Efficient Sampling Methods for Discrete Distributions

Karl Bringmann and Konstantinos Panagiotou

Max Planck Institute for Informatics
Campus E1.4, 66123 Saarbrücken, Germany

Abstract We study the fundamental problem of the exact and efficient generation of random values from a finite and discrete probability distribution. Suppose that we are given n distinct events with associated probabilities $p_1, \dots, p_n$. We consider the problem of sampling a subset, which includes the ith event independently with probability p_i, and the problem of sampling from the distribution, where the ith event has a probability proportional to p_i. For both problems we present on two different classes of inputs – sorted and general probabilities – efficient preprocessing algorithms that allow for asymptotically optimal querying, and prove almost matching lower bounds for their complexity.

1 Introduction

Generating random variables from finite and discrete distributions has long been an important building block in many applications. For example, in computer simulations usually a huge number of random decisions based on prespecified or dynamically changing distributions is made. In this work we consider two fundamental computational problems, namely sampling *independent events* and sampling *from a distribution*, on two different classes of inputs, sorted and unsorted probabilities. As we will see, there is a rich interplay in designing efficient algorithms that solve these different variants.

Our results are valid in the classical RealRAM model [1, 9] of computation. In particular, we will assume that the following operations take constant time:

- Accessing the content of any memory cell.
- Generating a uniformly distributed real number in the interval $[0, 1]$.
- Performing any basic arithmetical operation involving real numbers like addition, multiplication, division, comparison, truncation, and evaluating any fundamental function like exp and log.

In the remainder, we will abbreviate $[n] = \{1, \dots, n\}$ and we will write $\ln x$ for the natural logarithm of x and $\log x$ for the binary logarithm of x. Finally, we will write rand() for a uniform random number in $[0, 1]$.

1.1 Subset Sampling

We consider n independent events with indicator random variables $X_1, \ldots, X_n$, and $\Pr[X_i = 1] = p_i$. For shortcut we write $\mu = \mu_{\mathbf{p}} = \sum_{i=1}^{n} p_i = \mathbb{E}[\sum_{i=1}^{n} X_i]$ and $\mathbf{p} = (p_1, \ldots, p_n)$. Consider the random variable $X = X_{\mathbf{p}} = \{i \in [n] \mid X_i = 1\}$, which is the set of all events that occurred.

We concern ourselves with the problem of sampling X. We study this problem on two different classes of input sequences, sorted and general (i.e., not necessarily sorted) sequences; dependent on the class under consideration we call the problem SortedSubsetSampling or UnsortedSubsetSampling.

A *single-sample algorithm* for SortedSubsetSampling or UnsortedSubsetSampling gets input $\mathbf{p}$ and outputs a set $S \subseteq [n]$ that has the same distribution as X. Such an algorithm cannot run faster than $\mathcal{O}(1+\mu)$, as its expected output size is μ and any algorithm requires a running time of $\Omega(1)$. This runtime, however, is in general not achievable, as our results below make more precise. Hence, we consider a *preprocessing-query* variant of the problem, where we want to be able to answer queries in the optimal expected runtime of $\mathcal{O}(1 + \mu)$ after a certain preprocessing.

In the preprocessing-query variant we consider the interplay of two algorithms. First, the *preprocessing algorithm* P gets $\mathbf{p}$ as input and computes some auxiliary data $D = D(\mathbf{p})$. Second, the *query algorithm* Q gets input $\mathbf{p}$ and D, and samples X, i.e., for any $S \subseteq [n]$ we have $\Pr[Q(\mathbf{p}, D) = S] = \Pr[X_{\mathbf{p}} = S]$. Here Pr goes only over the random choices of Q, so that, after running the preprocessing once, running the query algorithm multiple times generates multiple independent samples. Note that if the preprocessing time is p and the query time is q, then we can generate a single sample of X in time $p + q$, so the single-sample variant of the problem is also solved by the preprocessing-query variant. In this paper we will not consider single-sample algorithms any further, because our constructed preprocessing-query algorithms are already for a *single* query as efficient as the best single-sample algorithm we can devise. This holds for all problem variants we consider.

The single-sample variant of UnsortedSubsetSampling can be solved trivially in time $\mathcal{O}(n)$; we just toss a biased coin for every p_i. A classic algorithm solves this problem for $p_1 = \ldots = p_n = p$ in the optimal expected time $\mathcal{O}(1+\mu)$, see e.g. the monographs [2] by Devroye and [5] by Knuth, where also many other cases are discussed. Indeed, observe that the index i_1 of the first sampled element is geometrically distributed, i.e., $\Pr[i_1 = i] = (1-p)^{i-1}p$. Such a random value can be generated by setting $i_1 = \lfloor \frac{\log \mathrm{rand}()}{\log(1-p)} \rfloor$. Moreover, after having sampled the index of the first element, we iterate the process starting at $i_1 + 1$ to sample the second element, and so on, until we arrive for the first time at an index $i_k > n$.

In this paper we generalize this bound for equal probabilities as far as possible. More precisely, we ask whether the optimal query time $\mathcal{O}(1+\mu)$ is achievable for larger classes of inputs and how much preprocessing is needed. We obtain the following answers.

Theorem 1. SortedSubsetSampling *can be solved in* $\mathcal{O}(\log n)$ *preprocessing time and* $\mathcal{O}(1+\mu)$ *expected query time. Moreover, the bound on the preprocessing*

time is nearly tight, as the sum of preprocessing and query time is $\Omega\left(\frac{\log n}{\log\log n}\right)$ *for any such algorithm, as* $n \to \infty$ *and* $\mu = \mu(n) \geqslant (\log n)^{-\mathcal{O}(1)}$.

Due to space limitations, the proof of the lower bound of Theorem 1 can be found in the appendix.

To avoid any confusion, note that we mean worst-case bounds whenever we speak of *(running) time* and expected bounds whenever we speak of *expected (running) time*. The next result addresses the case where the probabilities are not necessarily sorted.

Theorem 2. UnsortedSubsetSampling *can be solved in* $\mathcal{O}(n)$ *preprocessing time and* $\mathcal{O}(1+\mu)$ *expected query time. Moreover, this is optimal, as even any single-sample algorithm for* UnsortedSubsetSampling *needs time* $\Omega(n)$.

Both positive results in the previous theorems depend highly on each other. In particular, as it is demonstrated in Section 3, we prove them by repeatedly reducing the instance size n and switching from the one problem variant to the other.

The problem of UnsortedSubsetSampling was considered also recently in the two papers [11, 12], where algorithms with linear preprocessing time and suboptimal query time $\mathcal{O}(\log n + \mu)$ were designed. Thus, our results improve upon these running times, and provide accompanying and (almost) matching lower bounds.

1.2 Proportional Sampling

In the previous section we considered the problem of sampling subsets. Here we will focus on a slightly different and more classical problem. Given $\mathbf{p} = (p_1, \ldots, p_n) \in \mathbb{R}^n_{\geqslant 0}$, we define a random variable $Y = Y_{\mathbf{p}}$ that takes values in $[n]$ such that $\Pr[Y = i] = p_i/\mu$, where again $\mu = \sum_{i=1}^n p_i$. We call the problem of sampling Y SortedProportionalSampling or UnsortedProportionalSampling, if we consider it on sorted or general input sequences, respectively.

As previously, we consider two variations of the problem. In the *single-sample* variant we are given $\mathbf{p}$ and we want to compute an output that has the same distribution as Y. Moreover, in the *preprocessing-query* variant we have a precomputation algorithm that, given $\mathbf{p}$, computes some auxiliary data D, and a query algorithm that is given $\mathbf{p}$ and D and has an output with the same distribution as Y.

In this setting, we no longer output μ elements. So, it could be that the optimal expected query time reduces to $\mathcal{O}(1)$. For sorted sequences, this optimal query time can be indeed achieved after a relatively small preprocessing time, as the next result shows.

Theorem 3. SortedProportionalSampling *can be solved in* $\mathcal{O}(\log n)$ *preprocessing time and* $\mathcal{O}(1)$ *expected query time.*

For general input sequences, this problem can be solved by the technique known as *pairing* or *aliasing* [5, 13]. This result is not new, but will be used in the proofs of Theorem 1 and Theorem 2, so we include it for completeness.

Theorem 4. UnsortedProportionalSampling *can be solved in* $\mathcal{O}(n)$ *preprocessing time and* $\mathcal{O}(1)$ *query time. Moreover, this is optimal, as any single-sample algorithm for* UnsortedProportionalSampling *needs time* $\Omega(n)$.

The fundamental problem of the exact and efficient generation of random values from discrete and continuous distributions has been studied extensively in the literature. Knuth and Yao investigated in their seminal work [6] the power of several restricted devices, like finite-state machines; the articles [3, 14] provide a further refined treatment of the topic. However, their results are not directly comparable to ours, since they do not make any assumption on the sequence of probabilities, and use unbiased coin flips as the only source of randomness, but cannot guarantee efficient precomputation on general sequences. Furthermore, Hagerup, Mehlhorn and Munro [4] and Matias, Vitter and Ni [7] provided algorithms for a dynamic version of UnsortedProportionalSampling, where the probabilities may change over time. In particular, under certain mild conditions their results guarantee the same bounds as in Theorem 4.

The rest of the paper is structured as follows. In the following section we will show Theorem 4. Section 3 contains the proofs of Theorems 1 and 2, while Section 4 is devoted to the proof of Theorem 3. We discuss relaxations to our input model and possible extensions in Section 5.

2 Proportional Sampling on Unsorted Probabilities

In this section we consider UnsortedProportionalSampling and prove Theorem 4. The upper bound can be reached by the old technique known as *pairing* or *aliasing* [13]; see also Mihai Pătraşcu's blog [10] for a nice explanation. Basically, we use $\mathcal{O}(n)$ preprocessing to distribute the probabilities of all elements over n urns such that any urn contains probability mass of at most two elements. For querying we choose an urn uniformly at random and choose a random one of the two included elements according to their probability mass in the urn, which gives $\mathcal{O}(1)$ worst-case querying time.

The lower bound for Theorem 4 is provided by the following lemma, which reduces UnsortedProportionalSampling to searching in an unordered array. Moreover, the same proof yields the lower bound of Theorem 2 for UnsortedSubsetSampling.

Lemma 1. *Any single-sample algorithm for* UnsortedProportionalSampling *needs* $\Omega(n)$ *expected time. Moreover, any single-sample algorithm for* UnsortedSubsetSampling *needs* $\Omega(n)$ *expected time.*

Proof. Consider the instances $\mathbf{p}^{(k)} = (p_1^{(k)}, \ldots, p_n^{(k)})$ with $p_i^{(k)} = \delta_{ik}$, where δ_{ik} is the Kronecker delta. Any sampling algorithm for UnsortedProportionalSampling returns k on instance $\mathbf{p}^{(k)}$ with probability 1. This cannot be done better than with linear search for k, and randomness does not help, either. With varying μ, no better bound is possible, either: Simply set $p_i^{(k)} = \mu\delta_{ik}$.

Observe that on the same instance any sampling algorithm for UnsortedSubsetSampling returns $\{k\}$ with probability 1. This needs runtime $\Omega(n)$ for the same reasons. With varying μ, no better bound is possible, either: Set the first $s := \lceil \mu - 1 \rceil$ probabilities p_i to values that sum up to $\mu - 1$, and let $p_i^{(k)} = \delta_{ik}$ for $s < i \leqslant n$. Then we still need runtime $\Omega(n - \mu)$ for searching k. As we also need runtime $\Omega(\mu)$ for outputting the result, the claim follows. □

3 Subset Sampling

In this section we consider SortedSubsetSampling and UnsortedSubsetSampling and prove Theorems 1 and 2. An interesting interplay between both of these problems will be revealed on the way.

We begin with a first algorithm for unsorted probabilities that has a quite large preprocessing time, but will be used for a base case later. The algorithm uses Theorem 4, which we proved in the preceding section.

Lemma 2. UnsortedSubsetSampling *can be solved in* $\mathcal{O}(n^2)$ *preprocessing time and* $\mathcal{O}(1 + \mu)$ *expected query time.*

Proof. For $i \in [n]$ let X_i be the smallest sampled element which is at least i, or ∞, if no such element is sampled. X_i is a random variable with $\Pr[X_i = j] = p_j \prod_{i \leqslant k < j} (1 - p_k)$ and $\Pr[X_i = \infty] = \prod_{i \leqslant k \leqslant n} (1 - p_k)$. These probabilities can be computed in time $\mathcal{O}(n)$ for any i, i.e., in time $\mathcal{O}(n^2)$ for all i. After having computed the distribution of the X_i's, we execute, for each $i \in [n]$, the preprocessing of Theorem 4, see the beginning of Section 2, which allows us to quickly sample X_i later on. This preprocessing costs in total $\mathcal{O}(n^2)$.

For querying, we start at $i = 1$ and iteratively sample the smallest element $j \geqslant i$ (i.e., sample X_i), output j, and start over with $i = j + 1$. This is done until $j = \infty$ or $i = n + 1$. Note that any sample of X_i can be computed in $\mathcal{O}(1)$ time with our preprocessing, so that sampling $S \subseteq [n]$ will be done in time $\mathcal{O}(1 + |S|)$. The expected runtime is, thus, $\mathcal{O}(1 + \mu)$. □

After having this base case, we turn towards reductions between SortedSubsetSampling and UnsortedSubsetSampling. First, we give an algorithm for UnsortedSubsetSampling, that reduces the problem to SortedSubsetSampling. For this, we roughly sort the probabilities so that we get good upper bounds for each probability. Then these upper bounds will be a sorted instance. After querying from this sorted instance, we use rejection (see, e.g., [5]) to sample with the original probabilities.

Lemma 3. *Assume that* SortedSubsetSampling *can be solved in* $p(n, \mu)$ *preprocessing time and* $q(n, \mu)$ *expected query time, where* p *and* q *are monotonically increasing in* n *and* μ*. Then* UnsortedSubsetSampling *can be solved in* $\mathcal{O}(n + p(n, 2\mu + 1))$ *preprocessing time and* $\mathcal{O}(1 + \mu + q(n, 2\mu + 1))$ *expected query time.*

Proof. For preprocessing, we permute the input $\mathbf{p}$ so that it is approximately sorted, by putting it into buckets $B_k := \{i \in [n] \mid 2^{-k} \geqslant p_i \geqslant 2^{-k-1}\}$, for $k \in \{0, 1, \ldots, L\}$, and $B_L := \{i \in [n] \mid 2^{-L} \geqslant p_i\}$, where $L = \lceil \log n \rceil$. For each $i \in B_k$ we set $\overline{p}_i := 2^{-k}$, which is an upper bound on p_i. We sort the probabilities $\overline{p}_i$, $i \in [n]$, descendingly using bucket sort with the buckets B_k, yielding $\overline{p}'_1 \geqslant \ldots \geqslant \overline{p}'_n$. In this process we store the original index $\mathrm{ind}(i)$ corresponding to $\overline{p}'_i$, so that we can find $p_{\mathrm{ind}(i)}$ corresponding to $\overline{p}'_i$ in constant time. Then we run the preprocessing of SortedSubsetSampling on $\overline{p}'_1, \ldots, \overline{p}'_n$. Note that

$$\overline{\mu} := \sum_{i=1}^{n} \overline{p}'_i = \sum_{i=1}^{n} \overline{p}_i \leqslant \sum_{i=1}^{n} \max\left\{2p_i, \frac{1}{n}\right\} \leqslant 2\mu + 1.$$

For querying, we query $\overline{p}'_1, \ldots, \overline{p}'_n$ using SortedSubsetSampling, yielding $S' \subseteq [n]$. We compute $S := \{\mathrm{ind}(i) \mid i \in S'\}$. Each $i \in S$ was sampled with probability $\overline{p}_i \geqslant p_i$. We use rejection to get this probability down to p_i. For this, we generate for each $i \in S$ a random number rand() and check whether it is smaller than or equal to $\frac{p_i}{\overline{p}_i}$. If this is not the case, we delete i from S. Note that we have thus sampled i with probability p_i, and all elements are sampled independently, so that we can return S. □

We also give a reduction in the other direction, solving SortedSubsetSampling by UnsortedSubsetSampling.

Lemma 4. *Assume that* UnsortedSubsetSampling *can be solved in $p(n, \mu)$ preprocessing time and $q(n, \mu)$ expected query time, where p and q are monotonically increasing in n and μ. Then* SortedSubsetSampling *can be solved in $\mathcal{O}(\log n + p(1+\log n, 2\mu))$ preprocessing time and $\mathcal{O}(1 + \mu + q(1+\log n, 2\mu))$ expected query time.*

Proof. We consider blocks $B_k = \{i \in [n] \mid 2^k \leqslant i < 2^{k+1}\}$, with $k \in \{0, \ldots, L\}$ and $L := \lfloor \log n \rfloor$. For $i \in B_k$ we let $\overline{p}_i := p_{2^k}$, which is an upper bound on p_i. We will first sample with respect to the probabilities $\overline{p}_i$ - call the sampled elements *potential* - and then use rejection. For this, let X_k be an indicator random variable for the event that we sample *at least one* potential element in B_k. Then $q_k := \Pr[X_k = 1] = 1 - (1 - p_{2^k})^{|B_k|}$. Moreover, let Y_k be a random variable for the first potential element in block B_k minus 2^k. Let $Y_k = \infty$, if no element in B_k is sampled as a potential element. Then $\Pr[Y_k = i] = p_{2^k}(1-p_{2^k})^i$ for $i \in \{0, \ldots, |B_k| - 1\}$, and $\Pr[Y_k = \infty] = \Pr[X_k = 0] = 1 - q_k$. We calculate

$$\Pr[Y_k = i \mid X_k] = \frac{\Pr[Y_k = i]}{\Pr[X_k]} = \frac{p_{2^k}}{q_k}(1 - p_{2^k})^i.$$

Since this is a geometric distribution, we can sample from it in constant time as sketched in the introduction; see also [5].

Now, for preprocessing, we compute the probabilities q_k, which can be done in time $\mathcal{O}(\log n)$ (as $a^b = \exp(b \log a)$ can be computed in constant time on a Real RAM), and run the preprocessing of UnsortedSubsetSampling on them. Note that the q_k are in general unsorted.

For querying, we query the blocks B_k that contain potential elements using the query algorithm for UnsortedSubsetSampling. Then for each block B_k that contains a potential element, we sample all potential elements in this block. Note that the first of the potential elements in B_k is distributed as $\Pr[Y_k = i \mid X_k]$, which is geometric, so we can sample it in constant time, while all further potential elements are distributed as Y_k (but only on the remainder of the block), which is still geometric. After thus sampling potential elements $\overline{S}$, we reject each potential element with the right probability: We keep each $i \in \overline{S}$ only if $\text{rand}() \leqslant \frac{p_i}{\overline{p}_i}$. This yields a correctly distributed sample.

Let $\overline{\mu} := \sum_{i=1}^{n} \overline{p}_i$. The overall query time is at most $q(1+\log n, \overline{\mu}) + \mathcal{O}(1+|\overline{S}|)$ when sampling potential elements $\overline{S}$. As the expected value of $|\overline{S}|$ is $\overline{\mu}$, all we need to show in order to finish the proof is $\overline{\mu} \leqslant 2\mu$. For this, note that $\overline{p}_i \leqslant p_{\lceil i/2 \rceil}$. This yields

$$\overline{\mu} = \sum_{i=1}^{n} \overline{p}_i \leqslant \sum_{i=1}^{n} p_{\lceil i/2 \rceil} \leqslant 2 \sum_{i=1}^{n} p_i = 2\mu.$$

□

Next, we put above three lemmas together to prove the upper bounds of Theorems 1 and 2.

Proof (Theorem 2, upper bound). To solve UnsortedSubsetSampling, we use the reduction Lemma 3 and then Lemma 4, followed by the base case Lemma 2. This reduces the instance size from n to $\mathcal{O}(\log n)$, so that preprocessing costs $\mathcal{O}(n)$ for the invocation of the first lemma, $\mathcal{O}(\log n)$ for the second, and $\mathcal{O}(\log^2 n)$ for the third. Note that μ is increased by constant factors only, so that we indeed get the optimal query time $\mathcal{O}(1 + \mu)$. □

Proof (Theorem 1, upper bound). To solve SortedSubsetSampling, we use the reductions Lemma 4, Lemma 3, and Lemma 4 again, followed by the base case Lemma 2. This reduces the instance size from n to $\mathcal{O}(\log n)$ and further down to $\mathcal{O}(\log \log n)$, while μ is increased by constant factors only. For precomputation this yields a runtime of $\mathcal{O}(\log n)$ from Lemmas 4 and 3, $\mathcal{O}(\log \log n)$ from the second invocation of Lemma 4, and $\mathcal{O}(\log^2 \log n)$ from the base case Lemma 2, summing up to $\mathcal{O}(\log n)$. The query time is the optimal expected time $\mathcal{O}(1 + \mu)$. □

Note that we moved the proof of the lower bound of Theorem 1 for SortedSubsetSampling to the appendix due to space limitations.

4 Proportional Sampling on Sorted Probabilities

We prove Theorem 3 in this section, i.e., we show how to solve SortedProportionalSampling in $\mathcal{O}(\log n)$ preprocessing time and $\mathcal{O}(1)$ expected query time. We do this by first considering the special case of $\frac{1}{2} \leqslant \mu \leqslant 1$, so that we have a (nearly) proper probability distribution. Lemma 7 shows how to reduce SortedProportionalSampling to SortedSubsetSampling in this special case. Then we reduce the general case with arbitrary μ to the special case.

4.1 Special Case $1/2 \leqslant \mu \leqslant 1$

We first fix some notation for this section. Let $\mathbf{p}$ be an instance to SortedProportionalSampling with $\mu = \mu_{\mathbf{p}}$ in the range $[\frac{1}{2}, 1]$. Instead of $\mathbf{p}$ we consider $\mathbf{p}' = (p'_1, \ldots, p'_n)$ with $p'_i := \frac{p_i}{1+p_i}$. Note that $\mathbf{p}'$ ist still sorted and $\mu' := \sum_{i=1}^{n} p'_i$ is in the range $[\frac{\mu}{2}, \mu]$, thus in the range $[\frac{1}{4}, 1]$.

Let $Y = \text{SortedProportionalSampling}(\mathbf{p})$ be the random variable denoting proportional sampling on input $\mathbf{p}$, and $X = \text{SortedSubsetSampling}(\mathbf{p}')$ be the random variable denoting subset sampling on input $\mathbf{p}'$. Then conditioned on sampling exactly one element $X = \{i\}$, this element i is distributed exactly as Y, as formulated by the following lemma.

Lemma 5. *With the definitions and assumptions of this section we have for all* $i \in [n]$

$$\Pr[X = \{i\} \mid |X| = 1] = \Pr[Y = i].$$

Proof. Bayes' rule and straightforward calculation give

$$\begin{aligned}
\Pr[X = \{i\} \mid |X| = 1] &= \Pr[X = \{i\}] / \Pr[|X| = 1] \\
&= \left(\frac{p'_i}{1 - p'_i} \prod_{k=1}^{n} (1 - p'_k) \right) / \left(\sum_{j=1}^{n} \frac{p'_j}{1 - p'_j} \prod_{k=1}^{n} (1 - p'_k) \right) \\
&= \left(\frac{p'_i}{1 - p'_i} \right) / \left(\sum_{j=1}^{n} \frac{p'_j}{1 - p'_j} \right)
\end{aligned}$$

Plugging in the definition of p'_i yields

$$\Pr[X = \{i\} \mid |X| = 1] = p_i / \sum_{j=1}^{n} p_j = \Pr[Y = i].$$

□

Moreover, the probability of sampling exactly one element is large, as shown in the following lemma. Note that this bound is not best possible but sufficient for our purposes.

Lemma 6. *With the definitions and assumptions of this section we have*

$$\Pr[|X| = 1] \geqslant 1/8.$$

Proof. Clearly,

$$\Pr[|X| = 1] = \sum_{j=1}^{n} \frac{p'_j}{1 - p'_j} \prod_{k=1}^{n} (1 - p'_k).$$

Assume there is no p'_i greater than $1/2$. Then we have $1 - p'_i \geqslant 4^{-p'_i}$ for all $i \in [n]$, so we get

$$\Pr[|X| = 1] \geqslant \sum_{j=1}^{n} p'_j \cdot \prod_{k=1}^{n} 4^{-p'_k} = \mu' \cdot 4^{-\sum_{k=1}^{n} p'_k} = \mu' \cdot 4^{-\mu'} \geqslant \frac{1}{8}.$$

Otherwise, there is exactly one $p'_{i^*} > 1/2$, as $\mu' \leqslant 1$. Then $1 - p'_k \geqslant 4^{-p'_k}$ holds for all $k \in [n]$, $k \neq i^*$, which yields

$$\begin{aligned}\Pr[|X| = 1] \geqslant \Pr[X = \{i^*\}] = p'_{i^*} \prod_{\substack{1 \leqslant k \leqslant n \\ k \neq i^*}} (1 - p'_k) &\geqslant \frac{1}{2} \prod_{\substack{1 \leqslant k \leqslant n \\ k \neq i^*}} 4^{-p'_k} \\ &\geqslant \frac{1}{2} 4^{-\sum_{k=1}^{n} p'_k} = \frac{1}{2} 4^{-\mu'} \geqslant \frac{1}{8}.\end{aligned}$$

□

We put these facts together to show the following result.

Lemma 7. *Assume that* SortedSubsetSampling *can be solved in $p(n, \mu)$ preprocessing time and $q(n, \mu)$ expected query time, where p and q are monotonically increasing in n and μ. Then* SortedProportionalSampling *on instances with $\frac{1}{2} \leqslant \mu \leqslant 1$ can be solved in $\mathcal{O}(p(n, 1))$ preprocessing time and $\mathcal{O}(q(n, 1))$ expected query time.*

Proof. For preprocessing, given input $\mathbf{p}$, we run the preprocessing of SortedSubsetSampling on input $\mathbf{p}'$. This does not mean that we compute the vector $\mathbf{p}'$ beforehand, but if the preprocessing algorithm of SortedSubsetSampling reads the i-th input value, we compute $p'_i = \frac{p_i}{1+p_i}$ on the fly, so that this needs runtime $\mathcal{O}(p(n, 1))$. It allows to sample X later on in expected runtime $\mathcal{O}(q(n, 1))$ using the same trick of computing $\mathbf{p}'$ on the fly.

For querying, we repeatedly sample X until we sample a set S of size one. Returning the unique element of S results in a proper sample according to SortedProportionalSampling by Lemma 5. Moreover, by Lemma 6 and the fact that sampling X needs expected time $\mathcal{O}(q(n, 1))$ after our preprocessing, we need expected query time $\mathcal{O}(q(n, 1))$. □

4.2 General Case

Lemma 8. *Assume that* SortedProportionalSampling *on instances with $\frac{1}{2} \leqslant \mu \leqslant 1$ can be solved in $p(n)$ preprocessing time and $q(n)$ expected query time. Then* SortedProportionalSampling *(for general instances) can be solved in $\mathcal{O}(\log n + p(n))$ preprocessing time and $\mathcal{O}(q(n))$ expected query time.*

Proof. We need to compute a good upper bound $\overline{\mu} \geqslant \mu$. For this we reuse an idea of the proof of Lemma 4: For $i \in [n]$ let 2^k be the largest power of two less

than or equal to i, and set $\overline{p}_i := p_{2^k}$. Then $\overline{\mu} := \sum_{i=1}^n \overline{p}_i \geqslant \sum_{i=1}^n p_i = \mu$, and we have $\overline{p}_i \leqslant p_{\lceil i/2 \rceil}$, so that

$$\overline{\mu} = \sum_{i=1}^{n} \overline{p}_i \leqslant \sum_{i=1}^{n} p_{\lceil i/2 \rceil} \leqslant 2 \sum_{i=1}^{n} p_i = 2\mu.$$

Hence, $\overline{\mu}$ is indeed a good upper bound on μ. Moreover, $\overline{\mu}$ can be computed in time $\mathcal{O}(\log n)$, as

$$\overline{\mu} = \sum_{k=0}^{\lfloor \log n \rfloor} p_{2^k}(\min\{2^{k+1} - 1, n\} - 2^k + 1).$$

Now, for preprocessing, we compute $\overline{\mu}$ and consider $\mathbf{p}' = (p'_1, \ldots, p'_n)$ with $p'_i := \frac{p_i}{\overline{\mu}}$. Since $\overline{\mu} \geqslant \mu \geqslant \frac{\overline{\mu}}{2}$ we have $\mu' := \sum_{i=1}^n p'_i$ in the range $[\frac{1}{2}, 1]$. Thus, we can run the preprocessing of SortedProportionalSampling (on instances with bounded μ) on $\mathbf{p}'$. We do this without computing the whole vector $\mathbf{p}'$. Instead, if the preprocessing algorithm reads the i-th input value, we compute $p'_i = \frac{p_i}{\mu}$ on the fly. This way we need a runtime of $\mathcal{O}(\log n + p(n))$.

For querying, we query according to $\mathbf{p}'$ within expected runtime $\mathcal{O}(q(n))$, where we again compute values of $\mathbf{p}'$ on the fly as needed. As we want to sample proportional to the input probabilities, a sample with respect to $\mathbf{p}'$ has the same distribution as a sample with respect to $\mathbf{p}$, so that we simply return the sample we have. □

Proof (Theorem 3). To solve SortedProportionalSampling we take Lemmas 8 and 7 and Theorem 1 together. □

5 Relaxations

In this section we describe some natural relaxations for the input model studied so far in this paper.

Large Deviations for the Running Times The query runtimes in Theorems 1, 2 and 3 are, in fact, not only small in expectation, but they are also concentrated, i.e., they satisfy large deviation estimates in the following sense. Let t be the expected runtime bound and T the actual runtime. Then

$$\Pr[T > kt] = e^{-\Omega(k)},$$

where the asymptotics are with respect to k. This is shown rather straightforwardly along the lines of our proofs of these theorems. The fundamental reason for this is that the size of the random set X is concentrated. Indeed, let X_i be an indicator random variable for the i-th element as above. Then for any $a > 1$ we obtain along the lines of the proof of the Chernoff bound

$$\Pr[|S| > k(\mu + 1)] = \Pr[a^{\sum_{i=1}^n X_i} > a^{k(\mu+1)}] \leqslant \frac{\mathbb{E}[a^{\sum_{i=1}^n X_i}]}{a^{k(\mu+1)}}.$$

Then, the independence of the X_i's implies that

$$\begin{aligned}\Pr[|S| > k(\mu+1)] &\leqslant \frac{\prod_{i=1}^{n} \mathbb{E}[a^{X_i}]}{a^{k(\mu+1)}} \\ &= \frac{\prod_{i=1}^{n}(ap_i + (1-p_i))}{a^{k(\mu+1)}} \leqslant \exp((a-1)\mu - k(\mu+1)\ln a).\end{aligned}$$

Setting $a = k+1$ yields

$$\Pr[|S| > k(\mu+1)] \leqslant \exp(k\mu - k(\mu+1)\log(k+1)) \leqslant (k+1)^{-k},$$

for $k \geqslant 2$, as claimed.

Partially Sorted Input The condition of sorted input for SortedSubsetSampling and SortedProportionalSampling can easily be relaxed, as long as we have sorted upper bounds of the probabilities. Given input $\mathbf{p}$ and sorted $\overline{\mathbf{p}}$ with $p_i \leqslant \overline{p}_i$ for all $i \in [n]$, we simply sample according to $\overline{\mathbf{p}}$ and use rejection to get down to the probabilities $\mathbf{p}$. This allows for the optimal query time $\mathcal{O}(1+\mu)$ as long as $\overline{\mu} = \sum_{i=1}^{n} \overline{p}_i = \mathcal{O}(1+\mu)$, where $\mu = \sum_{i=1}^{n} p_i$.

Unimodular Input Many natural distributions $\mathbf{p}$ are not sorted, but unimodular, meaning that p_i is monotonically increasing for $1 \leqslant i \leqslant m$ and monotonically decreasing for $m \leqslant i \leqslant n$ (or the other way round). Knowing m, we can run the algorithms developed in this paper on both sorted halfs, and combine the return values, which gives an optimal query algorithm for unimodular inputs. Alternatively, if we have strong monotonicity, we can search for m in time $\mathcal{O}(\log n)$ using ternary search, which does not increase our precomputation time.

This can be naturally generalized to k-modular inputs, where the monotonicity changes k times.

Approximate Input In some applications it may be costly to compute the probabilities p_i exactly, but we are able to compute approximations $\overline{p}_i(\varepsilon) \geqslant p_i \geqslant \underline{p}_i(\varepsilon)$, with relative error at most ε, where the cost of computing these approximations depends on ε. We can still guarantee optimal query time, if the costs of computing these approximations are small enough, see e.g. [8].

Indeed, we can surely sample a superset $\overline{S}$ with respect to the probabilities $\overline{p}_i(1)$. Then we want to use rejection, i.e., for each element $i \in \overline{S}$ we want to compute a random number $r := \text{rand}()$ and delete i from $\overline{S}$ if $r \cdot \overline{p}_i(1) > p_i$, to get a sample set S. This check can be performed as follows. We initialize $k := 1$. If $r \cdot \overline{p}_i(1) > \overline{p}_i(2^{-k})$ we delete i from $\overline{S}$. If $r \cdot \overline{p}_i(1) \leqslant \underline{p}_i(2^{-k})$ we keep i and are done. Otherwise, we increase k by 1. This method needs an expected number of $\mathcal{O}(1)$ rounds of increasing k; the probability of needing k rounds is $\mathcal{O}(2^{-k})$. Hence, if the cost of computing $\overline{p}_i(\varepsilon)$ and $\underline{p}_i(\varepsilon)$ is $\mathcal{O}(\varepsilon^{-c})$ with $c < 1$, the expected overall cost is constant, and we get an optimal expected query time of $\mathcal{O}(1+\mu)$.

Bibliography

[1] Borodin, A., Munro, I.: The computational complexity of algebraic and numeric problems. American Elsevier Publishing Co., Inc., New York-London-Amsterdam (1975)

[2] Devroye, L.: Nonuniform random variate generation. Springer-Verlag, New York (1986)

[3] Flajolet, P., Saheb, N.: The complexity of generating an exponentially distributed variate. Journal of Algorithms 7(4), 463–488 (1986)

[4] Hagerup, T., Mehlhorn, K., Munro, J.I.: Maintaining discrete probability distributions optimally. In: 20th International Colloquium on Automata, Languages and Programming (ICALP '93). pp. 253–264 (1993)

[5] Knuth, D.E.: The Art of Computer Programming. Vol. 2: Seminumerical Algorithms. Addison-Wesley Publishing Co., Reading, Mass., third edn. (2009)

[6] Knuth, D.E., Yao, A.C.: The complexity of nonuniform random number generation. In: Algorithms and complexity (Proc. Sympos., Carnegie-Mellon Univ., Pittsburgh, Pa., 1976), pp. 357–428 (1976)

[7] Matias, Y., Vitter, J.S., Ni, W.C.: Dynamic generation of discrete random variates. Theory of Computing Systems 36(4), 329–358 (2003)

[8] Nacu, Ş., Peres, Y.: Fast simulation of new coins from old. The Annals of Applied Probability 15(1A), 93–115 (2005)

[9] Preparata, F.P., Shamos, M.I.: Computational Geometry. Texts and Monographs in Computer Science, Springer-Verlag, New York (1985)

[10] Pătraşcu, M.: Webdiarios de motocicleta, sampling a discrete distribution (2011), infoweekly.blogspot.com/2011/09/sampling-discrete-distribution.html

[11] Tsai, M.T., Wang, D.W., Liau, C.J., Hsu, T.S.: Heterogeneous subset sampling. In: 16th Annual International Conference on Computing and Combinatorics (COCOON '10). pp. 500–509 (2010)

[12] Tsai, M.T., Wang, D.W., Liau, C.J., Hsu, T.S.: Heterogeneous subset sampling (2012), submitted for publication

[13] Walker, A.J.: New fast method for generating discrete random numbers with arbitrary distributions. Electronic Letters 10, 127–128 (1974)

[14] Yao, A.C.: Context-free grammars and random number generation. In: Combinatorial algorithms on words (Maratea, 1984), vol. 12, pp. 357–361. Springer (1985)

Appendix

Here we present the proof of the lower bound of Theorem 1 for SortedSubset-Sampling.

Proof (Theorem 1, lower bound). Let (P, Q) be a preprocessing and a query algorithm, and let $\mathbf{p}$ be an instance. Let $D = P(\mathbf{p})$ be the result of the precomputation. By definition we have for any $S \subseteq [n]$

$$\Pr[Q(\mathbf{p}, D) = S] = \left(\prod_{i \in S} p_i\right)\left(\prod_{i \in [n]\setminus S} (1 - p_i)\right) =: P_{\mathbf{p}}(S),$$

meaning that we sample a set with the right probability independent from the random choices in the preprocessing.

Let $C \subseteq [n]$ be the positions $i \in [n]$ where the preprocessing read the value p_i during the computation of D. Without loss of generality, we can assume that $1, n \in C$, i.e., that the preprocessing read p_1 and p_n, as this adjustment of the algorithm does not increase its runtime asymptotically. Furthermore, without loss of generality, we can assume that the query algorithm reads all positions i in its return set $S = Q(\mathbf{p}, D)$.

Now, for instance $\mathbf{p}$ and $S \subseteq B \subseteq [n]$, let $P_{\mathbf{p}}(B, S)$ be the probability that algorithm $Q(\mathbf{p}, D)$ reads exactly the values p_i with $i \in B$ and returns the set S. We clearly have

$$\sum_{B \supseteq S} P_{\mathbf{p}}(B, S) = P_{\mathbf{p}}(S). \tag{1}$$

Furthermore, if we assume an expected query runtime of at most $t = t(n, \mu_{\mathbf{p}})$, then there is a set $S^* \subseteq [n]$ with

$$\sum_{\substack{B \supseteq S^* \\ |B| \leqslant 2t}} P_{\mathbf{p}}(B, S^*) \geqslant \frac{1}{2} P_{\mathbf{p}}(S^*), \tag{2}$$

since otherwise

$$\Pr[Q(\mathbf{p}, D) \text{ runs for time} \leqslant 2t] \leqslant \sum_{\substack{B, S \subseteq [n] \\ B \supseteq S \\ |B| \leqslant 2t}} P_{\mathbf{p}}(B, S) < \frac{1}{2} \sum_{S \subseteq [n]} P_{\mathbf{p}}(S) = \frac{1}{2},$$

in contrast to Markov's inequality. Here we used that $|B|$ is a lower bound on the runtime of $Q(\mathbf{p}, D)$.

From (2) we infer, using the maximum-arithmetic mean inequality, that there is a set $B^* \supseteq S^*$ with

$$P_{\mathbf{p}}(B^*, S^*) \geqslant \frac{1}{2\binom{n}{2t}} P_{\mathbf{p}}(S^*). \tag{3}$$

Now we fix the instance $\mathbf{p} = (p_1, \dots, p_n)$ by setting

$$p_i := \frac{\alpha}{i},$$

for a parameter $0 < \alpha \leqslant 1/2$ to be chosen later. Fixing sets S^*, B^* as above for this instance $\mathbf{p}$, we define a second instance $\mathbf{p}' = (p'_1, \dots, p'_n)$ by setting

$$p'_i := \min\{p_j \mid i \geqslant j \in B^* \cup C\}.$$

This means that $\mathbf{p}$ and $\mathbf{p}'$ agree on the read positions B^* and C, and at all other positions p'_i is as large as possible with $\mathbf{p}'$ still being sorted. This means that the prepocessing and the query algorithm cannot distinguish between both instances, implying a critical property we will use,

$$P_{\mathbf{p}'}(B^*, S^*) = P_{\mathbf{p}}(B^*, S^*).$$

With this, we get

$$P_{\mathbf{p}'}(S^*) \overset{(1)}{\geqslant} P_{\mathbf{p}'}(B^*, S^*) = P_{\mathbf{p}}(B^*, S^*) \overset{(3)}{\geqslant} \frac{1}{2\binom{n}{2t}} P_{\mathbf{p}}(S^*). \tag{4}$$

We next bound $P_{\mathbf{p}}(S^*)$ and $P_{\mathbf{p}'}(S^*)$. For the former we get

$$\begin{aligned} P_{\mathbf{p}}(S^*) &= \left(\prod_{i \in S^*} p_i\right)\left(\prod_{i \in [n] \setminus S^*} (1 - p_i)\right) \\ &= Q \prod_{i=1}^{n} (1 - p_i), \end{aligned}$$

where $Q := \prod_{i \in S^*} \frac{p_i}{1 - p_i}$. Since $p_i \leqslant \alpha \leqslant 1/2$ we have $1 - p_i \geqslant 4^{-p_i}$ for all $i \in [n]$, so we get

$$P_{\mathbf{p}}(S^*) \geqslant Q \cdot 4^{-\sum_{i=1}^{n} p_i} \geqslant Q \cdot 4^{-\alpha(1 + \ln n)} \geqslant Q \cdot 2^{-(1 + \ln n)}.$$

Since $\ln n < \log n$ (and even $\log n - \ln n \to \infty$ for $n \to \infty$) we get for large enough n

$$P_{\mathbf{p}}(S^*) \geqslant \frac{2Q}{n}. \tag{5}$$

Let $B^* \cup C = \{i_1, \dots, i_k\}$ with $i_1 \leqslant \dots \leqslant i_k$. By assumption, we have $i_1 = 1$, $i_k = n$, and we define $i_{k+1} := n + 1$. For $P_{\mathbf{p}'}(S^*)$ we now get

$$\begin{aligned} P_{\mathbf{p}'}(S^*) &= \left(\prod_{i \in S^*} p'_i\right)\left(\prod_{i \in [n] \setminus S^*} (1 - p'_i)\right) \\ &= \left(\prod_{i \in S^*} p_i\right)\left(\prod_{i \in (B^* \cup C) \setminus S^*} (1 - p_i)\right)\left(\prod_{\ell=1}^{k} (1 - p_{i_\ell})^{i_{\ell+1} - i_\ell - 1}\right) \\ &= Q \prod_{\ell=1}^{k} (1 - p_{i_\ell})^{i_{\ell+1} - i_\ell}. \end{aligned}$$

Using $1 - x \leqslant e^{-x}$ for $x \geqslant 0$ this yields

$$P_{\mathbf{p}'}(S^*) \leqslant Q \cdot \exp\left(-\sum_{\ell=1}^{k} p_{i_\ell}(i_{\ell+1} - i_\ell)\right) = Q \cdot \exp\left(-\alpha \sum_{\ell=1}^{k} \left(\frac{i_{\ell+1}}{i_\ell} - 1\right)\right).$$

Now we use the arithmetic-geometric mean inequality, yielding

$$\frac{1}{k}\sum_{\ell=1}^{k} \frac{i_{\ell+1}}{i_\ell} \geqslant \left(\prod_{\ell=1}^{k} \frac{i_{\ell+1}}{i_\ell}\right)^{1/k} \geqslant n^{1/k},$$

so that we get

$$P_{\mathbf{p}'}(S^*) \leqslant Q \cdot \exp\left(-\alpha k(n^{1/k} - 1)\right).$$

Combining this with (4) and (5), and dividing by Q we get

$$\exp\left(-k\alpha(n^{1/k} - 1)\right) \geqslant \frac{1}{n\binom{n}{2t}} \geqslant n^{-(2t+1)}.$$

Taking the logarithm on both sides yields

$$k\alpha(n^{1/k} - 1) \leqslant (2t+1)\ln n,$$

which is in turn equivalent to

$$k \geqslant \frac{\log n}{\log(1 + (2t+1)\ln n/(k\alpha))}.$$

As $k \geqslant 1$, this implies

$$k \geqslant \frac{\log n}{\log(1 + (2t+1)\ln n/\alpha)}.$$

Assuming optimal expected query time $t = \mathcal{O}(1+\mu)$, and since $\mu = \Theta(\alpha \log n)$ we get

$$k = \Omega\left(\frac{\log n}{\log((1 + 1/\mu)\log n)}\right).$$

For $\mu = \mu(n) \geqslant \frac{1}{\log^{\mathcal{O}(1)} n}$ this yields, using that $k = B^* \cup C$ is a lower bound on the sum of preprocessing and query time, that preprocessing and querying together need running time at least

$$\Omega\left(\frac{\log n}{\log\log n}\right).$$

□