A new algorithm for sampling without replacement

Paul Crowley

Google LLC

November 23, 2022

Abstract

Many ways of fairly choosing k distinct integers from $\{0 \dots n-1\}$ are in the literature. However, they either require a hash-based data structure such as a set or dictionary, or show asymptotically poor performance for some values of k, n. We present here an algorithm that requires no such data structure or auxiliary storage, only an integer sort taking $\mathcal{O}(k \log k)$ time; the algorithm is based on a method of fair multiset choosing. In our benchmarks the algorithm outperforms all we compare it to wherever $k > 100$ and $n > 100k$, and has acceptable performace for all values of k, n.

1 Introduction

In Python the algorithm works as follows:

```python
def sorted_choose(n, k):
    "k distinct integers 0 <= x < n, sorted"
    t = n - k + 1
    d = [None] * k
    for i in range(k):
        r = random.randrange(t + i)
        if r < t:
            d[i] = r
        else:
            d[i] = d[r - t]
    d.sort()
    for i in range(k):
        d[i] += i
    return d
```

2 Intuition behind the algorithm

Every way of choosing 6 integers from the range $\{0 \dots 10\}$ is equivalent to a way of arranging 6 stars and 5 bars into a sequence; there are $\binom{6+5}{6}$ ways of doing so.

To choose one fairly, we start with a sequence of five bars $|||||$ and insert six stars, one after another, in randomly chosen positions.

For the first star, there are six possible places it can go, and we choose one at random: $||| \star ||$. We record that it has three bars to its left $\{3\}$.

There are now six items in the sequence, and thus seven possible places to place the second star. In two of those seven cases—before the existing star, and after it—it will have three bars to its left. Let's suppose we choose the first position: $\star||| \star ||$. We append the number of bars to the left of the new star to our record, which becomes $\{3, 0\}$. The positions have changed from $\{3\}$ to $\{4, 0\}$ but because we're not recording positions, only bars to the left, we don't need to update the first entry.

We place three more stars in random positions, ending up with $\star\star\star| \star || \star ||$ and a record of $\{3, 0, 0, 1, 0\}$. Now there's one star left to place; there is one position it can be placed after the last bar, but four before the first bar, so it is four times more likely to be placed before the first bar than after the last. Let's suppose it's placed at the fifth position: $\star\star\star| \star\star|| \star ||$, $\{3, 0, 0, 1, 0, 1\}$ We now want to know the position of each star; we find this by sorting the list $\{0, 0, 0, 1, 1, 3\}$ and adding to each entry its index so that the value reflects the stars as well as the bars to its left, returning the answer $\{0, 1, 2, 4, 5, 8\}$

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

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

3 Multisets

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 $[0, 0, 1]$. The algorithm works by choosing a multiset at random, then converting that to a choice of subset.

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 $m_1 \uplus m_2$ is the multiset such that $(m_1 \uplus m_2)(x) = m_1(x) + m_2(x)$ 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.

We define random sampling from a multiset to be analogous to drawing from a set, where each element's probability of being drawn is proportional to its multiplicity: $\Pr[x = y | x \leftarrow\!\$\; m] = \frac{m(y)}{|m|}$.

Define $\mathcal{M}(U, k) = \{m \in U \to \mathbb{N} : |m| = k\}$ the set of multisets over universe U of cardinality k. The "stars and bars" method shows that for a set S and for any U and k such that $|S| = |U| + k - 1$, there is a bijection between $\mathcal{M}(U, k)$ and the set of cardinality k subsets of S, from which we infer that $|\mathcal{M}(U, k)| = \binom{|U|+k-1}{k}$ wherever $|U| + k > 0$.

4 Multiset choosing

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 $\frac{1}{4}$. If we choose three independent elements from U and add them together to make a multiset, our answer will favour multisets with lower multiplicities; 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.

Algorithm 1: Fair 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. ChooseMultiset is trivially fair for $k = 0$, so we assume it is fair for $k - 1$ and proceed by induction. For a multiset $m \in \mathcal{M}(U, k)$:

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

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

```python
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
```

5 Multisets and choices

To generate a random k-element subset of $\{0 \ldots n-1\}$, we can apply this method to generate a random multiset from the universe $\{0 \ldots n-k\}$ and use the "stars and bars" bijection to convert to the desired subset.

Our implementation of ChooseMultiset represents its result in $\mathcal{M}([0 \ldots n-k], k)$ 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 k-element subset of $\{0 \ldots n-1\}$ 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 $[0 \ldots n-1]$ fairly among all ways of doing so.

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