Kevin's Last Sketch: Frequent Distinct Tuples

Kevin Lang, Lee Rhodes

Abstract

We develop a streaming algorithm (or sketch) that addresses the following problem: In a stream of key-value pairs, identify the keys that have the largest number of distinct values. Our solution also enables this problem to be solved with multiple dimensional data. We call this the Frequent Distinct Tuples (FDT) Sketch and do not believe that this solution has been published. This sketch is fully mergeable and can be used in the context of set expressions. This sketch has already been implemented in our open source library [15] and is production ready.

1 Introduction

In the data sciences, sketches (a.k.a., streaming algorithms) are small, stateful programs that process massive data as a stream and can provide approximate answers, with mathematical guarantees, to computationally difficult queries orders-of-magnitude faster than traditional, exact methods.

DataSketches.github.io [15], developed at Verizon Media, is an open source, high-performance library of sketching algorithms that have been carefully crafted to meet rigorous standards of quality and performance and provide capabilities required for large-scale production systems that must process and analyze massive data. This library is used in a number of the data processing platforms at Verizon Media as well as at a number of outside companies.

2 The Frequent Distincts Challenge

The reader may already be familiar with the problem of identifying the most frequent items in a stream. For example, the owners of a web site that sells songs would likely be very interested in understanding which song titles are the most frequently requested during the last day, hour or minute. Note that this is inherently a *distinct* type query as we do not want any duplicate song titles in the list that is returned. The streaming sketch solution to this problem is the well-known Frequent Items Sketch which has a rich literature [1,4,5,9,12,13]. It is also called a Heavy-Hitters Sketch, which has a parallel literature [2,3,6,7,10,11,14,16,17]. An implementation of the Frequent Items algorithm described in our IMC paper [1] is already offered in our DataSketches library [15].

However, it is often the case that a simple query for the most frequent items from a stream will not satisfy user needs. Data is often fragmented into multiple dimensions such as demographics, geography, time or many other attributes. We would like to be able to answer queries such as:

- What are the geographic regions that are requesting the largest number of songs?
- Which songs are being requested by the largest number of geographic regions?
- What user age-groups are requesting the largest number of songs?
- What songs are being requested by the largest number of age-groups?

Note that these are also *distinct* type queries but now they involve querying over multiple dimensions.

Multi-dimensional distinct queries are very common with databases, but when the computational cost of the query is high and the data stream becomes very large, responding to such queries quickly becomes problematic. This is where streaming algorithms, such as sketches, should be able to help. Unfortunately, most of the frequent-items and heavy-hitter algorithms commonly discussed in the literature are not suited for this kind of query, nor can they be easily adapted or extended to address this type of query.

For example, a simplistic attempt to embed a distinct counting sketch, such as HLL, inside a Frequent-Items sketch by replacing the item counters with HLL sketches produces horrible errors! The retained HLL sketches in the Frequent-Items sketch can move in and out of the sketch as the stream is being processed destroying the integrity of the retained HLL sketches, which require seeing the entire stream.

3 Review: Theta and Tuple Sketches

The Theta Sketch [8, 15] has been in our library since about 2013 and is very heavily used inside Verizon Media as well as outside the company. The details of exactly how the Theta Sketch works are beyond the scope of this paper, but it is well documented elsewhere [8, 15]. Nevertheless, one of the key concepts to understand for our purposes here is that after a stream has been processed by the sketch its internal data structure can be represented by the simple model in Figure 1.

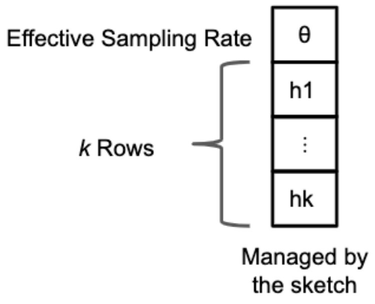

Figure 1: Simplified Theta Sketch Data Structure

The data structure consists of an array of k 64-bit hash values $h_1, ..., h_k$, where k is specified by the user and controls the size and accuracy of the sketch. The input to a Theta Sketch is a stream of $items$, where each item can occur many times in the stream. A stream with duplicates is called a $multiset$. One of the key functions of the sketch is to deduplicate the incoming items. However, to simplify our discussion we will ignore this part of the processing and consider the input stream to have u distinct items. For practical purposes, each hash value retained by the sketch can be considered to have a one-to-one correspondence with a distinct input item presented to the sketch. A key property of the set of k hash values retained by the sketch is that they represent a uniform random sample of the domain of u distinct items presented to the sketch.

The value $theta$, θ, is the effective sampling rate for the sketch in its current state. The sketch can only hold a maximum of k hash values and, given an input stream of size u distinct values, the value of theta is approximately k/u. When u is not known (the usual case), the unbiased estimate of u is simply $\hat{u} = k/\theta$.

The Tuple Sketch [15] is an extension of the Theta Sketch and was invented at Yahoo. Instead of retaining a single hash value for a distinct item and rejecting all duplicates of that same item, the Tuple Sketch can retain an associated row of attributes along with each hash value as shown in Figure 2. The input fed to the sketch is now a stream of $tuples$: $\{item, a_1, a_2, ..., a_n\}$. Normally, only the $item$ component of the tuple is fed to the hash function that generates the hash value to be considered by the sketch. The other attributes just go along for the ride. The attributes can be any object, but typically are strings, counters, bit-maps, or lists. Upon receiving a duplicate $item$ of a tuple in the stream of tuples, instead of rejecting the tuple, as the Theta Sketch does, the Tuple Sketch can be instructed to update the n attributes associated with the hash of the $item$ based on a set of rules defined by the user.

Used in this way, the properties of the sketch with respect to the retained $rows$ is similar to the Theta Sketch's management of retained items. The hash column of the Tuple Sketch represents a uniform random sample of the domain of distinct $items$ obtained from the input tuples. The attributes retained in the sketch represent an aggregation or summary of all the attributes associated with the duplicate $items$ of the incoming tuples. The unbiased estimate of the size of u distinct items from which the sketch was created is still k/θ.

One of the powerful advantages of the Tuple Sketch is the concept of subsetting [8]. Once the sketch has been created and has processed a stream of data, we can treat the matrix of k rows by n columns as a rectangular structure that can be queried like a simple database table.

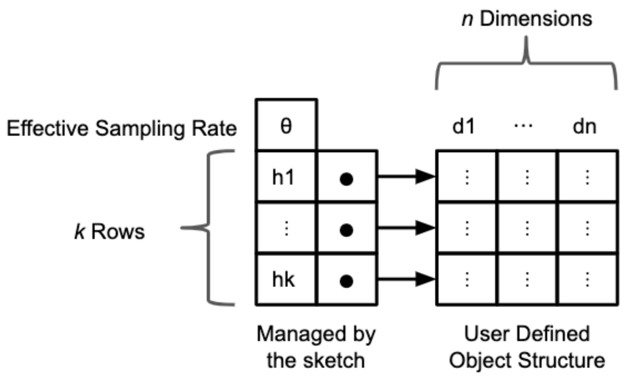

Figure 2: Simplified Tuple Sketch Data Structure

For example, we can select rows where the attributes satisfy some criteria. Let the number of rows that satisfy the criteria be r. The unbiased estimate of the size of the distinct population from the original domain of distinct $items$ associated with the r rows is simply computed as r/θ. Leveraging this subsetting capability of Tuple Sketches is already being used in production at Verizon Media.

4 Kevin's Key Insight

As part of Kevin's last project before his untimely passing, his key insight was that we could use the Tuple Sketch structure to also solve the Frequent Distincts problem. Instead of viewing the input to the Tuple Sketch as a tuple consisting of a key $item$ with associated attributes, we can view the input tuple as an n dimensional vector: $\{d_1, d_2, ..., d_n\}$. In addition, instead of choosing only one element of the vector to feed the hash function, we feed the entire tuple into the hash function. The hash column of Figure 2 now represents a uniform random sample of $tuples$ from the input domain of u distinct tuples. The columns associated with each hash value contain copies of each of the dimensional elements of the input tuple. If the input tuple has n dimensions, then the Tuple Sketch has a hash column and n associated columns. If a duplicate tuple is detected, it is rejected as in a Theta Sketch. There is no aggregation. If the tuple is not a duplicate it is added to the sketch based on the selection rules of the sketch.

The analysis occurs in the post-processing phase after the sketch has processed all the input data. To illustrate how this works let us start with a simple example.

Suppose our problem is to identify the IP addresses that are associated with the largest number of unique user IDs. Our input tuples are of the form $\{IP, ID\}$. If an input tuple is selected by the sketch the IP address is mapped to d_1 and the ID is mapped to d_2 of the associated matrix. After processing all the input data, the hash column is not required[1] for this analysis as its job is to assist with the tasks of deduplication and tuple selection. Since its job is done, we may ignore it.

We can now treat the Tuple Sketch's associated columns as a simple database table. We define the IP column as the *key* dimension and the ID column as the *free* dimension. We then perform a group-by on the key dimension. The resulting data structure might look something like Figure 3(a) where all rows of each IP address are grouped together.

Since the contents of the rows are a uniform random sample, the number of rows of group IP_1, when transformed by theta, correspond to the number of unique $\{IP_1, ID*\}$ combinations that actually appeared in the input stream. Inverse sorting the groups by their size reveals the IP addresses with the largest number of user IDs.

[1]The hash column will still be required if this sketch is further updated or participates in merging with other sketches

We can also obtain an unbiased estimate of the distinct counts for each group in the input domain. Let r_1 be the number of rows of IP_1 and v_1 represent the number of unique tuples of $\{IP_1, ID*\}$ in the input domain. The unbiased estimate of v_1 is r_1/θ. Similarly, the unbiased estimate of v_2 is r_2/θ.

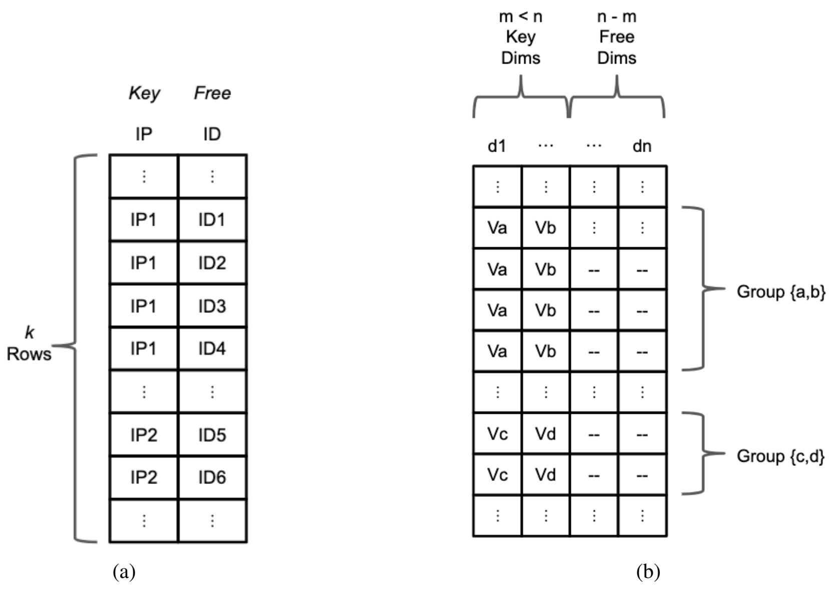

Figure 3:
(a) Tuple Sketch data structure for IP,ID problem and
(b) Tuple Sketch data structure with multiple dimensions

This concept can be extended to multiple dimensions as shown in Figure 3(b) where m is the number of key dimensions.

Without reprocessing the input stream, we may reverse the assignment of which column is the key and which column is the free column. Performing a group-by on the key and sorting the groups by size allows us to discover the user *IDs* with the largest number of *IP* addresses! The unbiased estimates of the actual counts from the input domain can be obtained as before.

The basic post-processing rule is that the user defines $m < n$ dimensions as the key dimensions, which leaves $n - m$ dimensions as free dimensions. In the implementation of the FDT Sketch in the library, the selection of which dimensions are the key dimensions is an input parameter to the post-processing phase. Performing a group-by on the key dimensions and sorting by the resulting group sizes reveals the keys that have the largest number of combinations of the free dimensions.

5 Accuracy

The error distribution of Theta and Tuple Sketches is asymptotically Gaussian, with large enough k (the sketch capacity) and large enough u. With the Gaussian assumption, the Relative Standard Error (RSE), which is the standard deviation normalized by the distinct count, can be shown [8] to be $RSE \le 1/\sqrt{k-1}$. RSE. The upper and lower bounds on the estimate with a confidence of 68% can be computed as

$$UB = estimate + RSE,$$
$$LB = estimate - RSE.$$

Because Theta and Tuple Sketches allow set intersection and difference operations and the Tuple Sketch enables subsetting of the retained entries of the sketch, small sample counts can occur frequently. As a result, in our library implementation we compute the error bounds using the more accurate binomial distribution instead of the simplified Gaussian approximation. Nonetheless, independent of the estimation algorithm, the fewer the number of rows in the group the wider will be the distribution of error around the estimate.

One of the concepts from the Frequent Items Sketch is the *noise floor*, the point below which accurate estimates of whether an item is actually a *heavy hitter* or *frequent item* cannot be made. Similarly, for this FDT Sketch we would like to establish a threshold, T, which is a fraction, less than one, of the input unique domain of u. If the relative size of a group r/k is less than T, the sketch cannot provide a guarantee of the estimation accuracy, or whether the size of the group is significant or not.

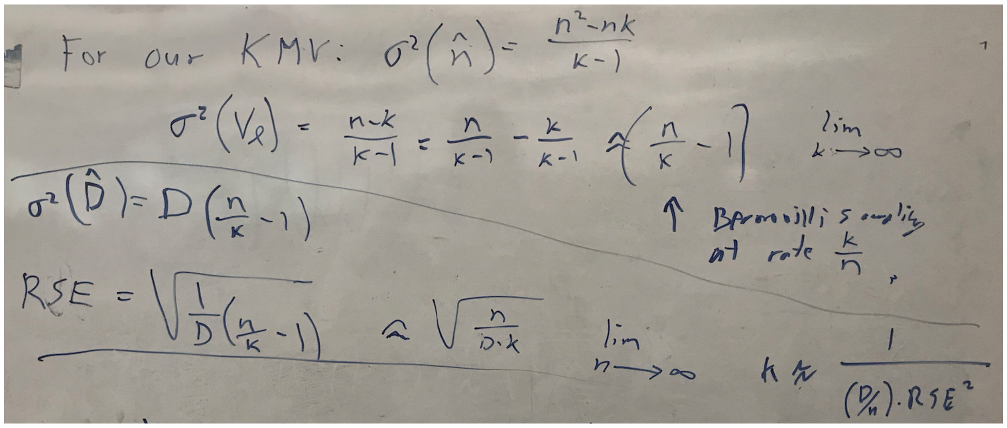
For our KMV: $\sigma^2(\hat{n}) = \frac{n^2-nk}{k-1}$
$\sigma^2(V_\ell) = \frac{n-k}{k-1} = \frac{n}{k-1} - \frac{k}{k-1} \approx \left(\frac{n}{k} - 1\right)$
$\lim_{k\to\infty}$
$\sigma^2(\hat{D}) = D\left(\frac{n}{k} - 1\right)$
↑ Bernoulli sampling at rate $\frac{k}{n}$
$RSE = \sqrt{\frac{1}{D}\left(\frac{n}{k}-1\right)} \approx \sqrt{\frac{n}{D\cdot k}}$
$\lim_{n\to\infty}$
$k \approx \frac{1}{(D/n)\cdot RSE^2}$

Figure 4: Kevin's whiteboard notes

A week or so before Kevin's death, he explained his idea on how to solve the Frequent Distincts problem. The only record we have of this conversation is this whiteboard image (Figure 4) where he demonstrated (in a hand-wavy way) that the scheme we have been explaining above can be proved to work with well understood accuracy. The darker pen scribbles are notes Lee wrote as Kevin was describing his logic.

At the top Kevin establishes that the variance follows the normal binomial distribution due to a Bernoulli sampling process just like our other sketches. In the last line he defines a variable D, which is the number of tuples in the raw domain of distinct tuples that correspond to the subsampled rows in the sketch. Thus, $\widehat{D} \approx r/\theta$. Since $\hat{n} \approx k/\theta$, $T = \widehat{D/n} \approx r/k$ is a threshold that varies from zero to one and represents the fraction of the raw domain of distinct tuples that we want to establish as the noise floor. For groups that occupy a larger fraction of the domain, we want to make sure that the sketch can capture the group and that the accuracy of the size estimate of that group will be satisfactory. At that threshold, $RSE \approx \sqrt{n/(Dk)} \approx 1/\sqrt{Tk}$ which reduces to $1/\sqrt{r}$ for thresholds less than one and $1/\sqrt{k}$ when the threshold is one. This closely matches our earlier $1/\sqrt{k-1}$ for large enough k.

We can also solve for the required k given a desired RSE and T as $k \approx 1/(T * RSE^2)$. This is implemented in the library as an alternative way of configuring the sketch.

6 Characterizing the Error

In order to characterize the error performance we created a test that generated a power-law distribution of tuples over several orders of magnitude with about 850 thousand tuples of 4 dimensions between two points on the log-log axis of frequency and number of tuples at that frequency. The two points were $\{16384, 1\}$ and $\{1, 16384\}$. The test algorithm generated 16 points per octave of intermediate generation points along that line. This resulted in a power-law slope of -1. The sketch threshold was set at .01 with a target RSE of .05. This resulted in a sketch $K = 65536$ (because k must be a power of 2). In post processing, 3 of the dimensions were defined as the key and the remaining dimension was free. This entire characterization suite was then executed over many trials.

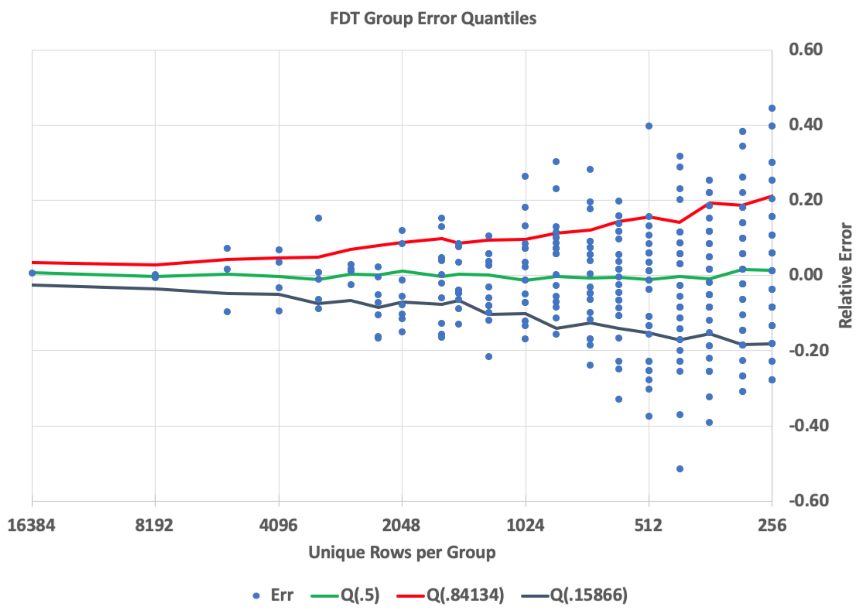

Figure 5: RSE performance vs Group Size

In Figure 5 the single group with the largest row count (16384) is the dot at the left of the graph. As the groups get smaller the number of groups at that size become more frequent. At a group row count of 1 there were 16384 groups generated but only the largest groups down to a group size of 256 were plotted to simplify the plot. The plot illustrates how the error bounds of the distribution gradually increase as the group size gets smaller, which is what we expect. The red and black curves represent +/- one standard deviation and the green curve is the median.

There are a few caveats with this sketch. The first is that if the frequency distribution is extremely steep, the number of groups captured could be only one, even though other groups with a size above the threshold might exist. The second is that if the distribution is very flat all of the groups could have sizes below the threshold, and the results could be meaningless. The last is a word of caution: do not use too many dimensions. Not only will the sketch grow quite large, but with many dimensions the data will likely be highly fragmented and the largest groups may fall below the threshold.

7 Remembering Kevin

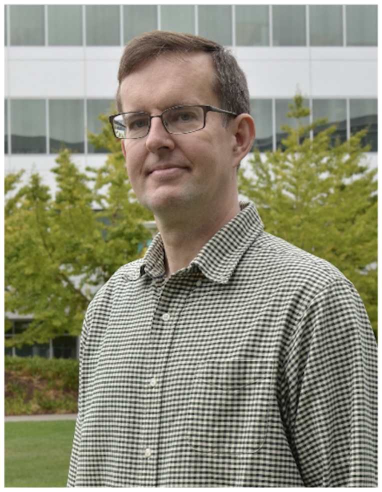

Figure 6: Kevin J. Lang

Before Kevin Lang tragically passed away in April, 2019, he came up with the solution for the FDT sketch presented in this paper. Kevin's solution was elegant because it not only addressed the problem, but took advantage of a number of technologies that we already had implemented in our library such as the Tuple Sketch, binomial bounds estimation, and more. As a result, after he had explained the solution, we were able to implement and test the new sketch relatively quickly.

Our DataSketches project owes a great deal to Kevin as he has contributed enormously over the past 8 years to this project. We could not have achieved what we have without him and we will miss him.

We would like to dedicate this paper to the memory of Kevin and his outstanding contributions to the science and engineering of sketching algorithms.

References

[1] Daniel Anderson, Pryce Bevan, Kevin J. Lang, Edo Liberty, Lee Rhodes, and Justin Thaler. A high-performance algorithm for identifying frequent items in data streams. In *Internet Measurement Conference*, 2017. https://conferences.sigcomm.org/imc/2017/papers/imc17-final255.pdf. [p. 1]

[2] Ran Ben Basat, Gil Einziger, Roy Friedman, Marcelo Caggiani Luizelli, and Erez Waisbard. Constant time updates in hierarchical heavy hitters. *arXiv preprint arXiv:1707.06778*, 2017. To Appear in ACM SIGCOMM 2017. [p. 1]

[3] Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J Strauss. Space-optimal heavy hitters with strong error bounds. *ACM Transactions on Database Systems (TODS)*, 35(4):26, 2010. [p. 1]

[4] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Automata, languages and programming*, pages 784–784, 2002. [p. 1]

[5] Graham Cormode and Marios Hadjieleftheriou. Finding the frequent items in streams of data. *Communications of the ACM*, 52(10):97–105, 2009. [p. 1]

[6] Graham Cormode, Flip Korn, S Muthukrishnan, and Divesh Srivastava. Finding hierarchical heavy hitters in data streams. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 464–475. VLDB Endowment, 2003. [p. 1]

[7] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. Diamond in the rough: Finding hierarchical heavy hitters in multi-dimensional data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pages 155–166, 2004. [p. 1]

[8] Anirban Dasgupta, Kevin J. Lang, Lee Rhodes, and Justin Thaler. A framework for estimating stream expression cardinalities. In *Proceedings of* ICDT, pages 6:1–6:17, 2016. [pp. 2, 3, 5]

[9] Lukasz Golab, David DeHaan, Erik D Demaine, Alejandro Lopez-Ortiz, and J Ian Munro. Identifying frequent items in sliding windows over on-line packet streams. In *Proceedings of IMC*, pages 173–178. ACM, 2003. [p. 1]

[10] John Hershberger, Nisheeth Shrivastava, Subhash Suri, and Csaba D Tóth. Space complexity of hierarchical heavy hitters in multi-dimensional data streams. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 338–347. ACM, 2005. [p. 1]

[11] Yuan Lin and Hongyan Liu. Separator: sifting hierarchical heavy hitters accurately from data streams. *Advanced Data Mining and Applications*, pages 170–182, 2007. [p. 1]

[12] Amit Manjhi, Vladislav Shkapenyuk, Kedar Dhamdhere, and Christopher Olston. Finding (recently) frequent items in distributed data streams. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 767–778. IEEE, 2005. [p. 1]

[13] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of* ICDT, pages 398–412, 2005. [p. 1]

[14] Michael Mitzenmacher, Thomas Steinke, and Justin Thaler. Hierarchical heavy hitters with the space saving algorithm. In *Proceedings of ALENEX*, pages 160–174, 2012. [p. 1]

[15] Lee Rhodes, Kevin Lang, Alexander Saydakov, Justin Thaler, Edo Liberty, and Jon Malkin. DataSketches: A Java, C++ and Python software library for streaming data algorithms. Apache License, Version 2.0, 2015. `https://datasketches.github.io`. [pp. 1, 2]

[16] Patrick Truong and Fabrice Guillemin. Identification of heavyweight address prefix pairs in ip traffic. In *Teletraffic Congress, 2009. ITC 21 2009. 21st International*, pages 1–8. IEEE, 2009. [p. 1]

[17] Yin Zhang, Sumeet Singh, Subhabrata Sen, Nick G. Duffield, and Carsten Lund. Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In *Proceedings of the 4th ACM SIGCOMM Internet Measurement Conference, IMC 2004, Taormina, Sicily, Italy, October 25-27, 2004*, pages 101–114, 2004. [p. 1]