A new algorithm for sampling without replacement

Paul Crowley

Google LLC

November 8, 2018

1 Introduction

Many ways of fairly choosing k distinct integers from $[0 \dots n-1]$ are in the literature. Here I introduce a method which as far as I know is novel; in my tests it is the most efficient for most values of k, n, while requiring no complex data structure such as an efficient set or dictionary, just an integer sort taking $\mathcal{O}(k \log k)$ time.

To gain an intuition for the method, we use the "stars and bars" method. There are $\binom{6+5}{6}$ ways of arranging 6 stars and 5 bars into a sequence. To choose one of these ways fairly, we start with a sequence of 5 bars and choose where to put the first star; there are six possible places.

$$| \ | \ | \ | \ |$$

We place the first star in the fourth position.

$$| \ | \ | \ \star \ | \ |$$

We record that there are three bars to its left. There are now seven places we can place the second star. It may end up with anything from zero to five bars to its left, but because there's a space either side of the first star, it's twice as likely it'll have three bars to its left as four or any other number.

$$\star \ | \ | \ | \ \star \ | \ |$$

We place the next star at the beginning and record that there are 0 bars to its left. If we had recorded the position of the first star, that would now be out of date as everything moves one position to the right, but the number of bars to its left stays constant at three.

By the time all but the last star is placed, it's four times as likely that a star will be placed with no bars to its left than with five bars.

$$\star \ \star \ \star \ | \ \star \ | \ | \ \star \ | \ |$$

After placing the last star, we sort the list counting bars to the left for each star. The position of each stars is just the number of bars or stars to its left, so to each we add its index in the list to learn the positions; our final answer is $\{0, 1, 2, 4, 5, 8\}$.

$$\begin{array}{cccccccccccc} 0 & 0 & 0 & & 1 & 1 & & & 3 & & & \\ \star & \star & \star & | & \star & \star & | & | & \star & | & | \\ 0 & 1 & 2 & & 4 & 5 & & & 8 & & & \end{array}$$

Thus to get a sequence without duplicates, we start with a procedure that deliberately biases towards duplicates.

2 Choosing a multiset

A multiset is an extension of a set in which elements can appear more than once. $[0, 1, 1]$ is the same multiset as $[1, 0, 1]$ but distinct from $[0, 0, 1]$.

We represent multisets as functions $m : U \to \mathbb{N}$; the set U is the *universe*, and in what follows we consider only finite universes. For any $y \in U$ we call $m(y)$ the *multiplicity* of y in m. A multiset has a *cardinality* $|m| = \sum_{x \in U} m(x)$ and a *support* set $\mathrm{Supp}(m) = \{x \in U : m(x) > 0\}$. Where U is clear from context, for any set $S \subseteq U$ we consider $\overline{S}$ to be S viewed as a multiset, ie the multiset such that $\mathrm{Supp}(\overline{S}) = S$ and $|\overline{S}| = |S|$:

$$\overline{S}(x) = \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{otherwise} \end{cases}$$

The sum of multisets $m = m_1 \uplus m_2$ is the multiset such that $m(x) = m_1(x) + m_2(x)$ for all $x \in U$. $m = m_1 \ominus m_2$ is the unique multiset such that $m_1 = m \uplus m_2$, and is defined only if this exists.

Define $\mathcal{M}(U, k) = \{m \in U \to \mathbb{N} : |m| = k\}$; we prove later that $|\mathcal{M}(U, k)| = \binom{|U|+k-1}{k}$. In what follows we consider the problem of choosing an element from $\mathcal{M}(U, k)$ fairly. For example, $\mathcal{M}(\{0, 1\}, 3) = \{[0, 0, 0], [0, 0, 1], [0, 1, 1], [1, 1, 1]\}$; for each of these four, our algorithm should output it with probability $\frac{1}{4}$. Simply choosing three independent elements from U and adding them together to make a multiset won't work; in accordance with the binomial theorem, $[0, 0, 0]$ will be drawn with probability $\frac{1}{8}$, while $[0, 0, 1]$ will be drawn with probability $\frac{3}{8}$, reflecting the

three ways this multiset can be written as a sequence. It is straightforward to show by induction that in general, this naive method chooses $m \in \mathcal{M}(U, k)$ with probability

$$\frac{k!}{|U|^k \prod_{x \in U} m(x)!}$$

Algorithm 1: Good multiset choosing

procedure ChooseMultiset(U, k)
  if k = 0 **then**
    return $\overline{\varnothing}$
  else
    $m' \leftarrow$ ChooseMultiset$(U, k-1)$
    $x \leftarrow_\$ \overline{U} \uplus m'$
    return $m' \uplus \overline{\{x\}}$
  end if
end procedure

To address this, in ChooseMultiset we introduce a counter-bias in the selection of x, which favours duplicates. We define random sampling from a finite multiset to be analogous to drawing from a set, where each element's probability of being drawn is proportional to its multiplicity: $\Pr[x = y | x \leftarrow_\$ m] = \frac{m(y)}{|m|}$. ChooseMultiset is trivially fair for $k = 0$, so we assume it is fair for $k - 1$ and proceed by induction. For a multiset $m \in \mathcal{M}(U, k)$:

$$\begin{aligned}
&\Pr[\text{ChooseMultiset}(U,k) \to m] \\
=&\Pr\left[m' \uplus \overline{\{x\}} = m | m' \leftarrow \text{ChooseMultiset}(U, k-1), x \leftarrow_\$ \overline{U} \uplus m'\right] \\
=&\Pr\left[m' \uplus \overline{\{x\}} = m | m' \leftarrow_\$ \mathcal{M}(U, k-1), x \leftarrow_\$ \overline{U} \uplus m'\right] \\
=&\sum_{y \in \text{Supp}(m)} \Pr\left[m' = m \ominus \overline{\{y\}} | m' \leftarrow_\$ \mathcal{M}(U, k-1)\right] \Pr\left[x = y | x \leftarrow_\$ \overline{U} \uplus (m \ominus \overline{\{y\}})\right] \\
=&\sum_{y \in \text{Supp}(m)} \frac{1}{|\mathcal{M}(U, k-1)|} \frac{(U \uplus (m \ominus \overline{\{y\}}))(y)}{|U \uplus (m \ominus \overline{\{y\}})|} \\
=&\frac{1}{|\mathcal{M}(U, k-1)|} \sum_{y \in \text{Supp}(m)} \frac{1 + (m \ominus \overline{\{y\}})(y)}{|U| + |m \ominus \overline{\{y\}}|} \\
=&\frac{1}{|\mathcal{M}(U, k-1)|} \sum_{y \in \text{Supp}(m)} \frac{m(y)}{|U| + k - 1} \\
=&\frac{k}{(|U| + k - 1)|\mathcal{M}(U, k-1)|} = \frac{k}{(|U| + k - 1)\binom{|U|+k-2}{k-1}} = \frac{1}{\binom{|U|+k-1}{k}} = \frac{1}{|\mathcal{M}(U, k)|}
\end{aligned}$$

This algorithm is straightforward to implement. This Python implementation takes integers n, k and returns a sorted list of integers in $[0 \dots n-1]$.

```
def choose_multiset(n, k):
    d = []
    for i in range(k):
        r = random.randrange(n + i)
        if r < n:
            d.append(r)
        else:
            d.append(d[r - n])
    d.sort()
    return d
```

3 Multisets and choices

For a set U, define $\binom{U}{k} = \{S \subseteq U : |S| = k\}$; then $\left|\binom{U}{k}\right| = \binom{|U|}{k}$. Using the method of "stars and bars", we will use our solution to the multiset choosing problem to choose fairly at random from $\binom{U}{k}$.

Consider a sequence that contains k stars and b bars in any order.

0	1	2		4	5			8		
⋆	⋆	⋆	|	⋆	⋆	|	|	⋆	|	|
0	0	0		1	1			3		

If we label each star with its position in the sequence, we get an element of $\binom{[0\dots(b+k-1)]}{k}$; if we label each star with the number of bars to its left, we get an element of $\mathcal{M}([0 \dots b], k)$. In each case, this relation is a bijection with the possible sequences, demonstrating a bijection between the two sets. It trivially follows that $|\mathcal{M}(U, k)| = \binom{|U|+k-1}{k}$ as stated above.

Our implementation of ChooseMultiset represents its result in $\mathcal{M}([0 \dots b], k)$ as a sorted list of integers. In "stars and bars" representation, each entry in the list represents a star, and the integer is the number of bars to its left. Converting this to a sorted element of $\binom{|U|+k-1}{k}$ simply means adding to each the number of stars to its left, which is equal to its position in the sequence; the Python code below returns a sorted list of k distinct integers in $[0 \dots n-1]$ fairly among all ways of doing so.

```
def choose_binom(n, k):
    d = choose_multiset(n - k + 1, k)
    for i in range(k):
        d[i] += i
    return d
```