the same form as in Theorem 3.2, except that we no longer have the asymptotic analysis that $1/(1-x) \sim \sqrt{n}/c$.

For a second example, which shows that a good deterministic second half is not always the first component, consider the integer partition underlying a random set partition, where all set partitions are equally likely. See [7, Section 10.1] for more details. The summary is that with $x = x(n)$ to solve $xe^x = n$, so that $x \sim \log n$, we want independent Z_i where Z_i is Poisson-distributed with parameter $\lambda_i = x^i/i!$, and getting lucky is to have $\sum_{i=1}^{n} iZ_i = n$. For the deterministic second half we have $B = Z_j$, where we pick j to maximize the speedup. Thus we want to minimize $\max_k \mathbb{P}(Z_j = k)$, that is, to maximize λ_j, so we take $j = \lfloor x \rfloor$. Since $j = x + o(\sqrt{x})$, we have $\lambda_j = x^j/j! \sim e^x/\sqrt{2\pi x}$, and since $e^x = n/x$, we

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

have $\lambda_j \sim n/\sqrt{2\pi x^3}$. As in the previous paragraph, again the optimal threshold function is given by an explicit expression which is easy to evaluate. The speedup factor is $1/\max_k \mathbb{P}(Z_j = k)$, which is asymptotic to $\sqrt{2\pi\lambda_j} \sim (2\pi)^{1/4}\sqrt{n}/x^{3/4}$. Pittel showed [38] that the expected number of proposals for rejection sampling is asymptotic to $\sqrt{2\pi n(x+1)}$. Hence the expected number of proposals using the deterministic second half PDC is asymptotically $x^{5/4}/(2\pi)^{1/4}$, which is $O(\log^{5/4}(n))$.

For a third example, we consider a sampling procedure in [9], in which a random plane partition is generated in two stages. Stage 1 is rejection sampling, where the proposal is an array of independent geometrically distributed random variables $\{Z_{i,j}\}$, $0 \leqslant i, j < n$.[4] To get an instance of weight n, the expected number of proposals, using rejection sampling, is of order $n^{2/3}$. There is an algorithm, to make the proposal, whose work is of the same order as the entropy lower bound, which is order $n^{2/3}$, so the cost for stage 1 is order $n^{4/3}$. Stage 2 is to apply a bijection due to Pak [36], which, as implemented in [9], takes order $n\log^3(n)$. Combined, [9] has order $n^{4/3}$ to name a random ensemble of weight n, plus order $n\log^3 n$ to implement the bijection, so the order $n^{4/3}$ first stage dominates the computation. Using our deterministic second half PDC, with $B = Z_{0,0}$, we obtain a speedup of order $n^{1/3}$, bringing the cost of stage 1 to $O(n)$, and hence bringing the total cost of the algorithm down to the cost of implementing the bijection, *viz.*, order $n\log^3 n$.

3.4. Divide-and-conquer, by small versus large

We now return to the example of random sampling of integer partitions. Using $x = x(n)$ from (3.6), for any fixed choice $b \in \{1, 2, \ldots, n-1\}$, we let

$$A = (Z_{b+1}, Z_{b+2}, \ldots, Z_n), \quad B = (Z_1, \ldots, Z_b), \tag{3.11}$$

noting that one extreme case, $b = 1$, was handled in Section 3.3. With

$$T_A = \sum_{i=b+1}^{n} iZ_i, \qquad T_B = \sum_{i=1}^{b} iZ_i, \tag{3.12}$$

we want to sample from (A, B) conditional on $(T_A + T_B = n)$. The standard paradigm for deterministic divide-and-conquer, that the two tasks should be roughly equal, suggests b of order $\sqrt{n}$.

In order to simulate X, we will use rejection sampling, as reviewed in Section 2.2. To find the optimal rejection probabilities, we want the largest C such that

$$C \max_j \frac{\mathbb{P}(T_B = n - j)}{\mathbb{P}(T_n = n)} \leqslant 1,$$

or equivalently,

$$C = \frac{\mathbb{P}(T_n = n)}{\max_j \mathbb{P}(T_B = j)}. \tag{3.13}$$

[4] With $\mathbb{P}(Z_{i,j} \geqslant k) = (x^{i+j+1})^k$ and $x = x(n) = \exp(-(2\zeta(3)/n)^{1/3})$, akin to (3.2) and (3.6).

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

Once an A has been accepted, and we have our target $n - T_A$ for the sum T_B, we simply propose $B = (Z_1, Z_2, \ldots, Z_b)$ until $Z_1 + 2Z_2 + \cdots + bZ_b = n - T_A$; then the pair (A, B) is our random partition.

Remark 5. The values $\mathbb{P}(T_B = j)$ for $j = 0, 1, \ldots, n$ can be computed using the recursion (3.1); what we really have is a variant of the $n \times n$ table method of Section 3.1, in which the table is $b \times n$. The computation time for the entire table is $b\,n$. However, one only needs to store the current and previous row, (or with overwriting, only the current row), so the storage is n. Once we have the last row of the $b \times n$ table, we can easily find C and indeed the entire threshold function t.

3.5. Self-similar, recursive probabilistic divide-and-conquer: $p(z) = d(z)p(z^2)$

The methods of Sections 3.2–3.3 have acceptance costs that go to infinity with n. We now demonstrate a recursive probabilistic divide-and-conquer algorithm that has an asymptotically constant acceptance cost.

A well-known result in partition theory is

$$p(z) = \prod_i (1 - z^i)^{-1} = \left(\prod_i 1 + z^i\right)\left(\prod_i \frac{1}{1 - z^{2i}}\right) = d(z)p(z^2), \tag{3.14}$$

where $d(z) = \prod_i 1 + z^i$ is the generating function for the number of partitions with distinct parts, and $p(z^2)$ is the generating function for the number of partitions where each part has an even multiplicity. This can of course be iterated to, for example, $p(z) = d(z)d(z^2)p(z^4)$, *etc*., and this forms the basis for a recursive algorithm.

Recall from (3.2) in Section 3.2 that each $Z_i \equiv Z_i(x)$ is geometrically distributed with $\mathbb{P}(Z_i \geqslant k) = x^{ik}$. The *parity bit* of Z_i, defined by

$$\epsilon_i = 1(Z_i \text{ is odd}),$$

is a Bernoulli random variable $\epsilon_i \equiv \epsilon_i(x)$, with

$$\mathbb{P}(\epsilon_i(x) = 1) = \frac{x^i}{1 + x^i}, \quad \mathbb{P}(\epsilon_i(x) = 0) = \frac{1}{1 + x^i}. \tag{3.15}$$

Furthermore, $(Z_i(x) - \epsilon_i)/2$ is geometrically distributed, as $Z_i(x^2)$, again in the notation (3.2), and $(Z_i(x) - \epsilon_i)/2$ is independent of ϵ_i. What we really use is the converse: with $\epsilon_i(x)$ as above, independent of $Z_i(x^2)$, the $Z_i(x)$ constructed as

$$Z_i(x) := \epsilon_i(x) + 2Z_i(x^2), \quad i = 1, 2, \ldots$$

indeed has the desired geometric distribution.

Theorem 3.3. *The asymptotic acceptance cost for one step of the recursive divide-and-conquer algorithm using* $A = (\epsilon_1(x), \epsilon_2(x), \ldots)$, $B = ((Z_1(x) - \epsilon_1)/2, (Z_2(x) - \epsilon_2)/2, \ldots)$, *is* $\sqrt{8}$.

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

Proof. The acceptance cost $1/C$ can be computed via (3.13) and (3.9), with

$$C = \frac{\mathbb{P}_x(T=n)}{\max_k \mathbb{P}_{x^2}\left(T=\frac{n-k}{2}\right)} = \frac{\mathbb{P}_x(T=n)}{\max_k \mathbb{P}_{x^2}(T=k)} \sim \frac{\frac{1}{\sqrt{2\pi}\,\sigma(x)}}{\frac{1}{\sqrt{2\pi}\,\sigma(x^2)}}$$
$$\sim \frac{n^{-3/4}}{(n/4)^{-3/4}} = 4^{-3/4} = 8^{-1/2}. \qquad \square$$

In effect, the recursive algorithm is to determine $(Z_1, Z_2, \ldots, Z_n)$ conditional on $(Z_1 + 2Z_2 + \cdots = n)$, by finding the binary expansions: first the 1 bits of all the Z_i, then the 2 bits, then the 4 bits, and so on. Note that the rejection probabilities can be computed in time which is little-o of the cost of generating the random bits; see Remark 1 and Section 3.6.

3.5.1. Exploiting a parity constraint. Theorem 3.3 states that the asymptotic acceptance cost for proposals of $A = (\epsilon_1(x), \epsilon_2(x), \ldots)$ is $2\sqrt{2}$, and this already takes into account an obvious lower bound of 2, since the parity of $T_A = \epsilon_1 + \epsilon_2 + \cdots + \epsilon_n$ is nearly equally distributed over {odd, even}, and rejection is guaranteed if T_A does not have the same parity as n. An additional speedup is attainable by moving ϵ_1 from the A side to the B side: instead of simulating ϵ_1, there will now be a trivial task, just as there was for Z_1 in the '$b = 1$' procedure of Section 3.3. That is, we switch to $A = (\epsilon_2(x), \epsilon_3(x), \ldots)$ and $B = (\epsilon_1(x), (Z_1(x) - \epsilon_1)/2, (Z_2(x) - \epsilon_2)/2, \ldots)$; the parity of the new T_A dictates, deterministically, the value of the first component of B, under the conditioning on $h(a, B) = 1$. The rejection probabilities for a proposed A are like those in Theorem 3.3, but with an additional factor of $1/(1+x)$ or $x/(1+x)$, depending on the parity of $n + \epsilon_2 + \cdots + \epsilon_n$. Since $x = x(n) \to 1$ as $n \to \infty$, these two factors both tend to $1/2$, so the constant C as determined by (3.13) becomes, asymptotically, twice as large.

Theorem 3.4. *The asymptotic acceptance cost for one step of the recursive divide-and-conquer algorithm using* $A = (\epsilon_2(x), \epsilon_3(x), \ldots)$ *and* $B = (\epsilon_1(x), (Z_1(x) - \epsilon_1)/2, (Z_2(x) - \epsilon_2)/2, \ldots)$ *is* $\sqrt{2}$.

Proof. The acceptance cost $1/C$ can be computed, as in the proof of Theorem 3.3, with the only change being that in the display for computing C, the expression under the $\max_k$, which was $\mathbb{P}(T(x^2) = (n-k)/2)$, changes to

$$\mathbb{P}\left(2|n-k+\epsilon_1(x) \text{ and } T(x^2) = \left\lfloor \frac{n-k}{2} \right\rfloor\right)$$
$$= \mathbb{P}(2|n-k+\epsilon_1(x)) \times \mathbb{P}\left(T(x^2) = \left\lfloor \frac{n-k}{2} \right\rfloor\right) \sim \frac{1}{2}\mathbb{P}\left(T(x^2) = \left\lfloor \frac{n-k}{2} \right\rfloor\right). \qquad \square$$

3.5.2. The overall cost of the main problem and all its subproblems. Informally, for the algorithm in the previous section, the main problem has size n and acceptance cost $\sqrt{2}$, applied to a proposal cost asymptotic to $c_0\sqrt{n}$, for a net cost $\sqrt{2}\,c_0\sqrt{n}$. The first subproblem has random size, concentrated around $n/4$, and hence half the cost of the

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

main problem. The sum of a geometric series with ratio $1/2$ is twice the first term, so the net cost of the main problem and all subproblems combined is $2\sqrt{2}\,c_0\sqrt{n}$.

In framing a theorem to describe this, we try to allow for a variety of costing schemes. We believe that the first sentence in the hypotheses of Theorem 3.5 is valid, with $\theta = 1/2$, for the scheme of Remark 3.2. The second sentence, about costs of tasks other than proposals, is trivially true for the scheme of Remark 3.2, but may indeed be false in costing schemes which assign a cost to memory allocation, and communication.

Theorem 3.5. Assume *that the cost* $C(n)$ *of proposing the* $A = (\epsilon_2(x), \epsilon_3(x), \ldots)$ *in the first step of the algorithm of Section 3.5.1 is given by a deterministic function with* $C(n) \sim c_0 n^\theta$ *for some positive constant* c_0 *and constant* $\theta \geqslant 1/2$, *or even more generally,* $C(n) = n^\theta$ *times a slowly varying function of* n. *Assume that the cost of all steps of the algorithm, other than making proposals,*[5] *is relatively negligible, that is,* $o(C(n))$. *Then, the asymptotic cost of the entire algorithm is*

$$\frac{1}{1 - 1/4^\theta}\,\sqrt{2}C(n) \leqslant 2\sqrt{2}\,C(n).$$

Proof. The key place to be careful is in the distinction between the distribution of a proposed $A = (\epsilon_2(x), \epsilon_3(x), \ldots)$, and the distribution after rejection/acceptance. For proposals in which the ϵ_i are independent, with $T_A := \sum_2^n i\,\epsilon_i(x)$ and $x = x(n)$ from (3.6), calculation gives $\mathbb{E}\,T_A \sim n/2$ and $\mathrm{Var}(T_A) \sim (1/c)n^{3/2}$. Chebyshev's inequality for being at least k standard deviations away from the mean, to be used with $k = k(n) = o(n^{1/4})$ and $k \to \infty$, gives

$$\mathbb{P}(|T_A - \mathbb{E}\,T_A| > k\,\mathrm{SD}(T_A)) \leqslant 1/k^2.$$

Now consider the good event G that a proposed A is accepted; conditional on G, the ϵ_i are no longer independent. But the upper bound from Chebyshev is robust, with $\mathbb{P}(|T_A - \mathbb{E}\,T_A| > k\,\mathrm{SD}(T_A)|G) \leqslant 1/(k^2\,\mathbb{P}(G))$. Since $\mathbb{P}(G)$ is bounded away from zero, by Theorem 3.4, we still have an upper bound which tends to zero, and shows that $(n - T_A)/2$, divided by n, converges in probability to $1/4$.

Write $N_i \equiv N_i(n)$ for the random size of the subproblem at stage i, starting from $N_0(n) = n$. The previous paragraph showed that for $i = 0$, $N_{i+1}(n)/N_i(n) \to 1/4$, where the convergence is *convergence in probability*, and the result extends automatically to each fixed $i = 0, 1, 2, \ldots$. We have deterministically that $N_{i+1}/N_i \leqslant 1/2$, so in particular $N_i > 0$ implies $N_{i+1} < N_i$. Set $C(0) = 0$, redefining this value if needed, so that the cost of all proposals is exactly the random sum

$$S(n) := \sum_{i \geqslant 0} C(N_i(n)).$$

It is then routine analysis to use the hypothesis that $C(n)$ is regularly varying to conclude that $S(n)/C(n) \to 1/(1 - 4^{-\theta})$, where again the convergence is convergence in probability.

[5] Here we mean the arithmetic to compute acceptance/rejection thresholds, the generation of the random numbers used in those acceptance/rejection steps, and merging the accepted proposals into a single partition.

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

The deterministic bound $N_{i+1}(n)/N_i(n) \leqslant 1/2$ implies that the random variables $S(n)/C(n)$ are *bounded*, so it also follows that $\mathbb{E}\, S(n)/C(n) \to 1/(1-4^{-\theta})$. □

3.5.3. A variation based on $p(z) = p_{\text{odd}}(z)\, p(z^2)$. With

$$p_{\text{odd}}(z) := \prod_{i \text{ odd}} (1 - z^i)^{-1},$$

Euler's identity $d(z) = p_{\text{odd}}(z)$ suggests a variation on the algorithm of Section 3.5. It is arguable whether the original algorithm, based on $p(z) = d(z)\, p(z^2)$, and the variant, based on $p(z) = p_{\text{odd}}(z)\, p(z^2)$, are genuinely different.

Arguing that the variant algorithm is different. The initial proposal is

$$A = (Z_1(x), Z_3(x), Z_5(x), \ldots).$$

Upon acceptance, we have determined $(C_1(\lambda), C_3(\lambda), C_5(\lambda), \ldots)$, where λ is the partition of n that the full recursive algorithm will determine, and $C_i(\lambda)$ is the number of parts of size i in λ. The B task will find $(C_2(\lambda), C_4(\lambda), C_6(\lambda), \ldots)$ by iterating the divide-and-conquer idea, so that the second time through the A procedure determines $(C_2(\lambda), C_6(\lambda), C_{10}(\lambda), \ldots)$, and the third time through the A procedure determines $(C_4(\lambda), C_{12}(\lambda), C_{20}(\lambda), \ldots)$, and so on.

Arguing that the variant algorithm is essentially *the same.* Just as in Euler's bijective proof that $p_{\text{odd}}(z) = d(z)$, the original algorithm had a proposal $A = (\epsilon_1(x), \epsilon_2(x), \ldots)$, which can be used to construct the proposal $(Z_1(x), Z_3(x), Z_5(x), \ldots)$ for the variant algorithm. That is, one can check that starting with independent $\epsilon(i, x) \equiv \epsilon_i(x)$ given by (3.15), for $j = 1, 3, 5, \ldots,$

$$Z_j := \sum_{m \geqslant 0} \epsilon(j\, 2^m, x)\; 2^m$$

indeed has the distribution of $Z_j(x)$ specified by (3.2), with $Z_1, Z_3, \ldots$ independent. Conversely, one can check that starting with the independent geometrically distributed $Z_1(x), Z_3(x), \ldots,$ taking base 2 expansions yields independent $\epsilon_1(x), \epsilon_2(x), \ldots$ with the Bernoulli distributions specified by (3.15). Hence one *could* program the two algorithms so that they are coupled: starting with the same seed, they would produce the same T_A for the initial proposal, and the same count of rejections before the acceptance for the first time through the A procedure, with same T_A for that first acceptance, and so on, including the same number of iterations before finishing. Under this coupling, the original algorithm produces a partition μ of n, the variant algorithm produces a partition λ of n – and we have implicitly defined the deterministic bijection f with $\lambda = f(\mu)$.

Back to arguing that the algorithms are different. We believe that the coupling described in the preceding paragraph supplies rigorous proofs for the analogues of Theorems 3.3 and 3.4. For Theorem 3.5, however, one should also consider the computational cost of Euler's bijection, for various costing schemes, and we propose the following analogue, for the variant based on $p(z) = p_{\text{odd}}(z)\, p(z^2)$, combined with the trick of moving $\epsilon_1(x)$ from the A side to the B side, as in Section 3.5.1.

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

Theorem 3.6. Assume *that the cost* $D(n)$ *of proposing* $(Z_1(x), Z_2(x), \ldots, Z_n(x))$, *with* $x = x(n)$, *satisfies* $D(n) = n^\theta$ *times a slowly varying function of* n. *Assume* also *that the cost* $D_A(n)$ *of proposing* $A = (Z_1(x) - \epsilon_1(x), Z_3(x), Z_5(x), \ldots)$ *satisfies* $D_A(n) \sim D(n)/2$.

Then, the asymptotic cost of the entire algorithm is

$$\frac{1}{1 - 1/4^\theta} \sqrt{2}\, D(n)/2 \leqslant \sqrt{2}\, D(n).$$

Proof. Essentially the same as the proof of Theorem 3.5. □

It is plausible that the cost function $C(n)$ from Theorem 3.5 and the the cost function $D(n)$ from Theorem 3.6 are related by $C(n) \sim D(n)$; note that this depends on the choice of costing scheme, essentially asking whether or not the algorithmic cost of carrying out Euler's odd-distinct bijection is negligible.

3.6. Complexity considerations

In Remark 1 and at the end of Section 2.2 we note that in the general view of probabilistic divide-and-conquer algorithms, a key consideration is computability of the acceptance threshold $t(a)$. Recall that we write $p(n)$ for the number of integer partitions of size n. The case of integer partitions, using any of the divide-and-conquer algorithms of Section 3.5, is perhaps exceptionally easy, in that computing the acceptance threshold is essentially the same as evaluating $p(n)$, an extremely well-studied task.

For $n > 10^4$, a *single term* of the Hardy–Ramanujan asymptotic series suffices to evaluate $p(n)$ with relative error less than 10^{-16}; see Lehmer [27, 28].[6] This single term is

$$hr_1(n) := \frac{\exp(y)}{4\sqrt{3}\left(n - \frac{1}{24}\right)} \left(1 - \frac{1}{y}\right), \quad \text{where } y = 2c\sqrt{n - \frac{1}{24}}, \tag{3.16}$$

and numerical tabulation,[7] together with Lehmer's guarantee, shows that

$$\| \ln p(n) - \ln hr_1(n) \| < 10^{-16} \quad \text{for all } n \geqslant 489. \tag{3.17}$$

While equation (3.17) is sufficient for most practical aspects of computing, with probability 2^{-k} we will need more than k bits of $p(n)$ in order to determine the rejection step, with k at most $\lceil \log_2(p(n)) \rceil$. In the algorithm to sample a partition of size n, we start with (3.2) with the choice $x = x(n) = e^{-c/\sqrt{n}}$; the normalizing constant is then $c(x) := \prod_{i \geqslant 1}(1 - x^i)$, and (3.5) is explicitly

$$\mathbb{P}_x(T = j) = p(j)\, x^j \prod_{i \geqslant 1}(1 - x^i).$$

[6] We thank David Moews for bringing these papers to our attention.
[7] In Mathematica® 8.

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

After taking into account the factors of 2 in (1.5), and writing $y = x^2$, the threshold function $t(\cdot) \equiv t_n(\cdot)$ for Algorithm 6 is given by

$$t(\ell) = \frac{\mathbb{P}_y(T = \ell)}{\max_j \mathbb{P}_y(T = j)} = \frac{p(\ell)\, y^\ell}{\max_j p(j)\, y^j}, \tag{3.18}$$

where the rightmost expression is the result of cancellation of the factor $c(y)$ from the middle expression. Let us first focus on the numerator on the far right side of (3.18). The cost of evaluating $p(n)$ is not so straightforward, however, and an approach is outlined in [23] which is $O(n^{1/2+o(1)})$. If one wants an arithmetic cost bound that is better than $O(n^{1/2+o(1)})$, then the following is needed.

Lemma 3.7. *The arithmetic cost of obtaining the first r bits of $p(n)$ is $O(r \log^5(n))$.*

Proof. Our proof is a modification of the proof in Section 3 of [23], which assumes that all terms in the Rademacher series expansion [39] are evaluated with an absolute error of at most $2^{-3}/N$, where N is the number of terms of the Rademacher series taken. Let $s := \lfloor \log_2 p(n) \rfloor$ denote a lower bound on the number of non-trivial bits in $p(n)$. In our case, since the main contribution of error comes from truncating the series, and this error, in the notation of equation (3.19), is $R(n, N)$, we choose an N which bounds $R(n, N)$ by 2^{s-r-1}, and hence only require each term of the series to be evaluated with an absolute error of $2^{s-r-4}/N$, which is much less restrictive.

The details are straightforward and messy, and the punchline is that when $r = o(\sqrt{n})$, and even in fact for r up to a small constant times $\sqrt{n}$, the first two terms of the Rademacher series to approximate $p(n)$ suffice.[8] Thus, we require that each of the terms in the Rademacher series are evaluated with an absolute error of 2^{s-r-5} or less. Then we substitute this into [23, equation 3.13] and obtain that the cost of evaluating the leading r bits of $p(n)$ is at most

$$O\left(\sum_{k=1}^{2} \log k\, M(r) \log^2(r)\right) = O(r \log^4(n)),$$

where $O(M(r)\log^2(r))$ is the cost of evaluating the largest r bits of an elementary function,[9] and $M(r) = O(r \log^{1+o(1)}(r))$ is the cost of multiplying two r-bit numbers. When $r = \Omega(\sqrt{n})$, we may simply evaluate $p(n)$ exactly, which was shown to require $O(n^{1/2} \log^{4+o(1)}(n))$ arithmetic operations in [23], and hence we have opted for the simpler and concrete bound 5 in place of $4 + o(1)$.

[8] Empirically the first term suffices, but the error analysis in [39, 28] requires a second term.

[9] It is noted in [23] that there are two competing algorithms to evaluate the largest r bits of elementary functions, one that is $O(M(r)\log(r))$, and another that is $O(M(r)\log^2(r))$, but is faster in practice. Since logarithmic terms do not affect the conclusion of Lemma 3.8, we are content with the slightly larger estimate.

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

The details are as follows. Let $\mu = \frac{\pi}{6}(24n-1)^{1/2}$. The Rademacher series for $p(n)$, as given by [27], is

$$p(n) = \frac{\sqrt{12}}{24n-1} \sum_{k=1}^{N} \frac{A_k(n)}{k^{1/2}} \left(\left(1 - \frac{k}{\mu}\right) e^{\mu/k} + \left(1 + \frac{k}{\mu}\right) e^{-\mu/k} \right) + R(n, N),$$

where $A_k(n)$ is a complicated sum of $24k$th roots of unity, and

$$R(n, N) = \sum_{k=N+1}^{\infty} \frac{A_k(n)}{k^{1/2}} \left(\left(1 - \frac{k}{\mu}\right) e^{\mu/k} + \left(1 + \frac{k}{\mu}\right) e^{-\mu/k} \right). \tag{3.19}$$

Let $v := \mu/N$, and define

$$F(n, N) := \frac{N^{-2/3}\pi^2}{\sqrt{3}} \left(\frac{\sinh(v)}{v^3} + \frac{1}{6} - \frac{1}{v^2} \right).$$

It was shown in [27] that

$$|R(n, N)| < F(n, N)$$

for all positive integers n and N.

Let t_k denote the kth term in the Rademacher series expansion, and suppose $\widehat{t_k}$ is the numerical approximation stored in a computer. In order to compute the leading r bits of $p(n)$, we need

$$|t_k - \widehat{t_k}| < \frac{0.125 \times 2^{s-r}}{N}. \tag{3.20}$$

This differs from [23, equation (3.1)] by the extra factor of 2^{s-r}, which means we can be less precise when computing the value of t_k. A theorem analogous to [23, Theorem 4] thus holds; that is, for $n > 2000$, for equation (3.20) to hold, it is sufficient to evaluate t_k using a precision of

$$b = \max(\log_2(N) + \log_2(|t_k|) - (s-r) + \log_2(n) + 3, \tfrac{1}{2}\log_2(n) + 5, 11)$$

bits.

There are two cases to consider: $r = o(\sqrt{n})$ and $r = \Omega(\sqrt{n})$. When $r = \Omega(\sqrt{n})$, we may simply evaluate $p(n)$ exactly, which requires $O(n^{1/2}\log^{4+o(1)}(n))$ arithmetic operations. When $r = o(\sqrt{n})$, however, we note that $\sum_{i=1}^{N} t_k$ has an absolute error of at most $F(n, N)$, which is at most $e^{\mu/N}/2$. Thus, $t_1 + t_2$ guarantees *at least* $\log_2(e)\mu/2$ correct bits, up to a logarithmic correction factor which can be made explicit.

We then make the assumption, as in [23, Section 3.2], that r-bit floating-point numbers can be multiplied in time $M(r) = O(r\log^{1+o(1)} r)$. Then, to calculate the partial sum consisting of two terms, we evaluate

$$O\left(\sum_{k=1}^{2} (\log k)\, M(r) \log^2(r) \right) = O(r\, \log^4(n)),$$

where $M(r) = O(r\log^{1+o(1)}(r))$ is assumed to be the cost of multiplying two r-bit numbers. □

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

Lemma 3.8. *For random U distributed uniformly in $(0,1)$, the expected number of arithmetic operations for determining whether $U \leqslant t(\ell)$, where $t(\ell)$ is given in (3.18), is $O(\log^5(n))$.*

Proof. The probability of needing more than r bits is given by 2^{-r}, and so the total expected number of arithmetic operations for the evaluation of $t(\ell)$ is given by

$$O\left(\sum_{k\geqslant 1} k\,\log^5(n)\,2^{-k}\right) = O(\log^5(n)). \qquad \square$$

To deal with the denominator of (3.18), there are two strategies. The first strategy is to find the argument j which achieves $\max_j p(j)\,y^j$, that is, which achieves $\max_j(j\log y + \log p(j))$. Now, [15, 33] shows that $n \mapsto p(n)$ is log-concave for all $n > 25$, so bisection search over the range $j = 1$ to n requires $O(\log n)$ evaluations of $p(j)$. To be very careful, we must also note that we are dealing with *approximate* evaluations of $p(j)$; since $p(k) - p(k-1) \sim p(k)/\sqrt{k}$, we need only calculate the first $\log(n)$ bits of two terms of the form $j\log x + \log p(j)$ in order to determine which is greater, and Lemma 3.7 applies. The second strategy is to modify the threshold function $t(\ell)$ in (3.18), by giving away a factor of the form $1+\delta$ for some fixed $\delta > 0$; this has the effect of increasing the number of rejections, and hence the number of random bits used, by this same factor. In detail, (3.17) implies that $c_0 := \max_{j\geqslant 1} |\ln p(j) - \ln hr_1(j)| < \infty$, and one can easily evaluate c_0. Hence, with $1+\delta := \exp(c_0)$, we have $p(j) \leqslant (1+\delta)hr_1(j)$ for all $j \geqslant 1$, so the modified rejection threshold,

$$t'(\ell) := p(\ell)y^\ell / \max_j((1+\delta)hr_1(j)y^j),$$

satisfies $t' \leqslant 1$, and is a valid rejection threshold.

3.7. Partitions with restrictions

The self-similar recursive divide-and-conquer method of Section 3.5 is nearly unbeatable for large n, for unrestricted partitions. There are many classes of partitions with restrictions that iterate nicely, and should be susceptible to a corresponding recursive divide-and-conquer algorithm, *provided* efficient enumeration, analogous to (3.16), is available. Some of these classes, with their self-similar divisions, are

(1) distinct parts, $d(z) = d_{\text{odd}}(z)d(z^2)$;
(2) odd parts, $p_{\text{odd}}(z) = d_{\text{odd}}(z)p_{\text{odd}}(z^2)$;
(3) distinct odd parts, $d_{\text{odd}}(z) = d_*(z)d_{\text{odd}}(z^3)$.

Here

$$d_*(z) = (1+z)(1+z^5)(1+z^7)(1+z^{11})\cdots$$

represents distinct parts $\equiv \pm 1$ mod 6. Other recurrences are discussed in [24, 36, 40], and the standard text on partitions [4].

It is straightforward to apply deterministic second half to restricted partitions, as the rejection probability is given explicitly by a ratio of geometric probabilities. To apply recursive PDC to restricted partitions, one must have some means of explicitly bounding

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

the error in any approximation to the number of partitions subject to restrictions, in order to evaluate the functional $\mathbb{1}(U < t(a))$. As Section 3.6 shows, this is accessible, via the work of Rademacher [39] and Lehmer [27, 28], when there are no restrictions. When the restrictions are of a certain form, then there are completely effective error rates available [41]. Other generalizations to Rademacher's method are also available [2, 1, 32, 19].

4. Probabilistic divide-and-conquer with mix-and-match

4.1. Algorithmic implications of the basic lemma

Assume that one wants a sample of fixed size m from the distribution of S. That is, one wants to carry out a simulation that provides $S_1, S_2, \ldots,$ with $S_1, S_2, \ldots, S_m$ being independent, with each equal in distribution to S. According to Lemma 2.1, this can be done by providing m independent pairs (X_i, Y_i), $i = 1$ to m, each equal in distribution to S.

A reasonable choice of how to carry this out (not using mix-and-match) is given in Algorithm 7. Note that in general, conditional on the result of stage 1, the Y_i in stage 2 are *not identically* distributed.

Algorithm 7 Probabilistic divide-and-conquer m samples

1 Generate $X_1, X_2, \ldots, X_m$ i.i.d. from $\mathcal{L}(A \mid h(A, B) = 1)$.
2 Conditional on $(X_1, \ldots, X_m) = (x_1, \ldots, x_m)$,
  generate $Y_1, Y_2, \ldots, Y_m$, independent, with distributions
  $\mathcal{L}(Y_i) = \mathcal{L}(B \mid h(x_i, B) = 1)$ for $i = 1, 2, \ldots, m$.
3 Return $((X_1, Y_1), (X_2, Y_2), \ldots, (X_m, Y_m))$.

4.2. *Simple* matching enables mix-and-match

An important class of PDC sampling algorithms are those for which the matching condition $h(A, B) = 1$ defines a disjoint union of complete bipartite graphs; that is, there exists a (set) partition $I_1, I_2, \ldots$ of $\mathcal{A}$ and a corresponding (set) partition $J_1, J_2, \ldots$ of $\mathcal{B}$ such that $h(A, B) = 1$ if and only if $A \in I_i$ and $B \in J_i$, $i = 1, 2, \ldots$. When this condition holds, we call the matching *simple*, and say that *mix-and-match* is enabled.

The following lemma serves to clarify the logical structure of what is needed to enable mix-and-match.

Lemma 4.1. *Given $h : \mathcal{A} \times \mathcal{B} \to \{0, 1\}$, the following two conditions are equivalent.*

Condition 1: $\forall a, a' \in \mathcal{A}, b, b' \in \mathcal{B}$,

$$1 = h(a, b) = h(a', b) = h(a, b') \text{ implies } h(a', b') = 1. \tag{4.1}$$

Condition 2: $\exists \mathcal{C}$, *and functions* $c_{\mathcal{A}} : \mathcal{A} \to \mathcal{C}, c_{\mathcal{B}} : \mathcal{B} \to \mathcal{C}$, *so that* $\forall (a, b) \in \mathcal{A} \times \mathcal{B}$,

$$h(a, b) = 1(c_{\mathcal{A}}(a) = c_{\mathcal{B}}(b)). \tag{4.2}$$

□

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

We think of $\mathcal{C}$ as a set of *colours*, so that condition (4.2) says that a and b match if and only if they have the same colour.

Remark 6. When (4.2) holds, we can write the event that A matches B as a union indexed by the colour involved:

$$\{h(A,B)=1\} = \cup_{k\in\mathcal{C}}\{c_{\mathcal{A}}(A)=k, c_{\mathcal{B}}(B)=k\},$$

so that

$$p = \sum_{k\in\mathcal{C}} \mathbb{P}(c_{\mathcal{A}}(A)=k, c_B(B)=k),$$

and we see that at most a countable set of colours k contribute a strictly positive amount to p. As a notational convenience, we take $\mathbb{N}\subset\mathcal{C}$, and use positive integers k for the names of colours that have

$$\mathbb{P}(c_{\mathcal{A}}(A)=k, c_B(B)=k) > 0. \tag{4.3}$$

Remark 7. Here is an example which is *not* simple, that is, it does not satisfy (4.2). The configuration method for random r-regular graphs on n vertices, [10, Section 2.4], involves a uniform choice over a set of size $(r\ n-1)!!$. For any choice $1<b<n$, one might take $\mathcal{A},\mathcal{B}$ with $|\mathcal{A}\times\mathcal{B}| = (r\ n-1)!!, |\mathcal{B}| = (2b-1)!!$, so that A corresponds to the first $\frac{rn}{2}-b$ choices that need to be made to specify a configuration, and B corresponds to the final b choices. The matching function h is given by $h(A,B)$, the indicator that the multigraph implied by the configuration (A,B) has no loops and no multiple edges.

Remark 8. Observe that the matching function from Section 3 with $h(A,B) = 1(T_A + T_B = n)$, for $(A,B)\in\mathcal{A}\times\mathcal{B}$, satisfies condition 2 of Lemma 4.1, with $c_{\mathcal{A}}(A) = T_A$ and $c_{\mathcal{B}}(B) = n - T_B$. Hence *mix-and-match* is enabled.

The intent of the following lemma is to show that if h satisfies (4.2), then mix-and-match strategies can be used in stage 2 of the broad outline of Section 4.1. We consider a procedure which proposes a sequence $D_1, D_2, \ldots$ of elements of $\mathcal{B}$ with the following properties: we assume there is a sequence of σ-algebras $\mathcal{F}_0\subset\mathcal{F}_1\subset\mathcal{F}_2\subset\cdots$ for which D_n is $\mathcal{F}_n$ measurable. We think of $\mathcal{F}_0$ as carrying the information from stage 1 of an algorithm along the lines described in Section 4.1, carrying information such as 'which demands $a_1, a_2, \ldots$ must be met' – or reduced information, such as the colours $c_{\mathcal{A}}(a_1),\ldots,c_{\mathcal{A}}(a_m)$.

Lemma 4.2. *Assume that h satisfies* (4.2) *and that the following conditions hold.*

(1) *For every $n\geqslant 1$ and k satisfying* (4.3)*, conditional on $\mathcal{F}_{n-1}$ together with $c_{\mathcal{B}}(D_n)=k$, the distribution of D_n is equal to $\mathcal{L}(B|c_{\mathcal{B}}(B)=k)$.*

(2) *For every k satisfying* (4.3)*,*

$$\text{with probability 1, infinitely many } n \text{ have } c_{\mathcal{B}}(D_n)=k. \tag{4.4}$$

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

(3) *Define stopping times* $\tau_i^{(k)}$, *the 'time n of the ith instance of* $c_{\mathcal{B}}(D_n) = k$', *by* $\tau_0^{(k)} = 0$ *and for* $i \geqslant 1$,

$$\tau_i^{(k)} = \inf\{n > \tau_{i-1}^{(k)} : c_{\mathcal{B}}(D_n) = k\}.$$

We write $D(n) \equiv D_n$ *to avoid multi-level subscripting, and define* $B_i^{(k)} := D(\tau_i^{(k)})$ *for* $i = 1, 2, \ldots$.

Then, *for each k, the* $B_1^{(k)}, B_2^{(k)}, \ldots$ *are independent, with the distribution* $\mathcal{L}(B|c_{\mathcal{B}}(B) = k)$, *and as k varies, these sequences are independent.*

Proof. The proof is a routine exercise by summing over all possible values for the random times $\tau_i^{(k)}$. Writing out the full argument would be notationally messy, and not interesting. □

4.3. Divide-and-conquer with mix-and-match

When a sample of size $m > 1$ is desired, and the matching function is simple, lemmas 2.1, 4.1 and 4.2 combine to suggest the following algorithm. Stage 1 of the algorithm generates samples $A_1, A_2, \ldots, A_m$ from X, according to Lemma 2.1. This creates a multiset of m colours, $\{c_1, \ldots, c_m\}$, where $c_i = c_{\mathcal{A}}(A_i)$. We think of these as m demands that must be met by stage 2 of the algorithm. One strategy for stage 2 is to generate an i.i.d. sequence of samples of B; initially, for each sample, we compute its colour $c = c_{\mathcal{B}}(B)$ and check whether c is in the multiset of demands $\{c_1, \ldots, c_m\}$; when we find a match, we pair B with one of the A_i of the matching colour, to produce our first sample, which we set as $S_i = (A_i, B)$. Then we reduce the multiset of demands by one element, and iterate, until all m demands have been met. Lemma 4.2 implies that the resulting list $(S_1, S_2, \ldots, S_m)$ is an i.i.d. sample of m values of S, as desired.

A simple algorithmic view of the mix-and-match PDC algorithm is shown in Algorithm 8, which assumes that mix-and-match is enabled. We refer to step 1 as the 'A phase' and step 2 as the 'B phase'. The A phase can be performed using the techniques of previous sections. The B phase is straightforward since we are sampling from the *unconditional* distribution $\mathcal{L}(B)$.

Algorithm 8 Probabilistic divide-and-conquer mix-and-match

1 Generate $X_1, X_2, \ldots, X_m$ i.i.d. from $\mathcal{L}(A \mid h(A, B) = 1)$.
2 Generate $B_1, B_2, \ldots$ i.i.d. from $\mathcal{L}(B)$ until all the X_i have a match.
3 Return $((X_1, B_{j_1}), (X_2, B_{j_2}), \ldots, (X_m, B_{j_m}))$.

Remark 9. For step 3, it is important to note that *the order in which completed samples are returned is not arbitrary!* In particular:

(a) We cannot report the sample in the order in which matching B_{j_i} are found. That is, if we let $(j_1), (j_2), \ldots, (j_m)$ denote the ordering of the sequence $j_1, \ldots, j_m$ from smallest to largest, and $i_1, \ldots, i_m$ the corresponding indices for the matching samples from the

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

A phase, then in general

$$((X_{1_1}, B_{(j_1)}), (X_{i_2}, B_{(j_2)}), \ldots, (X_{i_m}, B_{(j_m)}))$$

is *not* an i.i.d. sample from $\mathcal{L}(S)$.

(b) Continuing with the previous point, we can either apply a random permutation to the list, or report the sample in the order in which the X_i are generated in the A phase, as we have done in step 3.

The practical significance of item (a) above is that if we are running a simulation for m samples, and we wish to abort the simulation early and/or print out partial results for a smaller sample size $r \leqslant m$, *we can only certify that* $((X_1, B_{j_1}), \ldots, (X_r, B_{j_r}))$ *is an i.i.d. sample of size* r *from* $\mathcal{L}(S)$ *if every* X_i, $i \leqslant r$, *has been matched with a* B_{j_i}.

4.3.1. Roaming x. Consider a sample of size $m = 1$. Having accepted $X = (Z_{b+1}, \ldots, Z_n)$ with colour $k = T_A$, in the notation of (3.12), we now need Y, which is $B = (Z_1, \ldots, Z_b)$ conditional on having colour k, which simplifies to having $n - k = T_B := \sum_{i=1}^{b} iZ_i$. One obvious strategy is to sample B repeatedly until we get lucky. The distribution of B is specified by (3.2) and (3.6) – with a choice of parameter, $x = x(n)$, not taking into account the values of b and $n-k$. A computation similar to (3.4) shows that the distribution of $(Z_1(y), \ldots, Z_b(y))$ conditional on $\left(\sum_{i=1}^{b} iZ_i(y) = n - k\right)$ is the same, for all choices $y \in (0, 1)$.

As observed in [7, Section 5, page 114)], the y which maximizes $\mathbb{P}_y(T_B = n - k)$ is the solution of $n - k = \sum_{i=1}^{b} \mathbb{E}\, iZ_i(y)$. Thus, in the case $m = 1$, the optimal choice of y is easily prescribed. However, for large m, using mix-and-match brings into play a complicated coupon collector's situation. With a multiset of demands $\{c_1, \ldots, c_m\}$ from Section 4.3, the freedom to allow y to roam allows us to tilt the distribution in response to the demands that remain at each stage. The algorithm designer has many choices of global strategy. Based on computer experiments, it is not obvious whether or not a greedy strategy – picking y to maximize the chance that the next proposed B satisfies at least one of the demands – is optimal.

5. Computational considerations for implementing PDC on integer partitions

In Section 3, several methods were presented for the simulation of partitions of the integer n. The analysis focused on the asymptotic rejection probabilities, which varied by choice of (A, B).

The punchline is that instead of rejection sampling (Algorithm 4), the default algorithm should always be PDC deterministic second half with von Neumann's rejection method (Algorithm 5). It is guaranteed faster than rejection sampling, requires the same amount of memory, and under various restrictions on the part of an integer partition it is still applicable even when self-similar bijections do not exist. In addition, it is no more difficult to program on a computer.

Then, if more efficiency is needed, one can weigh the costs of storing all or part of a table of values or exploring the existence of self-similar bijections.

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

5.1. On proposing an instance of the independent process

All of the algorithms for integer partitions centre around the generation of variates from the process of independent random variables $\mathbf{Z} = (Z_1, Z_2, \ldots, Z_n)$, sampled under (3.2) and (3.6). The number of random bits needed to sample from this process is at least the (base 2) entropy $H(\mathbf{Z})$. With x chosen as in (3.6), it is non-trivial[10] to see that this entropy is asymptotically $\log_2(p(n))$, and hence, by Hardy–Ramanujan (3.10), asymptotic to $(2/\ln 2)c\sqrt{n}$, with $c = \pi/\sqrt{6}$.

5.1.1. Naïve proposals. The simplest procedure for sampling from the independent process $\mathbf{Z} = (Z_1, Z_2, \ldots, Z_n)$ is to sample each coordinate separately, using the fact that, if Z is geometrically distributed with parameter p, then $Z =_d \lfloor \ln(U)/\ln(1-p) \rfloor$ for U uniform over the interval $(0, 1)$. Iterating through all coordinates supplies a $O(n)$ procedure for sampling $\mathbf{Z}$.

Erdős and Lehner [20] showed that with probability tending to 1, the largest part is close to $2c\sqrt{n}\log(n)$, and that the number of part-sizes, corresponding to the number of non-zero Z_i in $\mathbf{Z}$, is close to $(1/c)\sqrt{n}$, with $c = \pi/\sqrt{6}$. This implies that with high probability $Z_i = 0$ for all $i \gg \sqrt{n}\log(n)$. There are several adaptations one can make, such as pooling the variables together under a single uniform random variable, but the implementations are messy.

A variation due to Sheldon Ross[11] would generate L, the largest index for which $Z_j > 0$, whose distribution is given by

$$\mathbb{P}(L = j) = x^j \prod_{k>j} (1 - x^k).$$

Conditional on $\{L = j\}$ for some $j > 0$, the distribution of $\mathbf{Z}$ is equal to the distribution of the vector $(Z_1, Z_2, \ldots, 1 + Z_j, 0, 0, \ldots)$.

5.1.2. A proposal on the same order as the lower entropy bound. Our recommended proposal is one that takes advantage of the relation between geometric and Poisson random variables. See [9] for an alternative description.

A geometric random variable Z with parameter $0 < a < 1$ can be represented as a sum of independent Poisson random variables Y_j, $j = 1, 2, \ldots$, as $Z = \sum_j jY_j$, where $\mathbb{E}\, Y_j = a^j/j$ (this is easily verified using generating functions). The random variables Y_j can be generated via a Poisson process as follows. Let $r = \sum_j a^j/j$, and divide the interval $[0, r]$ into disjoint intervals of length $a, a^2/2, a^3/3$, *etc.* Then $Y_j = k$ if exactly k arrivals occur in the interval of length a^j/j.

[10] Using the notation of entropy and conditional entropy as in [11],

$$H(\mathbf{Z}) = \sum_m \mathbb{P}(T = m) H(\mathbf{Z}|T = m) + H(T) \geqslant \sum_m \mathbb{P}(T = m) H(\mathbf{Z}|T = m) = \sum_m \mathbb{P}(T = m) \log_2(p(m)).$$

Then using Chebyshev's inequality together with (3.8), we see that the sum can be restricted to $m \geqslant m_0 = (1-\varepsilon)n$ with $\mathbb{P}(T \geqslant m_0) \to 1$ and $\varepsilon \to 0$, which proves the lower bound. The upper bound follows by observing that $H(T) = o(\log(p((1-\varepsilon)n)))$ for all ε sufficiently small.

[11] Personal communication.

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

To simulate the vector $(Z_1(x), Z_2(x), \ldots, Z_i(x), \ldots)$, with parameters $a = x, x^2, \ldots, x^i, \ldots$, we fix disjoint intervals of length x^{ij}/j for $i, j \geqslant 1$, and run a Poisson process on the interval $[0, s]$, where $s = \sum_{i,j} x^{ij}/j$.

Our claim that we have an algorithm using $O(\sqrt{n})$ calls to a random number generator is supported by the calculation here that, with $x = x(n) = \exp(-c/\sqrt{n})$ and $c = \pi/\sqrt{6}$,

$$s(n) := \sum_{i,j \geqslant 1} \frac{x^{ij}}{j} = \sum_j \frac{1}{j^2} \frac{jx^j}{1-x^j} \leqslant \frac{\pi^2}{6} \frac{x}{1-x} \sim c\sqrt{n}; \tag{5.1}$$

the inequality follows from the observation that for $0 < x < 1$,

$$jx^j \leqslant x + x^2 + \cdots + x^j \leqslant x(1-x_j)/(1-x).$$

5.2. Floating-point considerations and coin tossing

Is floating-point accuracy sufficient, in the context of computing an acceptance threshold $t(a)$? There is a very concrete answer, based on [25]; see [11, Section 5.12] for an accessible exposition. First, given $p \in (0, 1)$, a p-coin can be tossed using a random number of fair coin tosses; the expected number is exactly 2, unless p is a kth level dyadic rational, that is, $p = i/2^k$ with odd i, in which case the expected number is $2 - 2^{1-k}$. The proof is by consideration of say $B, B_1, B_2, \ldots$ i.i.d. with $\mathbb{P}(B = 0) = \mathbb{P}(B = 1) = 1/2$; after r tosses we have determined the first r bits of the binary expansion of a random number U which is uniformly distributed in $(0, 1)$, and the usual simulation recipe is that a p-coin is the indicator $\mathbb{1}(U < p)$. Unless $\lfloor 2^r p \rfloor = \lfloor 2^r U \rfloor$, the first r fair coin tosses will have determined the value of the p-coin. Exchanging the roles of U and p, we see that the number of bits of precision read from p is, on average, 2, and exceeds r with probability 2^{-r}. If a floating-point number delivers 50 bits of precision, the chance of needing more precision is 2^{-50} per evaluation of an indicator of the form $\mathbb{1}(U < p)$. Our divide-and-conquer does not require very many acceptance/rejection decisions; for example, with $n = 2^{60}$, there are about 30 iterations of the algorithm in Theorem 3.4, each involving on average about $\sqrt{2}$ acceptance/rejection decisions, according to Theorem 3.3. So one might program the algorithm to deliver exact results; most of the time determining acceptance thresholds $p = t(a)$ in floating-point arithmetic, but keeping track of whether more bits of p are needed. On the unlikely event, of probability around $30 \times \sqrt{2}/2^{50} < 4 \times 10^{-14}$, that more precision is needed, the program demands a more accurate calculation of $t(a)$. This would be far more efficient than using extended integer arithmetic to calculate values of $p(n)$ exactly; see Remark 1 and Section 3.6.

Another place to consider the use of floating-point arithmetic is in proposing the vector $(Z_1(x), \ldots, Z_n(x))$. If one call to the random number generator suffices to find the next arrival in a rate 1 Poisson process, we have an algorithm using $O(\sqrt{n})$ calls, which can propose the entire vector $(Z_1, Z_2, \ldots)$, using $x = x(n)$ from (3.6). The proposal algorithm, summarized in Section 5.1.2, is based on a compound Poisson representation of geometric distributions, and is similar to a coupling used in [5], Section 3.4.1.

Once again, suppose we want to guarantee *exact* simulation of a proposal $(Z_1(x), \ldots, Z_n(x))$. In the standard rate 1 Poisson process, run up to time $s(n)$, given by (5.1), we need to assign *exactly*, for each arrival, say at a random time R, the corresponding

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

index (i, j), such that the partial sum for $s(n)$ up to, but excluding the ij term, is less than R, but the partial sum, including the ij term, is greater than or equal to R. Based on an entropy result from Knuth and Yao [25], a crucial quantity is

$$h(n) := \sum_{i,j \geqslant 1} \frac{x^{ij}}{j} \ln \frac{j}{x^{ij}} \leqslant (c - \zeta'(2)/c)\ \sqrt{n}. \tag{5.2}$$

An exact simulation of the Poisson process, assigning ij labels to each arrival, can be done[12] with $O(s(n) + h(n))$ genuine random bits, and the bounds for $s(n)$ and $h(n)$ show that this is $O(\sqrt{n})$.

Acknowledgements

The authors would like to thank the anonymous referees for comments that contributed to the improvement of this paper, and to Fredrik Johansson for help with the proof of Lemma 3.7.

References

[1] Almkvist, G. (1991) Exact asymptotic formulas for the coefficients of nonmodular functions. *J. Number Theory* **38** 145–160.
[2] Almkvist, G. and Andrews, G. E. (1991) A Hardy–Ramanujan formula for restricted partitions. *J. Number Theory* **38** 135–144.
[3] Alonso, L. (1994) Uniform generation of a Motzkin word. *Theoret. Comput. Sci.* **134** 529–536.
[4] Andrews, G. E. (1984) *The Theory of Partitions*, Cambridge Mathematical Library.
[5] Arratia, R. (2002) On the amount of dependence in the prime factorization of a uniform random integer. In *Contemporary Combinatorics*, Vol. 10 of *Bolyai Society Mathematical Studies*, János Bolyai Mathematical Society, pp. 29–91.
[6] Arratia, R., Gordon, L. and Waterman, M. S. (1990) The Erdős–Rényi law in distribution, for coin tossing and sequence matching. *Ann. Statist.* **18** 529–570.
[7] Arratia, R. and Tavaré, S. (1994) Independent process approximations for random combinatorial structures. *Adv. Math.* **104** 90–154.
[8] Arratia, R. and Waterman, M. S. (1985) Critical phenomena in sequence matching. *Ann. Probab.* **13** 1236–1249.
[9] Bodini, O., Fusy, E. and Pivoteau, C. (2010) Random sampling of plane partitions. *Combin. Probab. Comput.* **19** 201–226.
[10] Bollobás, B. (2001) *Random Graphs*, second edition, Vol. 73 of *Cambridge Studies in Advanced Mathematics*, Cambridge University Press.
[11] Cover, T. M. and Thomas, J. A. (1991) *Elements of Information Theory*, Wiley Series in Telecommunications, Wiley.
[12] Denise, A. and Zimmermann, P. (1999) Uniform random generation of decomposable structures using floating-point arithmetic. *Theoret. Comput. Sci.* **218** 233–248.

[12] On each interval $(m-1, m]$ for $m = 1$ to $\lceil s(n) \rceil$, perform an exact simulation of the number of arrivals, which is distributed according to the Poisson distribution with mean 1. For each arrival on $(m-1, m]$, there is a discrete distribution described by those partition points lying in $(m-1, m)$, together with the endpoints $m-1$ and m; calling the corresponding random variable X_m, the sum of the base 2 entropies satisfies $\sum h(X_m) \leqslant h(n) + s(n)$, since each extra subdivision of one of the original subintervals of length x^{ij}/j adds at most one bit of entropy.

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

[13] DeSalvo, S. (2012) Probabilistic divide-and-conquer: a new exact simulation method, with integer partitions as an example and lower bound expansions for random Bernoulli matrices via novel integer partitions. PhD thesis, University of Southern California.
[14] DeSalvo, S. (2014) Probabilistic divide-and-conquer: deterministic second half. `arXiv:1411.6698`
[15] DeSalvo, S. and Pak, I. (2015) Log-concavity of the partition function. *Ramanujan J.* **38** 61–73.
[16] Devroye, L. (1986) *Non-Uniform Random Variate Generation*, Springer.
[17] Diffie, W. and Hellman, M. E. (1977) Special feature exhaustive cryptanalysis of the NBS data encryption standard. *Computer* **10** 74–84.
[18] Duchon, P., Flajolet, P., Louchard, G. and Schaeffer, G. (2004) Boltzmann samplers for the random generation of combinatorial structures. *Combin. Probab. Comput.* **13** 577–625.
[19] Engel, B. (2014) Log-concavity of the overpartition function. arXiv: 1412.4603
[20] Erdős, P. and Lehner, J. (1941) The distribution of the number of summands in the partitions of a positive integer. *Duke Math. J.* **8** 335–345.
[21] Fristedt, B. (1993) The structure of random partitions of large integers. *Trans. Amer. Math. Soc.* **337** 703–735.
[22] Hardy, G. H. and Ramanujan, S. (1918) Asymptotic formulae in combinatory analysis. *Proc. London Math. Soc.* s2–17 (1) 75–115.
[23] Johansson, F. (2012) Efficient implementation of the Hardy–Ramanujan–Rademacher formula. *LMS J. Comput. Math.* **15** 341–359.
[24] Knessl, C. and Keller, J. B. (1990) Partition asymptotics from recursion equations. *SIAM J. Appl. Math.* **50** 323–338.
[25] Knuth, D. E. and Yao, A. C. (1976) The complexity of nonuniform random number generation. In *Algorithms and complexity: Proc. Sympos., Carnegie-Mellon University, Pittsburgh, PA, 1976*, Academic Press, pp. 357–428.
[26] Lam, T., Lapointe, L., Morse, J. and Shimozono, M. (2010) Affine insertion and Pieri rules for the affine Grassmannian. *Mem. Amer. Math. Soc.* **208**(977).
[27] Lehmer, D. H. (1938) On the series for the partition function. *Trans. Amer. Math. Soc.* **43** 271–295.
[28] Lehmer, D. H. (1939) On the remainders and convergence of the series for the partition function. *Trans. Amer. Math. Soc.* **46** 362–373.
[29] Mathematica (2010) *Mathematica Edition: Version 8.0*, Wolfram Research, Inc.
[30] MATLAB (2011) *Version 7.12.0.635 (R2011a)*, The MathWorks, Inc.
[31] McKay, B. D. and Wormald, N. C. (1990) Uniform generation of random regular graphs of moderate degree. *J. Algorithms* **11** 52–67.
[32] McLaughlin, J. and Parsell, S. (2012) A Hardy–Ramanujan–Rademacher-type formula for (r, s)-regular partitions. *Ramanujan J.* **28** 253–271.
[33] Nicolas, J.-L. (1978) Sur les entiers N pour lesquels il y a beaucoup de groupes abéliens d'ordre N. *Ann. Inst. Fourier (Grenoble)* **28** 1–16.
[34] Nijenhuis, A. and Wilf, H. S. (1975) A method and two algorithms on the theory of partitions. *J. Combin. Theory Ser. A* **18** 219–222.
[35] Nijenhuis, A. and Wilf, H. S. (1978) *Combinatorial Algorithms: For Computers and Calculators*, Academic Press.
[36] Pak, I. (2006) Partition bijections: A survey. *Ramanujan J.* **12** 5–75.
[37] Pittel, B. (1997) On a likely shape of the random Ferrers diagram. *Adv. Appl. Math.* **18** 432–488.
[38] Pittel, B. (1997) Random set partitions: asymptotics of subset counts. *J. Combin. Theory Ser. A* **79** 326–359.
[39] Rademacher, H. (1937) A convergent series for the partition function $p(n)$. *Proc. Nat. Acad. Sci.* **23** 78–84.

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press

[40] Remmel, J. B. (1982) Bijective proofs of some classical partition identities. *J. Combin. Theory Ser. A* **33** 273–286.
[41] Sills, A. V. (2010) Rademacher-type formulas for restricted partition and overpartition functions. *Ramanujan J.* **23** 253–264.
[42] von Neumann, J. (1951) Various techniques used in connection with random digits. *Journal of Research of the National Bureau of Standards, Appl. Math. Series* **3**, 36–38.

https://doi.org/10.1017/S0963548315000358 Published online by Cambridge University Press