Succinct Sampling from Discrete Distributions

Karl Bringmann*
Max Planck Institute for Informatics
Campus E1.4
66123 Saarbrücken, Germany
kbringma@mpi-inf.mpg.de

Kasper Green Larsen†
MADALGO‡, Department of Computer Science
Aarhus University
Aarhus, Denmark
larsen@cs.au.dk

ABSTRACT

We revisit the classic problem of sampling from a discrete distribution: Given n non-negative w-bit integers $x_1, \ldots, x_n$, the task is to build a data structure that allows sampling i with probability proportional to x_i. The classic solution is Walker's alias method that takes, when implemented on a Word RAM, $\mathcal{O}(n)$ preprocessing time, $\mathcal{O}(1)$ expected query time for one sample, and $n(w+2\lg n+o(1))$ bits of space. Using the terminology of succinct data structures, this solution has redundancy $2n\lg n + o(n)$ bits, i.e., it uses $2n\lg n + o(n)$ bits in addition to the information theoretic minimum required for storing the input. In this paper, we study whether this space usage can be improved.

In the systematic case, in which the input is read-only, we present a novel data structure using $r + \mathcal{O}(w)$ redundant bits, $\mathcal{O}(n/r)$ expected query time and $\mathcal{O}(n)$ preprocessing time for any r. This is an improvement in redundancy by a factor of $\Omega(\lg n)$ over the alias method for $r = n$, even though the alias method is not systematic. Moreover, we complement our data structure with a lower bound showing that this trade-off is tight for systematic data structures.

In the non-systematic case, in which the input numbers may be represented in more clever ways than just storing them one-by-one, we demonstrate a very surprising separation from the systematic case: With only 1 redundant bit, it is possible to support optimal $\mathcal{O}(1)$ expected query time and $\mathcal{O}(n)$ preprocessing time!

On the one hand, our results improve upon the space requirement of the classic solution for a fundamental sampling problem, on the other hand, they provide the strongest known separation between the systematic and non-systematic case for any data structure problem. Finally, we also believe our upper bounds are practically efficient and simpler than Walker's alias method.

‡Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

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

†Kasper Green Larsen is a recipient of the *Google Europe Fellowship in Search and Information Retrieval*, and this research is supported in part by this Google Fellowship.

Categories and Subject Descriptors

E.1 [**Data**]: Data Structures

General Terms

Theory, Algorithms

1. INTRODUCTION

We revisit the classic problem of sampling from a discrete distribution: The input consists of non-negative numbers $x_1, \ldots, x_n$, and we want to build a data structure that supports the operation DRAW, which returns $i \in \{1, \ldots, n\}$ with probability $p_i = \frac{x_i}{\sum_j x_j}$. Multiple DRAW queries shall be independent. This problem has a classic solution by Walker [20], with preprocessing time improved by Kronmal and Peterson [14]; see [18] for an excellent explanation. The improved version of Walker's alias method needs $\mathcal{O}(n)$ preprocessing time, after which a DRAW query can be answered in $\mathcal{O}(1)$ worst case time. While the query time bound is clearly optimal, it has been noted in [1] that this has optimal preprocessing time, too.

We focus on the question of whether Walker's alias method has optimal space usage, or whether there are data structures for sampling from a discrete distribution that use less space. Unfortunately, as most sampling algorithms and data structures, the alias method is usually analyzed on the Real RAM model, where a memory cell may store an arbitrary real number, and any reasonable operation on two reals can be performed in constant time. In this model, an arbitrary amount of information can be stored in each memory cell. Hence, this model is not suited for analyzing space usage. In practice, where one usually implements algorithms and data structures analyzed on the Real RAM using double precision approximations, the space usage of the alias method would be $\mathcal{O}(n\tilde{w})$, where $\tilde{w}$ is the bit length of a double precision floating point number. More precisely, it uses n integers and n doubles. However, an algorithm or data structure that is exact on the Real RAM usually incurs some error when run on a physical computer, so that an analysis on a bounded precision machine model would be preferable to increase practicality.

We suggest to use the standard Word RAM model for this purpose, augmented by functionality for generating random numbers. In this model, a memory cell stores a w-bit integer (called word) and usual operations on two words can be performed in constant time. This includes bit level operations, such as $\wedge$, $\vee$, $\neg$, and arithmetic operations, like $+$, $\cdot$, and integer division. For sampling we need an additional operation RAND that produces a random word in constant time,[1] i.e., we assume to be able to draw w random bits in constant time. Thus, we can only draw a random number from a range $\{1, \dots, 2^\ell\}$, $\ell = \mathcal{O}(w)$, in constant worst-case time, i.e., from a range with size a power of 2. For all other ranges $\{1, \dots, k\}$, $k \in \mathbb{N}$, we can still uniformly sample from it in $\mathcal{O}(1)$ *expected* time when $k \leqslant 2^{\mathcal{O}(w)}$, e.g., by sampling in the range $\{1, \dots, 2^\ell\}$, where $2^\ell \geqslant k$ is the next power of 2, and rejecting as long as the sampled number does not lie in the goal range $\{1, \dots, k\}$. We make the usual assumption that $w = \Omega(\lg n)$, to be able to store pointers to all input elements.

Our sampling problem is the following on the Word RAM: The input consists of non-negative integers $x_1, \dots, x_n$, each of w bits. Build a data structure that supports an operation DRAW, which returns $i \in \{1, \dots, n\}$ with probability $p_i = x_i/S$, where $S := \sum_j x_j$. All invocations of DRAW shall be independent.

Is Walker's alias method efficient on the Word RAM? The biggest potential obstacle to that would be the computation with numbers of too long bit length. However, closely looking at the Real RAM version of the data structure one can see that all produced numbers are either integers less than n or rationals with denominator nS (more precisely, $\mathrm{lcm}(n, S)$), so they fit in $\mathcal{O}(1)$ words and can be processed in constant time. Thus, the data structure can easily be adapted and has preprocessing time $\mathcal{O}(n)$. There is one drawback: We need to draw uniform random numbers in $\{1, \dots, n\}$ and in $\{1, \dots, nS\}$, which can only be done in $\mathcal{O}(1)$ *expected* time. Thus, Walker's alias method degenerates to $\mathcal{O}(1)$ *expected* query time, which is theoretically unappealing, but makes not much difference for practice. In any case, no guarantee on worst-case query time is possible for our Word RAM model, as any probability we can generate in a bounded number of steps has as denominator a power of 2, but n and S are not bound to be powers of 2.

Working this out, on the Word RAM Walker's alias method needs $\Theta(n)$ preprocessing time, $\Theta(1)$ expected query time, and $n(w+2\lg n+o(1))$ bits of space. Thus the data structure has a space overhead of more than $2n \lg n$ bits compared to just storing the input numbers. Using the terminology from the world of succinct data structures, we say that the data structure has *redundancy* $2n\lg n+o(n)$ bits. Note that these are roughly the same requirements as for the Real RAM version of the data structure used with double precision approximations, only that now this is an exact data structure on a bounded precision machine.

[1]If we instead can only generate a random bit in constant time, then we can clearly simulate RAND in time $\mathcal{O}(w)$. However, even for uniform sampling we need $\Omega(\lg n)$ random bits, so that we cannot hope for a better query time than $\Theta(\lg n)$ in this case.

It is now a well-defined question to ask whether the space usage of Walker's alias method is optimal: *On the Word RAM, is there a data structure for sampling from a discrete distribution with redundancy less than* $2n\lg n+o(n)$ *bits?* Of course, such a data structure should use the optimal $\mathcal{O}(n)$ preprocessing time and $\mathcal{O}(1)$ expected query time, if possible.

In this paper we answer this question in two ways, in both cases improving upon Walker's data structure. First, we consider the *systematic case*, in which the input is read-only and also available at query time. This is a reasonable model if the input numbers may not be overwritten by the data structure, or if the input numbers are available only implicitly, i.e., we can afford to recompute each x_i when needed, but we cannot afford to store each x_i explicitly. In this case we present a data structure that uses $\mathcal{O}(n + w)$ *redundant bits*. In fact, Walker's classic data structure is *not* systematic, so that all $n(w + 2\lg n + o(1))$ bits stored in his solution are redundant, i.e., we improve by a factor of $\Theta(w) = \Omega(\lg n)$ over the alias method. We then generalize this result to further reduce the redundancy, at the cost of increasing the query time, yielding the following trade-off.

THEOREM 1.1. *For any* $1 \leqslant r \leqslant n$ *we can build a systematic data structure for sampling from a discrete distribution having* $r + \mathcal{O}(w)$ *bits of redundancy,* $\mathcal{O}(n/r)$ *expected query time, and* $\mathcal{O}(n)$ *preprocessing time.*

The question arises of whether one could save more than a factor of $\Theta(w)$ while still having $\mathcal{O}(1)$ expected query time, or, more generally, whether one can reduce the product rt further, where r is the redundancy and t the expected query time. With the following theorem we prove that this is impossible, showing optimality of the trade-off between redundancy and query time in Theorem 1.1.

THEOREM 1.2. *Consider any systematic data structure for sampling from a discrete distribution, having redundancy* r *and supporting* DRAW *in expected query time* t*. Then* $r \cdot t = \Omega(n)$.

This shows that Theorem 1.1 is asymptotically optimal with respect to all three aspects: space usage, query time, and preprocessing time (for the latter see [1], this proof also works in the Word RAM model).

So far we considered the systematic case, in which the input is read-only and always available. In the *non-systematic case*, on the other hand, the preprocessing is given access to the input, but the query algorithm is not. Thus, the preprocessing has to encode the input in some way in the data structure it outputs (possibly just storing the input without modifications). It is not immediately clear that such a data structure even needs to use nw bits of space, since two different inputs $x_1, \dots, x_n$ and $\hat{x}_1, \dots, \hat{x}_n$ represent the same distribution if there exists some $\alpha > 0$ such that $x_i = \alpha \hat{x}_i$ for all i. In fact, answering *range minimum queries* in an array of n ordered elements (given two indices i and j, return the index of the minimum element in the subarray from index

i through j) can be done using only $\mathcal{O}(n)$ bits, although one might first expect that $\Omega(n \lg n)$ bits are necessary, see e.g. [5]. However, we prove in Section 5 that such savings are not possible for sampling. More specifically, we prove the following result:

THEOREM 1.3. *Any non-systematic data structure for sampling from a discrete distribution must use at least nw bits of space for any $1 \leqslant w = o(n)$ and sufficiently large n.*

Thus, in contrast to range minimum queries, it is not possible to save even a single bit. As mentioned, Walker's data structure is non-systematic and has space usage $n(w + 2\lg n + o(1))$ bits, i.e., redundancy $2n \lg n + o(n)$ when analyzed as a non-systematic data structure. Very surprisingly, we show that it is possible to vastly improve over this bound in the non-systematic case. More precisely, we show that we need only 1 redundant bit to achieve optimal query time and preprocessing time!

THEOREM 1.4. *In the non-systematic case we can build a data structure for sampling from a discrete distribution that needs $nw + 1$ bits of space, $\mathcal{O}(1)$ expected query time, and $\mathcal{O}(n)$ preprocessing time.*

This is an astonishing result since it is the strongest obtained separation between the systematic and non-systematic case for *any* data structure problem: For redundancy r and expected query time t the optimal bound is $r \cdot t = \Theta(n)$ in the systematic case, while we have $r \cdot t = \mathcal{O}(1)$ in the non-systematic case. The largest previous separation was obtained for RANK and SELECT queries, where any systematic data structure must satisfy $r = \Omega((n/t) \lg t)$ [8], while there exist non-systematic data structures achieving $r = \Theta(n/(\lg n/t)^t) + \tilde{\mathcal{O}}(n^{3/4})$ [16].

Finally, we believe that our data structures are not only interesting from a theoretical point of view, but may also be of practical use. In fact, our systematic solution with $\mathcal{O}(1)$ query time is simpler than Walker's alias method, while our non-systematic solution with just 1 redundant bit is only slightly more involved. Furthermore, the constants hidden in the $\mathcal{O}$-notations are all small.

1.1 Related Work

Sampling. The majority of the literature on sampling uses the Real RAM model (see, e.g., [3]). Such algorithms and data structures are, in general, not exact on bounded precision machines and cannot be analyzed with respect to space usage.

In a seminal work Knuth and Yao [13] initiated the study of the sampling power of various restricted devices, like finite-state machines. They devise algorithms trying to minimize the use of random bits. However, they do not guarantee efficient precomputation on general sequences of probabilities, so that their results are incomparable to ours. These ideas have been further developed in [6, 7, 21].

Moreover, there are articles examining generalizations of the problem of sampling from a discrete distribution: The dynamic version of the problem, where the input numbers $x_1, \ldots, x_n$ may change over time, has been investigated in [10, 15]. In another direction, the special case of sorted inputs $x_1 \geqslant \ldots \geqslant x_n$ can be solved with a reduced preprocessing time of $\mathcal{O}(\lg n)$, as has been shown in [1]. The same paper also presents a generalization to sampling subsets. All of these papers only achieve $\mathcal{O}(1)$ *expected* sampling time, even on the Real RAM model, in contrast to Walker's alias method.

Succinct Data Structures. In the field of succinct data structures, the focus is on designing data structures that have space requirements as close to the information theoretic minimum as possible, while still answering queries efficiently. Here the space usage of a data structure is measured in the additive number of redundant bits used compared to the information theoretic minimum. As mentioned, previous work has focused on two types of data structures called systematic and non-systematic. Some of the most basic problems in the field include range minimum queries, RANK and SELECT. The systematic case is well understood for all three problems, with tight bounds for RANK and SELECT dating back to Raman et al. [19] and Golynski [8]. For constant query time, the redundancy needed for these two problems is $\Theta(n \lg \lg n / \lg n)$. For range minimum, Brodal et al. [2] proved that any systematic data structure with redundancy r must have worst case query time $t = \Omega(n/r)$. They complemented this lower bound with a data structure matching the entire trade-off curve.

The strongest separation between the systematic and non-systematic case had been limited, until somewhat recently, to a mere $\lg n$ factor in the redundancy for constant query time data structures, see e.g. [9]. In fact, it had been generally believed that a stronger separation would not be possible for problems such as RANK and SELECT. This belief was disproved in the seminal paper of Pătraşcu [16]. Here Pătraşcu demonstrated an exponential separation between the two cases by obtaining non-systematic RANK and SELECT data structures with redundancy $r = \Theta(n/(\lg n/t)^t) + \tilde{\mathcal{O}}(n^{3/4})$. Observe that the redundancy goes down exponentially fast with t and that redundancy $\mathcal{O}(n/\lg^c n)$ is possible in constant query time for any constant $c > 0$. Reducing the redundancy all the way to a constant while maintaining constant query time, as we do for our problem, was, however, proved impossible by Pătraşcu and Viola [17]. More specifically, they proved a redundancy lower bound of $r \geqslant n/(\lg n)^{\mathcal{O}(t)}$, thus almost matching the upper bound of Pătraşcu, except when $t \geqslant \lg^{1-o(1)} n$.

Finally, we mention an interesting problem for which extremely low redundancy and constant query time has been achieved before this work: The input to this problem consists of an array of n trits, i.e., numbers in $\{0, 1, 2\}$, and the goal is to represent the array in as close to $\lceil n \lg 3 \rceil$ bits as possible, such that each entry can be retrieved efficiently. For this problem, Dodis et al. [4] showed that constant query time can be achieved with a constant number of input dependent redundant bits plus $\mathcal{O}(\lg^2 n)$ precomputed bits depending only on n and the word size. From a separation point of view, this problem is, however, not interesting, as the problem makes no sense in the systematic setting.

1.2 Outline

In Section 2, we present our systematic data structures. In Section 3, we then demonstrate that it is possible to do much better in the non-systematic case. In Section 4, we complement our systematic data structures with a matching lower bound. Finally, in Section 5, we prove a lower bound on the amount of bits needed to represent an input distribution.

2. SAMPLING WITH READ-ONLY INPUT

In this section, we present a data structure that supports sampling from a discrete distribution if the input is read-only, i.e., in the systematic case. We achieve any desired redundancy of $r+\mathcal{O}(w)$ bits with expected query time $\mathcal{O}(n/r)$ and optimal preprocessing time $\mathcal{O}(n)$.

First, in the next section, we present a novel and practical data structure that uses $\mathcal{O}(n\lg n+w)$ redundant bits, $\mathcal{O}(1)$ expected query time, and $\mathcal{O}(n)$ preprocessing time. Then, in Section 2.2, we modify it to use only $\mathcal{O}(n+w)$ redundant bits. In Section 2.3, we show how to get any smaller redundancy while increasing query time.

2.1 Redundancy of $\mathcal{O}(n\lg n+w)$

Preprocessing. First, we compute $S=\sum_i x_i$ and store it using $\mathcal{O}(w)$ bits. Additionally, we store a sorted array A that contains numbers in $[n]=\{1,\dots,n\}$, namely, A contains, for each index i, the number i exactly $\lfloor nx_i/S\rfloor+1$ times. This finishes space usage.

Observe that A has size at most $2n$, since we have

$$|A|=\sum_i\left(\left\lfloor\frac{nx_i}{S}\right\rfloor+1\right)\leqslant\sum_i\left(\frac{nx_i}{S}+1\right)=\frac{nS}{S}+n=2n.$$

Moreover, A has entries in $[n]$, so that we can store it using $\mathcal{O}(n\lg n)$ bits. Note that A can easily be constructed in time $\mathcal{O}(n)$.

Sampling. Intuitively, if we return the value $A[k]$ for a uniform random $k\in\{1,\dots,|A|\}$, then this is close to sampling from the input distribution. We can make this into an exact sampling method with a slight modification, using the rejection method (see [3]) as follows.

1. Pick a uniformly random $k\in\{1,\dots,|A|\}$.
2. Rejection: If $k=1$ or $A[k-1]\neq A[k]$ then with probability $1-\mathrm{frac}(nx_{A[k]}/S)$ goto step 1.
3. Return $A[k]$.

Here, $\mathrm{frac}(x)=x-\lfloor x\rfloor$ is the fractional part of x. Note that in step 2 we check whether k is the first occurrence of $A[k]$ in A. If so, with some probability we throw away k and go to step 1 again, i.e., there is an implicit loop.

Let $i\in[n]$. What is the probability q_i of returning i in the first iteration of the implicit loop? There are $\lfloor nx_i/S\rfloor+1$ occurrences of i in A. If we randomly pick k to be the first occurrence of i in A, then we return i only with probability $\mathrm{frac}(nx_i/S)$ and reject it otherwise. If we pick k to be any other occurrence of i, then we return i right away. Thus, we have

$$q_i=\frac{\lfloor nx_i/S\rfloor+\mathrm{frac}(nx_i/S)}{|A|}=\frac{nx_i}{S|A|}.$$

Let $Q=\sum_j q_j$ denote the probability of returning anything in the first iteration of the implicit loop, i.e., the probability of leaving the loop in the first iteration. The total probability of sampling i with the above method is

$$\sum_{t\geqslant 0}q_i(1-Q)^t=\frac{nx_i}{S|A|}\sum_{t\geqslant 0}(1-Q)^t.$$

On the right hand side, the only term dependent on i is x_i. Hence, the probability of sampling i is proportional to x_i. Since we only sample numbers from $[n]$, this implies that the probability of sampling i is x_i/S, proving that the above method is indeed an exact sampling algorithm.

To show that it is also fast, note that the probability of leaving the loop in the first iteration is

$$Q=\sum_j q_j=\sum_j\frac{nx_j}{S|A|}=\frac{n}{|A|}\geqslant\frac{1}{2}.$$

Hence, the expected number of iterations is constant. In every iteration we sample uniform numbers in $[n]$ and $[S]$, which can be done in $\mathcal{O}(1)$ expected time, and, in particular, in time independent of the sampled number. Thus, the above sampling method needs in total $\mathcal{O}(1)$ expected time.

2.2 Redundancy of $\mathcal{O}(n+w)$

A simple encoding of A as in the last section is very wasteful. We show how to reduce the redundancy to $\mathcal{O}(n+w)$ bits. For this, we construct a bit array B of length $|A|$. The entry $B[k]$ is 1 if k is the first occurrence of $A[k]$ in A, and 0 otherwise. We store B in a data structure supporting RANK queries, where $\mathrm{RANK}_B(k):=\sum_{j=1}^k B[k]$ (with summation over the integers). Using, e.g., [11], RANK queries can be answered in constant time using a data structure of size $(1+o(1))|B|=\mathcal{O}(n)$ bits.

Observe that we have $\mathrm{RANK}_B(k)=A[k]$. Hence, using the RANK data structure for B, we can simulate the query algorithm from the last section and whenever it reads an array entry $A[k]$ we instead query $\mathrm{RANK}_B(k)$. Since we only need to store S and the RANK data structure for B, this reduces the redundancy to $\mathcal{O}(n+w)$ bits.

2.3 Arbitrary Redundancy

We show that we can further reduce the redundancy to $\mathcal{O}(n/k+w)$ bits at the cost of increasing the query time to $\mathcal{O}(k)$ for any integer $k\geqslant 1$. Choosing $k=cn/r$ for a sufficiently large constant $c>0$ implies Theorem 1.1.

We will partition $[n]$ into blocks of k elements. First, we show how to sample the block that contains the final sample. Then we show how to sample inside a block.

For ease of readability, assume that k divides n. On input $x_1,\dots,x_n$, consider the auxiliary instance $y_1,\dots,y_m$, where

$m = n/k$ and $y_i = \sum_{j=1}^{k} x_{ik+j}$. We first show how to sample with respect to $y_1, \ldots, y_m$. For this we make use of the data structure from the last section. Since we are not given input $y_1, \ldots, y_m$, but $x_1, \ldots, x_n$, we have to simulate this data structure and whenever it reads an input number y_i, we compute y_i on the fly from the input $x_1, \ldots, x_n$. This incurs an additional factor of k on both preprocessing and query time, totaling in $\mathcal{O}(mk) = \mathcal{O}(n)$ preprocessing and $\mathcal{O}(k)$ query time. Moreover, we need only $\mathcal{O}(m+w) = \mathcal{O}(n/k+w)$ bits of redundancy.

Next, given i as sampled above, we show how to sample $j \in \{ik+1, \ldots, (i+1)k\} =: J$ with probability x_j/y_i. To do so, we use a simple linear time sampling algorithm (see, e.g., [12, p. 120]): First we compute $y_i = \sum_{j \in J} x_j$. Then we sample a uniform random integer $R \in \{1, \ldots, y_i\}$. Finally, via linear search we determine the smallest index ℓ such that $\sum_{j=1}^{\ell} x_{ik+j} \geqslant R$ and return $ik + \ell$. This needs $\mathcal{O}(k)$ query time, and no preprocessing or redundancy.

Putting both parts together, for any index j there is a block i with $j \in \{ik+1, \ldots, (i+1)k\}$. We have probability y_i of sampling i in the first part and probability x_j/y_i of sampling j in the second part. In total, this yields a probability of x_j for sampling j, so we indeed described an exact sampling algorithm. We get the desired preprocessing time $\mathcal{O}(n)$, expected query time $\mathcal{O}(k)$, and redundancy $\mathcal{O}(n/k + w)$.

3. ONE ADDITIONAL BIT

In this section, we show that there is a data structure for sampling from a discrete distribution with redundancy 1, if the input is not read-only, i.e., in the non-systematic case. More precisely, we construct a data structure using $nw + 1$ bits of space in total that supports the operation DRAW in $\mathcal{O}(1)$ expected time and can be built in $\mathcal{O}(n)$ preprocessing time. Since we prove in Section 5 that it takes at least nw bits to describe the input, this corresponds to a redundancy of only 1 bit.

The First Bit. Let c be a sufficiently large constant integer to be fixed later. We start the description of the data structure by explaining the usage of the first bit of memory: In this bit we store whether we have

$$\sum_i x_i \geqslant 2^{w-c-1} n. \qquad (*)$$

Note that this can be computed in $\mathcal{O}(n)$ time preprocessing. The use of the rest of the bits depends on this first bit; we describe both cases in the next two paragraphs.

Large Sum. Assume that (*) holds, i.e., the first bit that we have stored is 1. Then the bound $x_i \leqslant 2^w$ for all i is on average a very tight upper bound, which is the standard situation to apply the rejection method (see [3]).

In this case, in the remaining nw bits we simply store the plain input $x_1, \ldots, x_n$. This completes the description of space usage and preprocessing.

To perform a DRAW operation we proceed as follows.

1. Pick a uniformly random number $i \in [n]$.
2. Rejection: With probability $1 - x_i/2^w$ goto 1.
3. Return i.

The analysis of this sampling method is similar to the analysis in Section 2.1. Note that in step 2 with some probability we go back to step 1, so there is an implicit loop. The probability of returning $i \in [n]$ in the first iteration of this loop is $\frac{x_i}{2^w n}$. Let Q denote the probability of returning anything in the first iteration of the implicit loop, i.e., the probability of leaving the loop in the first iteration. Then the total probability of sampling i with above method is

$$\sum_{t \geqslant 0} \frac{x_i}{2^w n} (1 - Q)^t.$$

Note that here the only term dependent on i is x_i. Hence, the probability of sampling i is proportional to x_i, so it has to be x_i/S, and we indeed have an exact sampling method.

To bound the method's expected runtime, consider the probability Q in more detail. We have

$$Q = \sum_i \frac{x_i}{2^w n} \overset{(*)}{\geqslant} 2^{-c-1} = \Omega(1).$$

Hence, the expected number of iterations of the implicit loop is bounded by a constant. In every iteration we sample a random number in $[n]$ and in $[2^w]$, which can be done in $\mathcal{O}(1)$ expected time, and, in particular, in time independent of the sampled number. Hence, in total the above method uses $\mathcal{O}(1)$ expected time.

Small Sum. Assume that we do not have (*), i.e., $\sum_i x_i < 2^{w-c-1} n$ and the first bit is 0. The intuition of how to proceed is as follows. Conditioned on $\sum_i x_i < 2^{w-c-1} n$, the entropy of the input is much less than nw. This allows to compress the input to $nw - \Omega(n)$ bits, while still guaranteeing efficient access to each input number x_i. Now we use the algorithm of Section 2, which generates $\mathcal{O}(n)$ redundant bits and performs a DRAW operation in $\mathcal{O}(1)$ expected time, if it is given access to the input numbers. The total space usage of writing down the compressed input and the redundant bits is then $nw - \Omega(n) + \mathcal{O}(n)$ bits, which is at most nw bits after adjusting constants.

In the following, we describe the details of the compression step. Let $I := \{i \in [n] \mid x_i \geqslant 2^{w-c}\}$. We store $x_1, \ldots, x_n$ in order, using w bits if $i \in I$, and only the $w - c$ least significant bits otherwise. This yields a bit string B. In order to read a value x_i from B we need to know where its encoding begins in B, and how many bits it uses. We achieve this by storing (the characteristic bit vector of) I in a data structure supporting RANK queries in $\mathcal{O}(1)$ query time using $\mathcal{O}(n)$ bits of space (using, e.g., [11]). Using this data, any value x_i can be read in constant time: Given i, we compute $k = \mathrm{RANK}_I(i-1)$. Then there are k input numbers encoded with w bits and $i - 1 - k$ input numbers encoded with $w-c$ bits preceding x_i. Hence, the encoding of x_i starts at position $kw + (i - 1 - k)(w - c)$. Moreover, the length of its encoding is w, if $i \in I$ (iff $\mathrm{RANK}_I(i) > \mathrm{RANK}_I(i-1)$), and $w - c$, otherwise.

Since this compression of the input allows us to read any input value in constant time, we can simulate the algorithm from Section 2 on the compressed input, decoding x_i whenever the algorithm reads it. This produces additional data of $\mathcal{O}(n)$ bits and allows us to sample from the input distribution in $\mathcal{O}(1)$ expected time.

In total the compressed input and the auxiliary data need $nw - (n - |I|)c + \mathcal{O}(n)$ bits. Since we are in the case where $\sum_i x_i < 2^{w-c-1}n$ and every $i \in I$ has $x_i \geqslant 2^{w-c}$, we can bound $|I| \leqslant n/2$. Thus, the total number of bits is at most $nw - \frac{n}{2}c + \mathcal{O}(n)$. For sufficiently large c, this is less than nw.

In both cases we need $\mathcal{O}(n)$ preprocessing, $\mathcal{O}(1)$ query time, and at most $nw + 1$ bits of storage, which finishes the proof of Theorem 1.4.

4. LOWER BOUNDS FOR READ-ONLY INPUTS

In this section, we prove a tight lower bound on the trade-off between redundancy and expected query time for read-only data structures for sampling from a discrete distribution. Throughout the section, we assume the availability of such a data structure using r redundant bits and supporting DRAW in expected time t.

Hard Distribution. For the proof, we consider a hard distribution over input numbers. Let $B \geqslant 1$ be a parameter to be fixed later and assume B divides n. We draw a random input $X = X_1, \ldots, X_n$ in the following manner: Partition the indices $\{1, \ldots, n\}$ into B consecutive groups of n/B indices each. For each group, select a uniform random index j in the group and let the corresponding input number X_j have the value 1. For all other indices in the group, we let the corresponding input number store the value 0. This constitutes the hard input distribution.

As a technical remark regarding our input distribution, note that the previous work on systematic RANK and SELECT structures allows access to multiple input elements by assuming the input is packed in machine words. Even though our hard distribution uses only 0's and 1's, we assume that the data structure may only access a single input number in one read operation. We note that this is a completely valid assumption, since we could just replace all 1's with $2^w - 1$ (or some other very large number) to enforce this restriction. Also, assuming that only one input number can be accessed with one read operation is more appropriate for situations, in which the input numbers are given implicitly, i.e., have to be computed when requested. Finally, we note that the tight lower bound for range minimum by Brodal et al. [2] also assumes that only one input element may be read in one operation.

For an intuition on why the above input distribution is hard, think of B as being a sufficiently large constant times r. Running DRAW on such an input must return an index j for which $X_j = 1$. Furthermore, the index returned is uniformly random among the indices having the value 1. Thus, the DRAW operation must find the location of a 1 inside a random block. Since there are so few redundant bits, we have less than 1 bit of information about each block on average, thus the only way to locate a 1 inside a block is to perform a linear scan, costing $\Omega(n/B) = \Omega(n/r)$ time. We prove that this intuition is correct in the rest of the section.

Note that

$$H(X) = B \lg(n/B)$$

bits, where $H(\cdot)$ denotes binary Shannon entropy. This is easily seen since X contains B 1's, each uniformly distributed inside a range of n/B indices.

An Encoding Proof. We prove our lower bound using an encoding argument. More specifically, we show that if a sampling data structures exists that is too efficient in terms of redundancy r and expected query time t, then we can use this data structure to encode (and decode) the random input X in less than $H(X)$ bits in expectation. This is an information theoretic contradiction.

The basic idea in the encoding procedure is to implement the claimed data structure on the input X and then run DRAW for $k = B/2$ times. We will then write down the input numbers X_j that the data structure reads during these executions along with the redundant bits. This will (essentially) be enough to recover the entire input X and thus we derive a contradiction if these numbers X_j and the redundant bits can be described in much less than $H(X)$ bits. Since the number of read operations depends on the query time, we derive lower bounds on the trade-off between the redundant bits and the query time.

As a last technical detail before we present the encoding and decoding procedures, we assume that on each invocation of DRAW, the sampling data structure is given access to a finite stream of uniform random bits that it uses to determine the index to return. Thus, if we fix the stream of random bits given to the data structure, the latter becomes completely deterministic and always returns the same index on the same input. The encoding and decoding procedures will share such random streams, thus they both know what "randomness" was used by the data structure when performing the k DRAWs. More formally, let $R_1, \ldots, R_k$ be k finite sequences of uniform random bits. Both the encoding and decoding procedure are given access to these sequences. Since X is independent of $R_1, \ldots, R_k$, we have $H(X \mid R_1 \cdots R_k) = H(X)$, i.e., we still derive a contradiction if the encoding uses less than $H(X)$ bits in expectation when the encoder and decoder share $R_1, \ldots, R_k$. We are finally ready to present the encoding procedure.

Encoding Procedure. Upon receiving the input numbers $X = X_1, \ldots, X_n$, we first implement the claimed data structure on X. Then we run DRAW for k times, using R_i as the source of randomness in the i'th invocation. We now do the following:

1. We write down the r redundant bits stored by the data structure on input X.

2. Now construct an initially empty set of indices C and an initially empty string z. For $i = 1, \ldots, k$ we examine the input numbers X_j read during the i'th Draw operation. For each such number X_j, in the order in which they are read, we first check whether $j \in C$. If not, we add j to C and append the value X_j (just a bit) to z. Otherwise, i.e., if j is already in C, we simply continue with the next number read. For each $i = 1, \ldots, k$, if the query algorithm is about to append the $(4t+1)$'st bit to z, we terminate the procedure for that i and continue with the next Draw. Also, if the i'th Draw operation terminates before $4t$ bits have been appended to z, we pad with 0s such that a total of $4t$ bits are always appended to z. Letting Y denote the number of 1s in z, the next part of the encoding consists of $\lg B$ bits specifying Y, followed by $\lg \binom{4tk}{Y}$ bits specifying z (there are $\binom{4tk}{Y}$ strings of length $|z| = 4tk$ having Y 1s).

   We note that the reason why we maintain C and only encode each X_j at most once, is that this forces Y to be proportional to the number of distinct 1s that we have seen. Since each distinct 1 reveals much information about X, this will eventually give our contradiction.

3. Finally, collect the set D containing all indices i for which $X_i = 1$ and where either $i \in C$ (it was read by one of the Draws), or the corresponding index was returned as the result of one of the Draws that terminated without appending more than $4t$ bits to z during step 2 (the data structure might return an index without reading the corresponding input number). For each $j = 0, \ldots, B-1$ (in this order), let i_j be the index of the 1 in X which is stored in the j'th group (numbers $X_{j(n/B)}, \ldots, X_{(j+1)(n/B)-1}$). If i_j is not contained in D, we write down the offset of i_j within its group, i.e. we write down the value $i_j - j(n/B)$. Since $|D| \geqslant Y$, this part of the encoding costs at most $(B-Y)\lg(n/B)$ bits.

Before analysing the expected size of the encoding, we present the decoding procedure:

Decoding Procedure. Recall we are given access to the random streams $R_1, \ldots, R_k$ during the decoding, i.e., we conditioned on these variables. To recover X from the above encoding, we now do the following:

1. First initialize an empty set $\bar{C}$, which eventually will contain pairs (i, Δ_i), where i is an index into $X = X_1, \ldots, X_n$ and Δ_i is the value stored at that index, i.e. $\Delta_i = X_i$. Now for $i = 1, \ldots, k$, start running the query procedure for Draw using R_i as the source of randomness. While running the i'th Draw, we maintain a pointer p_i into the string z which was constructed in step 2 of the encoding procedure. When starting the i'th Draw, p_i points to the first bit that was appended by the i'th Draw during step 2 of the encoding procedure. This bit is exactly the $((i-1)4t)$'th bit of z (counting from 0).

   When running the i'th Draw operation, the query procedure starts by requesting either some of the redundant bits or some input number. If it requests some of the redundant bits, we have those bits immediately from step 1 of the encoding procedure and can continue with the next step of the Draw procedure. If on the other hand it requests the number X_j, we first check whether there is a pair (j, Δ_j) in $\bar{C}$ for some Δ_j. If so, Δ_j equals X_j and we can continue the procedure. If not, we know from step 2 of the encoding procedure that the bit pointed to by p_i stores the value X_j and we can again continue the procedure after incrementing $p_i \leftarrow p_i + 1$ and adding the pair (j, X_j) to $\bar{C}$. If at any step we are about to increment p_i for the $(4t+1)$'st time, we simply abandon the i'th Draw and continue with the next. Clearly these k invocations of Draw allow us to recover the set D.

2. From the set D recovered above, we can deduce the groups in X for which the index of the corresponding 1 is not in D. It finally follows that we can recover X from D and the bits written down during step 3 of the encoding procedure.

What remains is to analyse the size of the encoding and derive the lower bound.

Analysis. The number of bits in the encoding, denoted by K, is precisely

$$\begin{aligned} K &= r + \lg B + \lg \binom{4tk}{Y} + (B-Y)\lg(n/B) \\ &\leqslant r + \lg B + Y\lg(4etk/Y) + (B-Y)\lg(n/B) \\ &= H(X) + r + \lg B - Y\lg(nY/4etkB) \\ &= H(X) + r + \lg B - Y\lg(nY/2etB^2). \end{aligned}$$

The only random variable in the above is Y and since

$$Y\lg(nY/2etB^2) = Y\lg Y + Y\lg(n/2etB^2)$$

is convex, we get from Jensen's inequality that

$$\mathbb{E}[K] \leqslant H(X) + r + \lg B - \mathbb{E}[Y]\lg(n\mathbb{E}[Y]/2etB^2).$$

Now observe that each of the $k = B/2$ calls to Draw returns an element not returned in any of the other Draws with probability at least $1/2$. Furthermore, we get from Markov's inequality that each Draw terminates within the first $4t$ steps with probability at least $3/4$. From a union bound, we conclude $\mathbb{E}[Y] \geqslant B/8$. Inserting this in the above, we have

$$\mathbb{E}[K] \leqslant H(X) + r + \lg B - B\lg(n/16etB)/8.$$

Choosing $B = 8r$, we get (using $r + \lg(8r) \leqslant 4r$):

$$\mathbb{E}[K] \leqslant H(X) + 4r - r\lg(n/2^5etr),$$

from which we conclude that the claimed data structure must satisfy

$$\begin{aligned} 4r &\geqslant r\lg(n/2^5etr) \Rightarrow \\ 4 + \lg(2^5e) &\geqslant \lg(n/tr) \Rightarrow \\ tr &= \Omega(n). \end{aligned}$$

This completes the proof of Theorem 1.2.

5. SPACE LOWER BOUND

In this section, we prove that the information theoretic minimum number of bits needed to sample from a discrete distribution is nw bits for any $1 \leqslant w = o(n)$ and n sufficiently large. For this, observe that two inputs $x_1, \ldots, x_n$ and $\hat{x}_1, \ldots, \hat{x}_n$ represent the same probability distribution only if there exists a value $\alpha > 0$ such that $x_i = \alpha \hat{x}_i$ for all $1 \leqslant i \leqslant n$. We want to show that there are not too many pairs of inputs for which this is true. To prove this, define an input set of w-bit integers $x_1, \ldots, x_n$ to be *irreducible* if for all $0 < \alpha < 1$, there is at least one $i \in \{1, \ldots, n\}$ for which αx_i is not an integer. Clearly, any two distinct and irreducible inputs represent two distinct probability distributions. First, we prove that the following condition is sufficient to guarantee irreducibility:

LEMMA 5.1. *An input set of w-bit integers $x_1, \ldots, x_n$ is irreducible if there are at least two distinct primes among $x_1, \ldots, x_n$.*

PROOF. Assume $x_i = p$ and $x_j = q$ for some $i \neq j$ and some primes $p \neq q$. Assume also that $x_1, \ldots, x_n$ is not irreducible. This implies the existence of a value $0 < \alpha < 1$ such that $\alpha p = c_p$ and $\alpha q = c_q$ for some integers $c_p, c_q \geqslant 1$. Now since $\alpha < 1$, we have $c_p < p$ and hence p is not a prime factor in c_p. But $c_q = \alpha q = c_p q/p$ and it follows that p is not a prime factor in $c_p q$, thus c_q cannot be integer, i.e., a contradiction. □

For the remaining part of the proof, consider drawing each x_i as a uniform random integer in $[2^w]$. The probability that a particular x_i is prime is $\Omega(1/w)$. Thus, for $2 \leqslant w = o(n)$ and any sufficiently large n, the number of distinct primes in the randomly chosen $x_1, \ldots, x_n$ is at least two with high probability, certainly with probability at least $3/4$. Since we chose $x_1, \ldots, x_n$ uniformly at random, we conclude that at least $(3/4)2^{nw}$ of the 2^{nw} possible inputs are irreducible. Therefore, any sampling data structure must use at least

$$\lceil \lg ((3/4)2^{nw}) \rceil = nw$$

bits of space. In the case $w = 1$, the result follows immediately.

Acknowledgment

The authors wish to thank the China Theory Week 2012 Workshop for initiating the collaboration leading to the results of this paper.

References

[1] K. Bringmann and K. Panagiotou. Efficient sampling methods for discrete distributions. In *Proc. 39th International Colloquium on Automata, Languages, and Programming (ICALP'12)*, pages 133–144. Springer, 2012.

[2] G. S. Brodal, P. Davoodi, and S. S. Rao. On space efficient two dimensional range minimum data structures. In *Proc. 18th Annual European Symposium on Algorithms (ESA'10)*, volume 6347 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2010.

[3] L. Devroye. *Nonuniform random variate generation*. Springer, New York, 1986.

[4] Y. Dodis, M. Pătraşcu, and M. Thorup. Changing base without losing space. In *Proc. 42nd ACM Symposium on Theory of Computing (STOC'10)*, pages 593–602, 2010.

[5] J. Fischer and V. Heun. A new succinct representation of RMQ-information and improvements in the enhanced suffix array. In *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, volume 4614 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2007.

[6] P. Flajolet, M. Pelletier, and M. Soria. On buffon machines and numbers. In *Proc. 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'11)*, pages 172–183. SIAM, 2011.

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

[8] A. Golynski. Optimal lower bounds for rank and select indexes. *Theoretical Computer Science*, 387(3):348–359, Nov. 2007.

[9] A. Golynski, R. Raman, and S. S. Rao. On the redundancy of succinct data structures. In *Proc. 11th Scandinavian Workshop on Algorithm Theory (SWAT'08)*, pages 148–159, 2008.

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

[11] G. Jacobson. Space-efficient static trees and graphs. In *Proc. 30th Annual Symposium on Foundations of Computer Science (FOCS'89)*, pages 549–554. IEEE Computer Society, 1989.

[12] D. E. Knuth. *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms*. Addison-Wesley Publishing Co., Reading, Mass., third edition, 2009.

[13] D. E. Knuth and A. C. Yao. The complexity of nonuniform random number generation. In *Algorithms and Complexity: New Directions and Recent Results*, pages 357–428. Academic Press, 1976.

[14] R. A. Kronmal and A. V. Peterson. On the alias method for generating random variables from a discrete distribution. *The American Statistician*, 33(4):214–218, 1979.

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

[16] M. Pătraşcu. Succincter. In *Proc. 49th IEEE Symposium on Foundations of Computer Science (FOCS'08)*, pages 305–313, 2008.

[17] M. Pătraşcu and E. Viola. Cell-probe lower bounds for succinct partial sums. In *Proc. 21st ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, pages 117–122, 2010.

[18] M. Pătraşcu. Webdiarios de motocicleta, sampling a discrete distribution. http://infoweekly.blogspot.de/2011/09/follow-up-sampling-discrete.html, 2011.

[19] R. Raman, V. Raman, and S. R. Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4), 2007.

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

[21] A. C. Yao. Context-free grammars and random number generation. In *Combinatorial algorithms on words*, volume 12, pages 357–361. Springer, 1985.