MinHash Sketches:

A Brief Survey

Edith Cohen edith@cohenwang.com

June 14, 2016

1 Definition

Sketches are a very powerful tool in massive data analysis. Operations and queries that are specified with respect to the explicit and often very large subsets, can be processed instead in *sketch space* – that is, quickly (but approximately) from the much smaller sketches.

MinHash sketches (Min-wise sketches) are randomized summary structures of subsets (or equivalently 0/1 vectors). The sketches are *mergeable/composable* which means that adding an element or taking the union of multiple subsets can be performed in sketch space. MinHash sketches support approximate processing of cardinality and similarity queries – The sketches encode information on the cardinality of the set and have the property that the sketches of more similar sets are more similar. MinHash sketches and more generally coordinated samples are a locality sensitive hashing (LSH) scheme.

Historically, MinHash sketches are unweighted *coordinated samples*, studied in the statistics literature [3]. Coordination was important in survey sampling in order to minimize the change in the sample needed when to reflect changes in the population (the LSH property).

The first applications of MinHash sketches for data analysis were for small-state distint counters by Flajolet and Martin [20] and for sketching sets for efficient estimation of their relations by Cohen [6]. The term MinHash (Min-wise) was coined by Broder [4] in the context of a classic application for detecting near-duplicate Web pages.

1.1 Set Operations in Sketch Space

We introduce some necessary notation: The universe of elements is U, its cardinality is $n = |U|$, and $S(X)$ is the sketch of a subset $X \subset U$.

- **Inserting an element:** Given a set X and element $y \in U$, a sketch $S(X \cup \{y\})$ can be computed from $S(X)$ and y.
- **Merging two sets:** For two (possibly overlapping) sets X and Y, we can obtain a sketch of their union $S(X \cup Y)$ from $S(X)$ and $S(Y)$.

Support for insertion makes the sketches suitable for streaming, where elements (potentially with repeated occurrences) are introduced sequentially. Support for merges is important for parallel or distributed processing: We can sketch a data set that has multiple parts by sketching each part and combining the sketches. We can also compute the sketches by partitioning the data into parts, sketching each of the parts concurrently, and finally merging the sketches of the parts to obtain a sketch of the full data set.

1.2 Queries in Sketch Space

From the sketches of subsets we would like to (approximately) answer queries on the original data. More precisely, for a set pf subsets $\{X_i\}$ we are interested in estimating a function $f(\{X_i\})$. To do this, we apply an *estimator* $\hat{f}$ to the respective set of sketches $\{S(X_i)\}$.

We would like our estimators to have certain properties: When estimating nonnegative quantities (such as cardinalities or similarities), we would want the estimator to be non-negative as well. We are often interested in unbiased estimators and always in *admissible* estimators, which are Pareto optimal in terms of variance (variance on one instance can not be improved without increasing variance on another). We also seek good *concentration*, meaning that the probability of error decreases exponentially with the relative error.

We list some very useful queries that are supported by MinHash sketches:

- **Cardinality:** The number of elements in the set $f(X) = |X|$.
- **Similarity:** The Jaccard coefficient $f(X, Y) = |X \cap Y|/|X \cup Y|$, cosine similarity $f(X, Y) = |X \cap Y|/\sqrt{|X||Y|}$, or cardinality of the union $f(X, Y) = |X \cup Y|$.
- **Complex relations:** Cardinality of the union of multiple sets $|\bigcup_i X_i|$; number of elements occurring in at least 2 sets $|\{j \mid \exists i_1 \neq i_2,\ j \in X_{i_1} \cap X_{i_2}\}|$; set differences; etc.
- **Domain queries:** When elements have associated meta-data (age, topic, activity level), we can include this information in the sketch, which becomes a random sample of the set. Including this information allows us to process domain queries, which depend on the mata-data. For example, "the number of Californians in the union of two (or more) sets."

2 Constructions

Several constructions of MinHash sketches had been proposed. There are multiple variants which are optimized for different applications. The common thread is that the elements $x \in U$ of the universe U are assigned random *rank* values $r(x)$ (which are typically produced by a random hash function). The MinHash sketch $S(X)$ of a set X includes order statistics (maximum, minimum, or top/bottom-k values) of the set of ranks $\{r(x) \mid x \in X\}$. Note that when we sketch multiple sets, the same random rank assignment is common to all sketches (we refer to this as *coordination* of the sketches).

Before defining precise structures, we provide some intuition for the power of order statistics. We first consider cardinality estimation. The minimum rank value $\min_{x \in X} r(x)$ is the minimum of $|X|$ independent random variables and therefore, its expectation should be smaller when the cardinality $|X|$ is larger. Thus, the minimum rank carries information on $|X|$. We next consider the sketches of two sets X and Y. Recall that they are computed with respect to the same assignment r. Therefore, the minimum rank values carry information on the similarity of the sets: In particular, when the sets are more similar, their minimum ranks are more likely to be equal. Finally, the minimum rank element of a set is a random sample from the set, and therefore, as such, can support estimation of statistics of the set.

The different MinHash sketch structures vary in the way the rank assignment is used and in the choices of the domain and distribution of the ranks $r(x) \sim D$.

2.1 Structure

MinHash sketches are parameterized by an integer $k \geq 1$, which controls a tradeoff between the size of the sketch representation and the accuracy of approximate query results.

The three most common flavors of MinHash sketches are:

• A *k-mins sketch* [20, 6] includes the smallest rank in each of k independent rank assignments. There are k different rank functions r_i and the sketch $S(X) = (\tau_1, \ldots, \tau_k)$ has $\tau_i = \min_{y \in X} r_i(y)$. When viewed as a sample, it corresponds to sampling k times with replacement.

• A *k-partition sketch*[20, 19, 24], which in the context of cardinality estimation is called *stochastic averaging*, uses a single rank assignment together with a uniform at random mapping of items to k buckets. We use $b : U \to [k]$ for the bucket mapping and r for the rank assignment. The sketch $(\tau_1, \ldots, \tau_k)$ then includes the item with minimum rank in each bucket. That is $\tau_i = \min_{y \in X | b(y)=i} r_i(y)$. If the set is empty, the entry is the typically defined as the supremum of the domain of r.

• A *bottom-k sketch* [6, 4] $\tau_1 < \cdots < \tau_k$ includes the k items with smallest rank in $\{r(y) \mid y \in X\}$. Interpreted as a sample, it corresponds to sampling k elements without replacement. Related/alternative names for the same method are KMV sketch [2], coordinated order samples [3, 27, 25], or Conditional Random Sampling [23].

Note that all three flavors are the same when $k = 1$.

With all three flavors, the sketch represents k random elements of D. When viewed as random samples, MinHash sketches of different subsets X are *coordinated*, since they are generated using the same random rank assignments to the domain U. Coordination implies that similar subsets have similar sketches (a locality sensitive hashing property). It also allows us to support merges and similarity estimation much more effectively.

2.2 Rank distribution

Since we typically use a random hash function $H(x) \sim D$ to generate the ranks, it always suffices to store element identifiers instead of ranks, which means the representation of each

rank value is $\lceil \log_2 n \rceil$ bits and the bit size of the sketch is at most $k \log n$. This representation size is necessary when we want to support domain queries – The sketch of each set should identify the element associated with each included rank, so we can retrieve the meta-data needed to evaluate a selection predicate.

For the applications of estimating cardinalities or pairwise similarities, however, we can work with ranks that are not unique to elements, and in particular, come from a smaller discrete domain. Working with smaller ranks allows us to use sketches of a much smaller size, and also replace the dependence of the sketch on the domain size ($O(\log n)$ per entry) by dependence on the subset sizes. In particular, we can support cardinality estimation and similarity estimation of subsets of size at most m with ranks of size $O(\log \log m)$. Since the k rank values used in the sketch are typically highly correlated, the sketch $S(X)$ can be stored using $O(\log \log m + k)$ bits in expectation ($O(\log \log m + k \log k)$ bits for similarity). This is useful when we maintain sketches of many sets and memory is at a premium, as when collecting traffic statistics in IP routers.

For analysis, it is often convenient to work with continuous ranks, which without loss of generality are $r \sim U[0, 1]$ [6], since there is a monotone (order preserving) transformation from any other continuous distribution. Using ranks of size $O(\log n)$ is equivalent to working with continuous ranks.

In practice, we work with discrete ranks. For example, values restricted to $1/2^i$ for integral $i > 0$ [20], or more generally using a *base* $b > 1$ and using $1/b^i$. This is equivalent to drawing a continuous rank and rounding it down to the largest discrete point of the form $1/b^i$.

2.3 Streaming: Number of updates

Consider now maintaining a MinHash sketch in the streaming model. We maintain a sketch S of the elements X that we had seen until now. When we process a new element y, then if $y \in X$, the sketch is not modified. We can show that the number of times the sketch is modified is in expectation at most $k \ln n$, where $n = |X|$ is the number of distinct elements in the prefix. We provide the argument for bottom-k sketches. It is similar with other flavors. The probability that a new element has a rank value that is smaller than the kth smallest rank in X is the probability that it is in one of the first k positions in a permutation of size $n + 1$. That is, the probability is 1 if $n < k$ and is $k/(n + 1)$ otherwise. Summing over new distinct elements $n = 1, \ldots, |X|$ we obtain $\sum_{i=1}^{n} k/i \leq k \ln n$.

2.4 Inserting an element

We now consider inserting an element y, that is obtaining a sketch $S(X \cup \{y\})$ from $S(X)$ and y. The three sketch flavors have different properties and tradeoffs in terms of insertion costs. We distinguish between insertions that result in an actual update of the sketch and insertions where the sketch is not modified.

- k-mins sketch: we need to generate the rank of y, $r_i(y)$, in each of k different assign-

ments (k hash computations). We can then compare, coordinate-wise, each rank with the respective one in the sketch, taking the minimum of the two values. This means that each insertion, whether the sketch is updated or not, results in $O(k)$ operations.

- bottom-k sketch: We apply our hash function to generate $r(y)$. We then compare $r(y)$ with τ_k. If the sketch contains fewer than k ranks ($|S| < k$ or τ_k is the rank domain supremum) then $r(y)$ is inserted to S.

  Otherwise, the sketch is updated only if $r(y) < \tau_k$. In this case, the largest sketch entry τ_k is discarded and $r(y)$ is inserted to the sketch S. When the sketch is not modified, the operation is $O(1)$. Otherwise, it can be $O(\log k)$.

- k-partition sketch: We apply the hash functions to y to determine the bucket $b(y) \in [k]$ and the rank $r(y)$. To determine if an update is needed, we compare $r(y)$ and $\tau_{b(y)}$. If the latter is empty ($\tau_{b(y)}$ is the domain supremum) or if $r(y) < \tau_{b(y)}$, we assign $\tau_{b(y)} \leftarrow r(y)$.

2.5 Merging

We now considering compute the sketch $S(X \cup Y)$ from the sketches $S(X)$ and $S(Y)$ of two sets X, Y.

For k-mins and k-partition sketches, the sketch of $S(X \cup Y)$ is simply the coordinate-wise minimum $(\min\{\tau_1, \tau_1'\}, \ldots, \min\{\tau_k, \tau_k'\})$ of the sketches $S(X) = (\tau_1, \ldots, \tau_k)$ and $S(Y) = (\tau_1', \ldots, \tau_k')$. For bottom-$k$ sketches, the sketch of $S(X \cup Y)$ includes the k smallest rank values in $S(X) \cup S(Y)$.

3 Estimators

Estimators are most often derived specifically for a given sketch flavor and rank distribution.

Cardinality estimators were pioneered by Flajolet and Martin [20], continuous ranks were considered in [6], and lower bounds presented in [1]. State of the art practical solutions include [19, 8].

Cardinality estimation can be viewed in the context of the theory of point estimation: estimating the parameter of a distribution (the cardinality) from the sketch (the "outcome"). Estimation theory implies that current estimators are optimal (minimize variance) for the sketch [8]. Recently, *Historic Inverse Probability (HIP)* estimators were proposed, which apply with all sketch types, and improve variance by maintaining an approximate count alongside the MinHash sketch [8], which is updated when the sketch is modified.

Estimators for set relations were first considered in [6] (cardinality of union, by computing a sketch of the union and applying a cardinality estimator) and then in [4] (The Jaccard Coefficient, which is the ratio of intersection to union size). The Jaccard Coefficient can be estimated on all sketch flavors (when ranks are not likely to have collisions) by simply looking at the respective ratio in the sketches themselves. In general, many set relations can be estimated from the sketches and state-of-the-art derivation are given in [15, 10].

4 Applications

MinHash sketches are extensively applied to capture sets (images, text documents, logs) similarities, cardinalities, and other relations. We focus here on the earliest applications which popularized the technique for data analysis. The wide range of applications of approximate distinct counters, pioneered by [20], includes statistics collection at IP routers and counting distinct search queries [21]. MinHash sketches were first applied to estimate set relations by [6]: Given a directed graph, and a node v, we can consider the set $R(v)$ of all reachable nodes. It turns out that sketches $S(R(v))$ for all nodes v in a graph can be computed very efficiently, in nearly linear time. The approach naturally extends to sketching neighborhoods in a graph. The sketches of nodes support efficient estimation of important graph properties, such as the distance distribution, node similarities (compare their relations to other nodes) and influence of a set of nodes [12, 13]. Other very early applications are event tracking in multiple threads for the purpose of rollback recovery (maintain for each thread a sketch of all dependent events) [18] and optimizing the multiplication order plan of sparse matrices (sketches are used to estimate in near-linear time the sparsity of subproducts and the number of operations needed to compute them before performing a computation-heavy multiplication) [7]. In the early days of the Web, similarity estimation using sketches was applied to identify near-duplicate Web pages by sketching "shingles" that are consecutive lists or words [4]. This particular application hugely popularized MinHash sketches.

5 Extensions

Weighted elements MinHash sketches are summaries of sets or of 0/1 vectors. In many applications, each element $x \in U$ has a different intrinsic nonnegative weight $w(x) > 0$, and we are interested in "weighted" queries such as weighted sums $\sum_{x \in X} w(x)$ and weighted similarity $\sum_{x \in X \cap Y} w(x) / \sum_{x \in X \cup Y} w(x)$.

To obtain good approximation with weighted elements we need to use sketches so that the inclusion probability of an element increases with its weight. Whereas MinHash sketches are coordinated uniform samples, the weighted version are coordinated weighted samples. Such weighted samples are obtained using ranks which are drawn from a distribution that depend on the weights $w(y)$ [26, 27, 25, 6, 14, 15].

Multiple weights to an element A more complex model, that is suitable for many datasets, allow elements x to have a different nonnegative weight $w_A(x)$ in each set A. For example, sets can corresponds to days, elements to source-destination pairs, and weights to the amount of traffic. Queries over such data may aggregate over elements some function of the tuple of weights the element assumes across "sets." For example, the maximum, minimum, median, or the difference of the values or combinations. Some example similarity metrics are the 2-norm distance $\sum_x ||w_A(x) - w_B(x)||^2$ or weighted Jaccard $\frac{\sum_x \min\{w_A(x), w_B(x)\}}{\sum_x \max\{w_A(x), w_B(x)\}}$. Coordinated weighted samples can be computed in the same manner for each set. We point the reader to [17, 16, 10, 9] for the derivation of optimal estimators.

All-distances sketches (ADS) All-distances sketches [6, 11] are a generalization of MinHash sketches. They can be viewed as the union of the MinHash sketch for all prefixes of an ordered sets. Some important applications include streaming and sketching neighborhoods in graphs.

Hash functions We assumed here availability of truly random hash function. In practice, observed performance is consistent with this assumption. For completeness, we mention that the amount of independence needed was formally studied using *min-wise independent* hash functions [5, 22].

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. System Sci.*, 58:137–147, 1999.

[2] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *RANDOM*. ACM, 2002.

[3] K. R. W. Brewer, L. J. Early, and S. F. Joyce. Selecting several samples from a single population. *Australian Journal of Statistics*, 14(3):231–239, 1972.

[4] A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences*, pages 21–29. IEEE, 1997.

[5] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

[6] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. System Sci.*, 55:441–453, 1997.

[7] E. Cohen. Structure prediction and computation of sparse matrix products. *J. Combinatorial Optimization*, 2:307–332, 1999.

[8] E. Cohen. All-distances sketches, revisited: HIP estimators for massive graphs analysis. In *PODS*. ACM, 2014.

[9] E. Cohen. Distance queries from sampled data: Accurate and efficient. In *KDD*. ACM, 2014. full version: `http://arxiv.org/abs/1203.4903`.

[10] E. Cohen. Estimation for monotone sampling: Competitiveness and customization. In *PODC*. ACM, 2014. full version `http://arxiv.org/abs/1212.0243`.

[11] E. Cohen. All-distances sketches, revisited: HIP estimators for massive graphs analysis. *TKDE*, 2015.

[12] E. Cohen, D. Delling, F. Fuchs, A. Goldberg, M. Goldszmidt, and R. Werneck. Scalable similarity estimation in social networks: Closeness, node labels, and random edge lengths. In *COSN*. ACM, 2013.

[13] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *CIKM*. ACM, 2014.

[14] E. Cohen and H. Kaplan. Summarizing data using bottom-k sketches. In *ACM PODC*, 2007.

[15] E. Cohen and H. Kaplan. Leveraging discarded samples for tighter estimation of multiple-set aggregates. In *ACM SIGMETRICS*, 2009.

[16] E. Cohen and H. Kaplan. What you can do with coordinated samples. In *The 17th. International Workshop on Randomization and Computation (RANDOM)*, 2013. full version: `http://arxiv.org/abs/1206.5637`.

[17] E. Cohen, H. Kaplan, and S. Sen. Coordinated weighted sampling for estimating aggregates over multiple weight assignments. *VLDB*, 2(1–2), 2009. full: `http://arxiv.org/abs/0906.4560`.

[18] E. Cohen, Y.-M. Wang, and G. Suri. When piecewise determinism is almost true. In *Proc. Pacific Rim International Symposium on Fault-Tolerant Systems*, pages 66–71, December 1995.

[19] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *Analysis of Algorithms (AofA)*. DMTCS, 2007.

[20] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. System Sci.*, 31:182–209, 1985.

[21] S. Heule, M. Nunkesser, and A. Hall. HyperLogLog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *EDBT*, 2013.

[22] P. Indyk. A small approximately min-wise independent family of hash functions. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*. ACM-SIAM, 1999.

[23] P. Li, , K. W. Church, and T. Hastie. One sketch for all: Theory and application of conditional random sampling. In *NIPS*, 2008.

[24] P. Li, A. B. Owen, and C-H Zhang. One permutation hashing. In *NIPS*, 2012.

[25] E. Ohlsson. Sequential poisson sampling. *J. Official Statistics*, 14(2):149–162, 1998.

[26] B. Rosén. Asymptotic theory for successive sampling with varying probabilities without replacement, I. *The Annals of Mathematical Statistics*, 43(2):373–397, 1972.

[27] B. Rosén. Asymptotic theory for order sampling. *J. Statistical Planning and Inference*, 62(2):135–158, 1997.