CSC791 – Advanced Algorithms Krishna Karthik Gadiraju
H2: Count-Min Filters **Unity ID:** kgadira **Student ID:** 001042551

Bloom Filters vs Count-Min Filters

Category	Bloom Filter Sketch	Count-Min Sketch
Types of queries answered	**Set membership**: Bloom filters will be able to answer **whether an element has already been seen earlier or not**. There are variants of bloom filters that give approximate count, but they are less efficient than count-min sketches	**Frequency Estimation**: Count-min sketches can be used to obtain **the count of the element**. Queries such as **point queries, range queries, inner product, heavy hitters, histogram, median** etc. can be performed efficiently using Count-min sketches
Space – accuracy tradeoff	Bloom filters are designed to primarily answer queries about whether an element has already been seen earlier or not. A bloom filter **assumes it is OK to return false positives, but it will never return false negatives**. **Accuracy is dependent on the size of the filter and number of hash functions.** If the size is too small, more false negatives will be reported. The **length of the bloom filter must grow linearly** with number of elements inserted in order to **maintain a fixed false positive rate**.	Count-Min sketch is a **sub-linear space** data-structure, and it gives an **approximate solution**, not an exact solution. The space required for Count-Min sketch **proportional to $\frac{1}{\varepsilon}$**. If accuracy is considered in terms of how quickly the values are updated, it is sublinear in the size of the sketch. If accuracy is considered in terms of how accurately it returns the counts of the values, **count-min sketches will only provide an approximate solution, not an exact one**. The **error** in answering a query using CM Sketches is **within parameter $\boldsymbol{\varepsilon}$ with a probability of $\boldsymbol{\delta}$**. The width of bloom filter is dependent on ε and the depth on δ. The width is equal to **ceil(e/ ε) and depth equal to ceil(ln(1/ δ))**

Since sketches are based on summarizing n values into m smaller values, any query operation performed on them would be an approximation. The actions would be defined based on user-defined parameters ε *and* δ. i.e the error in answering a query using a sketch will be within ε and is determined by probability calculated using δ.

Fundamental Drawbacks of Sketch Structures	How Count-min sketch avoids/overcomes them
Comparitively high space Requirement: Sketches use space smaller than the actual data size, but this space used has a **high multiplicative factor** i.e, it is proportional to $\boldsymbol{\Omega(\frac{1}{\varepsilon^2})}$. Since ε usually is small, such as 0.1 or 0.01, its **inverse square will be high**. This high multiplicative factor makes space required for actions such as update processing and function computation require large amount of space and processing time.	**Comparitively low space Requirement:** CM Sketches have a lower multiplicative factor and their space is proportional to $\frac{1}{\varepsilon}$. This value is lesser than that of normal sketch structure. As a result, CMSketch requires lesser space.
Linear Update Time: Update time is **linearly dependent on size of sketch**. Since typical sketches can range from KB to MB, update time will become v high as size increases.	**Sublinear Update Time:** Since update time for CMSketch is **sublinearly proportional to size of sketch**, CM Sketches are faster in updating than other sketches and are not effected as much as the other sketches as the sketch size increases.
Computationally expensive Hash functions: Sketches usually use several hash functions, which have	**Computationally less expensive Hash functions used:**

a **strong requirement for independence** (such as p-wise independence). Computing such functions can be **computationally expensive** based on the type of hardware used.	Since the hash functions used by CMSketches **only need to be pairwise independent** computational expense incurred for using these functions is significantly lowered when compared to other sketches.
Limited aggregation capabilities: Standard sketches are not suitable for aggregation, since they only work for **single, pre-specified aggregate computations**.	**Complex aggregation capability:** CM Sketches can perform several complex aggregations and queries such as point queries, range queries etc.

Three types of queries answered by CM Sketch:
For a CM Sketch data is represented as a(t) = [a_1(t), a_2(t), ……., a_n(t)]. CM Sketch is represented as a two dimensional array of depth d and width w, and depth d: count[1,1]….count[d,w]. There are d hash functions.

Type of query	How CM Sketch Answers it
Point Query: Q(i) -> returns approximation of a_i	The estimation is given by finding all the counts for a_i in each row and then computing the minimum value. $$(1 \le j \le d\,): \hat{a} = min_j(count[j, h_j(i)]$$
Range Query: Q(l,r) -> returns approximation of $\sum_{i=l}^{r} a_i$	Range query is represented as a **combination of several point queries**. Dyadic ranges are used. Each point in [1….n] is a member of $\log_2$n dyadic ranges. The length of each dyadic range is 2^y. By **maintaining a sketch for each of the dyadic ranges**, and **updating the sketches accordingly for each update**, when a Q(l,r) is issued, **at most 2$\log_2$n dyadic ranges are computed**, to cover the entire given range and the equivalent **number of point queries** are called on the sketches. The values returned are all **summed up and returned finally**.
Inner Product Query: Q(**a**,**b**) <- returns approximate value of a.b=$\sum_{i=1}^{n} a_i\, b_i$	Find the dot product for all possible values of a and b, and then compute the minimum of all of them. $$\left(\widehat{a.b}\right) = min_j(\widehat{a.b})_j$$ $$= min_j(\sum_{k=1}^{w} count_a[j,k] * \; count_b[j,k]\,)$$

References:

- Classroom material (Slides)
- An Improved Data Stream Summary: The Count-Min Sketch and its Applications” by Graham Cormode and S. Muthukrishnan