Summer of Math Exposition 4: Sampling

Ian Chen

August 8, 2025

Sampling is a fundamental part in statistical studies. Instead of collecting data from a large population, which is often infeasible at worse and expensive at best, statisticians collect a *representative sample* and infer conclusions about the population from the sample. Here, we will explore algorithms to efficiently sample from data streams.

We expect a prerequisite knowledge of Big-O and elementary probability. See §4 for a refresher.

In §1, we develop the problem and a simple solution. Then, releasing the assumption that we know the population size, we explore a simple type of resevoir sampling algorithm in §2 and an application in §2.1. Finally, we refine it to optimality in §3. Proofs for some theorems are in §5, but I encourage the reader to attempt them as exercises.

1 Problem Definition

Let $D = (x_1, x_2, \ldots, x_N)$. Let k be the number of samples we wish the collect, and N the size of the stream (if known). We wish to compute a sample (w/o replacement) $S(D) \subset D$, where each x_i has an equal chance of being in the sample, and $\left|S(D)\right| = \min(n, \left|D\right|)$. We are allowed to use the functions *random*(), generating a uniform real in $[0, 1)$, and the *randint*(a, b), generating a uniform integer in $[a, b]$.

Here's an example:

There are a few properties of this approach that can be undesireable, which all stem from the fact that we need to store the entire data stream. This is because we make multiple passes of the data, first in finding the sum of all the weights, and a new pass when selecting an element (and removing). From here on, we explore resevoir sampling (Vitter 1985), which are single-pass sampling techniques.

2 Bottom-k Sampling

The key idea here is that we can select a random element by assigning a random key y_i for each x_i, and picking the minimum. Moreover, we can select k items by picking the k smallest keys.

The correctness follows from the following theorem.

Theorem 1. *Let $Y_1, \ldots, Y_n \overset{iid}{\sim} Y$, and suppose WLOG that they are distinct. Then, $\mathbf{Pr}(r_i \leq k) = \frac{k}{n}$, where $r_i = \left|\{Y_j \mid Y_j \leq Y_i\}\right|$ is the rank of Y_i.*

Algorithm 1 Bottom-k Sampling

```
function GenerateSample(D, k)
    Initialize a resevoir with the first k keys
    for each i > k in increasing order do
        y_i ← random()
        if y_i is in the lowest k keys explored so far then
            Update the resevoir to include x_i (and remove the largest)
        end if
    end for
end function
```

2.1 Application

This section requires terminology of graph theory and can be skipped. It is adapted from §3 of Cohen 1997.

For each vertex $v \in V$, we want to sample k vertices (w/o replacement) from its 2-hop neighborhood. Cohen 1997 use this sample to estimate the size of the neighborhood, but we focus more on the sampling process here.

Following the ideas developed in §2, we first assign a random key for each vertex. Then, for each vertex, we need to compute the k least keys in its neighborhood.

Algorithm 2 Sampling from 2-Hop Neighborhood

```
function Sample2Hop(V, E, k)
    for i from 1 → k do
        y_j ← random
        for j in increasing y_j do
            if v_j is done then
                continue
            end if
            for u ∈ two-hop neighborhood of v_j do
                Add v_j to S_u if u has not been added to yet
                Remove edges incident to u
                Mark u as done
            end for
        end for
    end forreturn {S_i}
end function
```

Theorem 2. *For each vertex, we can collect samples of k vertices (w/o) replacement from its 2-hop neighborhood in $O(k(n \log n + m))$ time.*

Proof. First, we find that Algorithm 2 correctly solves the problem because for each vertex, it finds the minimum k keys in its reachability set. Thus, by our analysis in §2, we correctly generate a sample.

Now, we prove the runtime. Consider an iteration of the algorithm. First, we generate random

weights, which is done in $O(n)$ time, and sort them in $O(n \log n)$ time. Finally, we proceed by doing a reachability search, which visits every edge at most once and so takes $O(m)$ time. Repeating this algorithm k times, we obtain a final runtime of $O(k(n \log n + m))$. □

3 Improved Resevoir Sampling

There is one more trick that we can apply to speed up the algorithm even further. Consider how much information we need to simulate the output of Algorithm 1; all we need to know is which indices do we update our resevoir of k smallest keys. In fact, the gap between indices follow a geometric distribution, and Algorithm 1 is just a complicated way of generating a geometric distribution. Using this observation, we can avoid examining each item one by one and instead skip between items (Li 1994; Efraimidis and Spirakis 2006):

Algorithm 3 GenerateSampleFast

```
1: function GenerateSampleFast(D, k)
2:     Initialize a resevoir with the first k keys
3:     i ← k + 1                                          ▷ the current index
4:     M ← random()^{1/k}                                 ▷ distributed like max_{1≤j≤j}(Uniform(0, 1))
5:     m ← −ln(random())/M          ▷ Geom(M), the steps before find a uniform with value ≤ M
6:     while i + m ≤ N do
7:         i ← i + m
8:         Replace a random item in resevoir with x_i
9:         M ← M · random()^{1/k}                         ▷ distributed like max_{1≤j≤j}(Uniform(0, M))
10:        m ← −ln(random())/M          ▷ Geom(M), the steps before find a uniform with value ≤ M
11:    end while
12: end function
```

4 Prerequisites

5 Proofs

Theorem 1. Suppose this. Then, we win. □

References

Cohen, Edith (Dec. 1997). "Size-Estimation Framework with Applications to Transitive Closure and Reachability". In: *Journal of Computer and System Sciences* 55.3, pp. 441–453. ISSN: 0022-0000. DOI: 10.1006/jcss.1997.1534. URL: http://dx.doi.org/10.1006/jcss.1997.1534.

Efraimidis, Pavlos S. and Paul G. Spirakis (Mar. 2006). "Weighted random sampling with a reservoir". In: *Information Processing Letters* 97.5, pp. 181–185. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2005.11.003. URL: http://dx.doi.org/10.1016/j.ipl.2005.11.003.

Li, Kim-Hung (Dec. 1994). "Reservoir-sampling algorithms of time complexity O(n(1 + log(N/n)))". In: *ACM Trans. Math. Softw.* 20.4, pp. 481–493. ISSN: 0098-3500. DOI: 10.1145/198429.198435. URL: https://doi.org/10.1145/198429.198435.

Vitter, Jeffrey S. (Mar. 1985). "Random sampling with a reservoir". In: *ACM Trans. Math. Softw.* 11.1, pp. 37–57. ISSN: 0098-3500. DOI: 10.1145/3147.3165. URL: https://doi.org/10.1145/3147.3165.