

# Sublinear Time and Space Algorithms 2026A – Lecture 9

## Hash Functions with Limited Randomness and Triangle Counting\*

Robert Krauthgamer

### 1 Hash Functions with Limited Randomness

**Idea:** The idea is to replace a truly random function  $h : [n] \rightarrow [n]$  with something that is easier to store.

As a running example, consider  $h_{p,q}(i) = pi + q \pmod n$ , where  $p, q$  are chosen at random. This can be also viewed as choosing  $h$  from a family  $H = \{h_{p,q} : p, q\}$ . While  $h(1), \dots, h(n)$  are random but with some correlations, they can be stored (even the entire  $h$ ) with much less space than a truly random function.

To analyze these families formally, we need some definitions.

**Independent random variables:** Recall that two (discrete) random variables  $X, Y$  are independent if

$$\forall x, y \quad \Pr[X = x, Y = y] = \Pr[X = x] \cdot \Pr[Y = y].$$

This is equivalent to saying that the conditioned random variable  $X|Y$  has exactly the same distribution as  $X$ . It implies that  $\mathbb{E}[XY] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$ .

The above naturally extends to  $k > 2$  variables, and then we say the random variables are mutually (or fully) independent.

**Pairwise independence:** A collection of random variables  $X_1, \dots, X_n$  is called *pairwise independent* if for all  $i \neq j \in [n]$ , the variables  $X_i$  and  $X_j$  are independent.

Example: Let  $X, Y \in \{0, 1\}$  be random and independent bits, and let  $Z = X \oplus Y$ . Then  $X, Y, Z$  are clearly not mutually (fully) independent, but they are pairwise independent.

Observation: When  $X_1, \dots, X_n$  are pairwise independent and have finite variance,  $\text{Var}(\sum_i X_i) = \sum_i \text{Var}(X_i)$ , exactly as if they were fully independent.

Exer: Prove this.

---

\*These notes summarize the material covered in class, usually skipping proofs, details, examples and so forth, and possibly adding some remarks, or pointers. The exercises are for self-practice and need not be handed in. In the interest of brevity, most references and credits were omitted.

Here too,  $k$ -wise independence means that every subset of  $k$  random variables is independent.

**Pairwise independent hash family:** A family  $H$  of hash functions  $h : [n] \rightarrow [M]$  is called *pairwise independent* if  $h(1), \dots, h(n)$  are pairwise independent when choosing random  $h \in H$ . This means that for all  $i \neq j \in [n]$ ,

$$\forall x, y \in [M] \quad \Pr_{h \in H} [h(i) = x, h(j) = y] = \Pr[h(i) = x] \cdot \Pr[h(j) = y].$$

A common scenario is that each  $h(i)$  is uniformly distributed over  $[M]$ , although this is not required in the above definition.

**Universal hashing:** A family  $H$  of hash functions  $h : [n] \rightarrow [M]$  is called *2-universal* if for all  $i \neq j \in [n]$ ,

$$\Pr_{h \in H} [h(i) = h(j)] \leq 1/M.$$

Observe that 2-universality is weaker than (follows from) pairwise independence when each  $h(i)$  is distributed uniformly over  $[M]$ , but it suffices for many algorithms.

**Construction of pairwise independent hashing:**

Assume  $M \geq n$  and that  $M$  is a prime number (if not, we can pick a larger  $M$  that is a prime). Pick random  $p, q \in \{0, 1, 2, \dots, M-1\} = [M]$  and set accordingly  $h_{p,q}(i) = pi + q \pmod{M}$ .

The family  $H = \{h_{p,q} : p, q\}$  is pairwise independent because for all  $i \neq j$ ,

$$\Pr_{h \in H} [h(i) \equiv x, h(j) \equiv y] = \Pr_{p,q} \left[ \begin{pmatrix} i & 1 \\ j & 1 \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} \equiv \begin{pmatrix} x \\ y \end{pmatrix} \right] = \Pr_{p,q} \left[ \begin{pmatrix} p \\ q \end{pmatrix} \equiv \begin{pmatrix} i & 1 \\ j & 1 \end{pmatrix}^{-1} \begin{pmatrix} x \\ y \end{pmatrix} \right] = \frac{1}{M^2},$$

where we relied on the above matrix being invertible.

Storing a function  $h_{p,q}$  from this family can be done by storing  $p, q$ , which requires  $\log |H| = O(\log M)$  bits. One can think of  $p, q$  as a random seed that generates (deterministically) the random variables  $h(0), \dots, h(n-1)$ .

In general,  $\log |H|$  bits suffice to store a choice of a function  $h \in H$ .

One can reduce the size of the range  $[M]$  (from large  $M \geq n$  to  $M = 2$  or say  $4/\alpha$ ), with a small overhead/loss.

**Exer:** Show that the correctness of algorithm CountMin (for  $\ell_1$  point query) extends to using a universal hash function, and analyze how much additional storage the hash function requires.

**Exer:** Show that the correctness of algorithm CountSketch (for  $\ell_2$  point query) can be implemented with limited (pairwise) independence and analyze how much additional storage the hash function requires.

Hint: use separate randomness for the hash function and for the signs.

**Exer:** Show that algorithm AMS (for estimating  $\ell_2$  norm) works even if the random signs  $\{r_i\}$  are only 4-wise independent.

## 2 Triangle Counting

**Goal:** Report the number of triangles, denoted by  $T$ , in a graph  $G$  given as a stream of  $m$  edges on vertex set  $V = [n]$ .

Motivation: The relative frequency of how often 2 friends of a person know each other is defined as

$$F = \frac{3T}{\sum_{v \in V} \binom{\deg(v)}{2}}.$$

We can compute  $\sum_{v \in V} \binom{\deg(v)}{2}$  exactly in  $O(n)$  space, by maintaining the degree of every vertex, and we can also approximate it using  $\text{polylog}(n)$  space using algorithms that estimate  $\ell_2$ -norm.

Distinguishing  $T = 0$  from  $T = 1$  is known to require  $\Omega(m)$  space [Braverman, Ostrovsky, and Vilenchik, 2013].

We will henceforth assume a known lower bound  $0 < t \leq T$ .

**First Approach [Bar-Yossef, Kumar and Sivakumar, 2002]:**

Idea: use frequency moments.

Define vector  $x \in \mathbb{R}^{\binom{n}{3}}$ , where every coordinate  $x_S$  counts the number of edges internal to a subset  $S \subset V$  of 3 vertices. Then

$$T = \#\{S \subset V, |S| = 3 : x_S = 3\}.$$

**Lemma:** Let  $F_p = \|x\|_p^p$  be the frequency moments for  $p = 0, 1, 2$  (well, actually  $F_0 = \|x\|_0$ ). Then

$$T = F_0 - 1.5F_1 + 0.5F_2.$$

Proof: As seen in class it suffices to verify that each coordinate  $x_S$  contributes the same amount to both sides.

**Why such a formula exists?:** We are looking for coefficients, i.e., a polynomial  $f(x_S) : \mathbb{R} \rightarrow \mathbb{R}$  with specific values  $f(3) = 1$  and  $f(2) = f(1) = f(0) = 0$ . We can do polynomial interpolation over 4 points. It would generally require degree 3, but  $F_0 = \mathbf{1}_{\{x_S > 0\}}$  gives an extra degree of freedom.

**Algorithm 1:**

Update: Maintain the frequency moments  $p = 0, 1, 2$  of vector  $x \in \mathbb{R}^{\binom{n}{3}}$ . Initially  $x = 0$ , and when an edge  $(u, v)$  arrives, increment  $x_S$  for every  $S \supseteq \{u, v\}$ .

Output: Compute moment estimates  $\hat{F}_p$  and report  $\hat{T} = \hat{F}_0 - 1.5\hat{F}_1 + 0.5\hat{F}_2$ .

**Correctness:** As was seen in class, suppose we compute frequency estimates  $\hat{F}_p \in (1 \pm \gamma)F_p$ . We can then set a suitable  $\gamma = \Omega(\frac{\varepsilon t}{mn})$  (for given  $t$  and  $\varepsilon$ ), and the additive error will be bounded by  $\varepsilon t \leq \varepsilon T$ .

**Storage:** The storage requirement is  $O(\gamma^{-2} \log n) = O(\varepsilon^{-2} (\frac{mn}{t})^2 \log n)$  words, which is effective when  $t$  is large (close to  $mn$ ), but poor for small  $t$ .

Observe that this algorithm works even for streams with deletions.