Time Adaptive Sketches (Ada-Sketches) for Summarizing Data Streams

Anshumali Shrivastava*
Dept. of Computer Science
Rice University
Houston, TX, USA
anshumali@rice.edu

Arnd Christian König
Microsoft Research
Redmond
WA 98052, USA
chrisko@microsoft.com

Mikhail Bilenko
Microsoft
Redmond
WA 98052, USA
mbilenko@microsoft.com

ABSTRACT

Obtaining frequency information of data streams, in limited space, is a well-recognized problem in literature. A number of recent practical applications (such as those in computational advertising) require temporally-aware solutions: obtaining historical count statistics for both time-points as well as time-ranges. In these scenarios, accuracy of estimates is typically more important for recent instances than for older ones; we call this desirable property *Time Adaptiveness*. With this observation, [20] introduced the HOKUSAI technique based on count-min sketches for estimating the frequency of any given item at any given time. The proposed approach is problematic in practice, as its memory requirements grow linearly with time, and it produces discontinuities in the estimation accuracy. In this work, we describe a new method, *Time-adaptive Sketches*, (ADA-SKETCH), that overcomes these limitations, while extending and providing a strict generalization of several popular sketching algorithms.

The core idea of our method is inspired by the well-known digital Dolby noise reduction procedure that dates back to the 1960s. The theoretical analysis presented could be of independent interest in itself, as it provides clear results for the time-adaptive nature of the errors. An experimental evaluation on real streaming datasets demonstrates the superiority of the described method over Hokusai in estimating point and range queries over time. The method is simple to implement and offers a variety of design choices for future extensions. The simplicity of the procedure and the method's generalization of classic sketching techniques give hope for wide applicability of Ada-sketches in practice.

1. INTRODUCTION AND MOTIVATION

"Scaling Up for High Dimensional Data and High Speed Data Streams" has been recognized one of the top 10 challenging problems in data mining research [25]. Mining big data streams brings three major challenges: *volume, velocity* and *volatility* [18]. Volume and velocity require processing a high volume of data in limited time. Consider the case of click-through probability prediction, a central problem for online advertising, which is estimated to be a $230 billion dollar industry[1]. This industry relies on tracking and responding to the behavior of billions of users, to whom large platforms must serve tens of millions of ad impressions per hour with typical response times under 50ms per request [15]. Moreover, it is essential to rapidly update predictive models as new clicks and non-clicks are observed [21]. Maximizing predictive accuracy demands keeping track of the counts of many combinations of events (such as clicks and impressions broken down by country, IP, advertiser, or ad-type), leading to a blow-up in the number of items under consideration. Volatility, on the other hand, corresponds to a dynamic environment with ever-changing patterns, making the scale of the problem even worse. A major reason for volatility is the drift in the distributions of patterns of interest over time, and constant churn of users, advertisers and content providers [18].

Obtaining frequency information, or statistical patterns of interest, from data streams is an important problem in this context. Count-based features are commonly used in practice for many predictive tasks. For example, features such as counts of number of clicks on "Amazon.com", given specific user characteristics, in the past hour, are common in click-through probability prediction. In particular, there is a need to estimate the count of a given item i (or event, or combinations thereof) during some specified time t, or in some $[t_1, t_2]$ range [23, 20, 16]. Since the number of possible combinations is enormous, we end up with the classical problem of counting under memory and time constraints over streaming data in one pass. Note that in the streaming setting, we only see increments in counts over time, and hence cannot store all items seen, which makes estimation with limited resources a challenging problem.

Typically, items with highest counts, commonly known as *heavy hitters*, are of most interest. However, identifying heavy hitters in streaming data is a non-trivial task with limited memory. There has been a strong line of work in computing limited-memory summaries, popularly called *sketches* [13] of data streams, allowing approximate estimation of the counts [12, 1, 14, 6, 22, 10, 2], even over dynamic temporal streams [8, 17]. Sketches allow high-accuracy estimation of heavy hitters. Notable among them is the Count-Min Sketch [7], an accurate technique widely popular due to its simplicity and parallelizability.

*Work done while visiting Microsoft.

[1]Source: http://www.emarketer.com/, Dec. 2014

SIGMOD'16, June 26-July 01, 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2882903.2882946

With more than 30 years of active research, existing data stream sketching techniques can reasonably address the challenges of volume and velocity. However, the volatility aspect of the problem has received little attention. With changing patterns, drift in distributions over time is commonly observed in data streams, sometimes with high variance in so-called bursty data streams. In a drifting distribution, as was pointed out in [18], "old data is of limited use, even if it could be saved and processed again later". This is because, in most applications, recent patterns have more predictive power compared to patterns in the past. For instance, most recent variations in credit history are much stronger indicators of a person's ability to make loan payments compared to variations in credit history from distant past.

Thus, given a limited resource budget, if we have to trade off errors effectively, we need to be more accurate in estimating the recent statistics (or counts) compared to the older ones [20]. In particular, we need sketches that are "time adaptive". Classical sketching techniques are oblivious to time, and hence do not provide this smarter trade-off. The focus of this paper is developing sketches for accurate estimation that provide such desired time adaptability.

An obviously tempting and reasonable attempt to achieve time adaptability is to allocate resources to different time periods disproportionately. More specifically, if we create large-size sketches to handle recent events, and use small-size sketches for older intervals, we can immediately achieve time adaptability, as sketch size directly translates to accuracy for reasonable algorithms. The key challenge lies in designing algorithms that dynamically adjust per-interval sketch sizes and reallocate them in streaming fashion as time passes. Addressing this problem was the key insight of the recently proposed HOKUSAI algorithm [20] for item aggregation.

An open problem for HOKUSAI algorithm lies in its allocation of a fixed-size sketch for each time point t, regardless of the number of distinct items contained in it. Let us consider a "bursty" data distribution, where a million items arrived at a single time point t', with many fewer arriving at subsequent times. In order to represent this data with high accuracy, HOKUSAI must allocate sketches with $O(10^6)$ cells not only to the time point t', but also to every time point $t > t'$, even though most of the space in sketches for subsequent times will be wasted.

As described in Section 2.6, discontinuities in the degradation of HOKUSAI estimates with time is a significant problem. Such issues are inherent to any approach that associates a sketch with every time instance t. As an alternative, we provide a *time-adaptive* solution based on a very different approach motivated by the noise reduction literature, which avoids the limitations of HOKUSAI, while yielding significantly higher accuracy at the same time.

Our Contributions: We present *Time-adaptive Sketches* (ADA-SKETCH) for estimating the frequency of any given item i at given time t, or over a time interval $[t_1, t_2]$, using limited space. Ada-sketches provide a strict generalization of several well-studied sketching algorithms in the data streams literature for obtaining temporal estimates. The proposed generalization, while retaining the counts of heavy hitters with good accuracy, also provides provable time-adaptive estimation guarantees. In particular, the estimate of the count of item i during time t is more accurate compared to the estimate of same item i during time t' for all $t' \leq t$. The key idea in our proposed technique is the use of pre-emphasis and de-emphasis transformations analogous to those used in the popular Digital Dolby systems for noise reduction.

We demonstrate that the method can provide a variety of desired time-adaptive estimates by appropriately engineering the choice of pre-emphasis and de-emphasis transformations. More formally, the problem of finding the right sketch for the application at hand reduces to solving a given set of simultaneous equations and finding the appropriate pre-emphasis and de-emphasis transformations. This reduction could be of independent theoretical interest in itself.

We provide theoretical analysis and quantifications of error bounds with Ada-sketches for both point and range queries. The analysis confirms the adaptive nature of errors with time. An experimental evaluation on two real streaming datasets supports the theoretical claims: estimation accuracy shows time-adaptive behavior for both point and range queries, as expected from our analysis. Experimental comparisons also clearly show the advantages of the described method over the previously proposed HOKUSAI [20] temporally-aware sketches.

Ada-sketches simultaneously address and allow trading off the three key challenges of data stream algorithms: *volume, velocity* and *volatility*. Holistically addressing these challenges has been called out as an important yet understudied research direction [18], and we hope that this work will lead to further exciting developments in the area.

2. REVIEW

2.1 Notation

We assume the usual turnstile streaming model with positive updates [22]. The set of all items will be denoted by $I = \{1, 2, ..., N\}$. Time starts from $t = 0$, and the current time will be denoted by $T > 0$. The total increment to item i during time instance (or interval) t will be denoted by c_i^t, which is an aggregation of many streaming updates arriving at t. The results in this paper, without loss of any generality and for notational convenience, will use c_i^t to refer to the current update. We consider t to be a discrete timestamp or time interval of some fixed length and appropriate granularity. $C^t = \sum_{i \in I} c_i^t$ denotes the total increment of all items during time t. We define stream moments as

$$\mathcal{M}^T = \sum_{t=0}^{T} C^t; \qquad \mathcal{M}_2^T = \sum_{t=0}^{T} (C^t)^2$$

2.2 Count-Min Sketch (CMS)

The *Count-Min Sketch* (CMS) [7] algorithm is a generalization of Bloom filters [4] that is a widely popular in practice for estimating counts of items over data streams. CMS is a data structure with a two-dimensional array of counter cells M of width w and depth d, shown in Figure 1. It is accessed via d pairwise-independent hash functions $h_1, h_2, ..., h_d : \{1, 2, ..., N\} \mapsto \{1, 2, ..., w\}$. Each counter is initialized with zero, and every update c_i^t is added for all d rows to counters $M(j, h_j(i))$, where $j = \{1, 2, ..., d\}$. A query for the count of item i reports the minimum of the corresponding d counters i.e., $\min_{j \in \{1,2,...,d\}} M(j, h_j(i))$. This simple algorithm has strong error guarantees and is very accurate for estimating heavy hitters over entire streams,

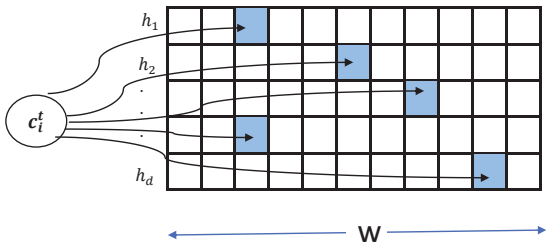

Figure 1: Count-Min Sketch Data Structure

with its simplicity and easy parallelization contributing to its wide adoption.

2.3 Lossy Counting (LC)

Lossy Counting [19] is another popular algorithm for counting frequent items over data streams. The algorithm keeps a fixed-size map of items and their counts seen so far. Whenever an update c_i^t arrives, the algorithm increments the map's counter corresponding to item i by c_i^t, if item i is present in the map. If the map does not contains item i, the algorithm allocates memory for i. When the map reaches its capacity, the algorithm simply deletes all items with counts below a specified threshold from the map, creating space for new items. The algorithm is motivated by the expectation that frequent items will survive periodic deletions. Despite the method's simplicity, it has near optimal theoretical guarantees for estimating heavy hitters with limited space [19].

2.4 The Need for Time Adaptive Sketches

In most applications that involve temporal data, most recent trends tend to be most informative for predictive purposes [18, 20]. For instance, in clickthrough predictions, recent click history provides best estimates of future clicks compared to logs from the more distant past. At the same time, old heavy hitters (or old but significant trends) also must be preserved, especially in the absence of more recent frequent occurrences of an item. With limited space, we can tolerate some error on counts of items in distant history, but would like to obtain highly accurate counts for recently seen items. While existing sketches, such as CMS and LC, are very good in retaining accurate information about heavy hitters, they do not take into account this temporal disparity in desired accuracy. It is possible to use straightforward extensions (discussed later) to existing sketching techniques to encode the temporal information; however, this results in error guarantees that are *independent* of time, and not adaptive to item recency. In this paper, we present adaptive variants of these sketches that achieve the desired time adaptability. To the best of our knowledge, only one technique has been designed specifically to capture frequency information over time, which we describe in the following:

2.5 Existing Method: Hokusai

In [20], the authors presented two modifications of CMS for aggregating information over temporal data streams. The first algorithm, referred to as *time aggregation*, aggregates information only in dyadic time intervals, i.e., intervals of the form $[t2^k + 1, ..., (t + 1)2^k]$. However, this method does not provide (*item*, *time*) estimation for individual items. Moreover, in order to retain efficiency, the method incurs deterministic over-counting which can make the errors unpredictable. Hence, the focus of our comparison is on the *item aggregation* algorithms (Algorithm 3 in [20]) proposed for constructing time-adaptive sketches. This method provides general (*item*, *time*) estimation. Throughout this paper we will refer the item aggregation algorithms of [20] as the Hokusai (name of the paper) algorithm.

Hokusai uses a set of Count-Min sketches for different time intervals, to estimate the counts of any item for a given time or interval. To adapt the error rate temporally in limited space, the algorithm uses larger sketches for recent intervals and sketches of smaller size for older intervals, as illustrated in Figure 2. In a streaming setting, this setup creates additional overhead of tracking time intervals and shrinking the sketch size for older ones as time progresses. To efficiently achieve such sketch shrinking, [20] utilize an elegant insight: the size of count-min sketches can be halved simply by adding one half of the sketch to the other half, as shown in Algorithm 1. This operation also requires halving the hash function ranges, which can be easily accomplished using modulo 2 operation. Thus, it is possible to maintain sketches of varying sizes as shown in Figure 2 and dynamically adjust them as time progresses.

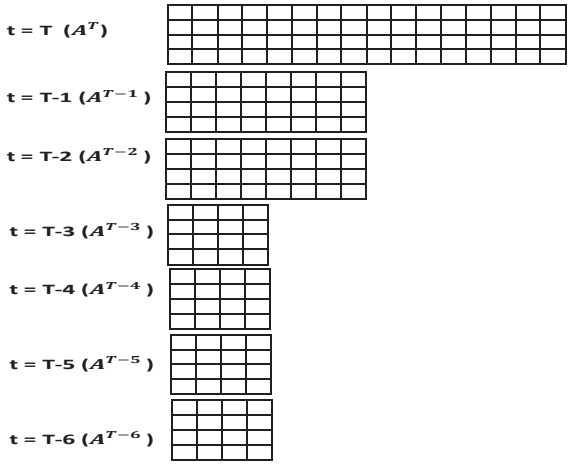

Figure 2: Hokusai

Algorithm 1 *Hokusai [20]*

```
t ← 0
while data arrives do
    Receive data in Sketch M for this t (Modify M)
    t ← t + 1
    A^t = M (Copy)
    for k = 1 to ⌊log_2 t⌋ do
        ∀ i, j in sketch A^{t-2^k}
        A^{t-2^k}[i, j] = A^{t-2^k}[i, j] + A^{t-2^k}[i, j + 2^{m-k}]
        Shrink to size 2^{m-k}
    end for
end while
```

2.6 Shortcomings of Hokusai

Because Count-Min sketch accuracy depends on size, with Hokusai, we expect counts for recent time instances (that use larger sketches) to be more accurate than counts corresponding to older time intervals (that use smaller sketches).

Although this idea of having different-sized sketches for different time intervals is reasonable and yields accuracies that are time-adaptive, it comes with several inherent shortcomings which are summarized in the following subsections.

1. Discontinuity: Observe that in Algorithm 1, whenever t increments, we halve the size of sketches for $\log_2 t$ time instances. This abruptly doubles the errors for all items in those time instances, and happens at every transition from t to t+1. Even if the current time interval t is empty (or nearly empty), i.e., we do not add anything to the data structure, we still shrink sketches, unnecessarily reducing the accuracy of count estimates for many items. For instance, in a temporal stream, if the current time t has very few or no items, we still double our errors for time interval $t-1$, as well as many preceding it. This is undesirable and unnecessary, as we show below. In practice, such burstiness in data streams over time is very common, with some intervals adding many more counts than others. Owing to this discontinuity, we see oscillating accuracy over time with HOKUSAI. Ideally, accuracy for past counts should decrease smoothly as more items are included.

2. Too many sketches: We are keeping sketches for all the time instances $t \in \{0,\ 1,\ 2, ...,\ T\}$. In practice, the total time range T can be very large. If we keep sketches for all time points up to T, we cannot avoid using $O(T)$ memory, even if most of these intervals are "empty" and not associated with any items at all. When keeping time at a fine granularity, the $O(T)$ memory requirement can be become prohibitively expensive, making memory overhead of $O(\log T)$ highly desirable. If we force the total sketch size to be $O(\log T)$ while using the HOKUSAI approach, at least $T - O(\log T)$ time instances will have sketches of size zero, with all information corresponding to them lost.

3. Inflexibility: Algorithm 1 is inflexible, as all $t \in [t - 2^i, t - 2^{i+1}]$ use sketches of same size. With every time step, the size reduction in sketches also happens by a fixed amount which is half. With every transition of time from t to t+1, Hokusai doubles the error for specific time instances. There is not much flexibility in terms of the error distribution. Because we often do not know beforehand what error distribution over time is needed, having error tolerance that is flexible temporally would be very useful.

4. Overhead for shrinking: Whenever t increments, we need to shrink $\log t$ sketches by half, which involves one pass over all counters for those sketches.

Nonetheless, the key insight of HOKUSAI – shrinking sketch size with time – is a natural solution for achieving time adaptability. In subsequent sections, we demonstrate that there is an alternative approach, formulated in Section 4, that achieves time adaptability while rectifying the issues summarized above.

3. CMS FOR TEMPORAL QUERIES

CMS is a widely popular technique for summarizing data streams where typically we are interested in accurately estimating the heavy hitters (or items with significant counts). Before we describe our proposal, we argue that there is a trivial modification of CMS using which we can keep track of items over time without worrying about time adaptability. Our time adaptive variants will then build upon this modification.

We are interested in estimating the counts of items during any specified interval of time. To use existing CMS (or even LC) for such task, we can treat same items occurring in different time intervals as different items. In particular, we can distinguish item i during time instance t from the same item i during a different time instance t'. After this distinction, we can keep track of all distinct items over time, $N \times T$ of them, using the vanilla CMS (or LC). We need hash functions that hash both time and item, i.e. our hash functions will be

$$h_1, h_2, ..., h_d : \{1, 2, ..., N\} \times \{0, 1, 2, ..., T\} \mapsto \{1, 2, ..., w\}.$$

Effectively, we only increase the total number of items. The total number of items will be $N \times T$ which can be a very large number, because both T and N are huge in practice. Even space requirements of $O(N)$ or $O(T)$ are prohibitive and so handling $N \times T$ items is clearly still a concern.

Given the near optimality of the existing sketches, we cannot hope to be able to decrease the errors for all items consistently for all the time. However, there is a room for smartly distributing the errors to achieve the desired time adaptability described in the next section.

4. OUR PROPOSAL: TIME ADAPTIVE SKETCHES (ADA-SKETCHES)

We present Adaptive Count-Min Sketch (Ada-CMS) and Adaptive Lossy Counting (Ada-LS), which are time adaptive variants of Count-Min Sketch (CMS) and Lossy Counting (LC) respectively. The idea behind these adaptive sketches is analogous to the widely popular noise reduction process used in Dolby[2] via *pre-emphasis* and *de-emphasis*.

4.1 Short Detour: Dolby Noise Reduction, Pre-Emphasis and De-Emphasis

In Dolby systems, while recording the signal, a pre-emphasis transformation is applied, which artificially inflates certain frequencies in the signal relative to the tape's overall noise levels. This preserves the inflated signals with lesser distortion while recording on the tape. During playback an opposite transformation de-emphasis is applied which reduces back the inflated signals canceling the effect of artificial pre-emphasis. In the process, noise is attenuated more sharply. Please refer to [9, 24] for more details.

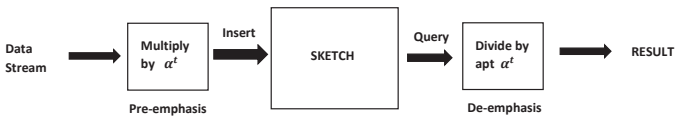

Figure 3: Adaptive Sketching via Pre-emphasis and De-emphasis analogous to Dolby Noise Reduction Procedure

We exploit the fact that Count-Min Sketch (CMS) and Lossy Counting (LS) have better accuracy for heavy hitters as compared to the rest of the items. While updating

[2]http://en.wikipedia.org/wiki/Dolby_noise-reduction_system

the sketch we apply pre-emphasis and artificially inflate the counts of more recent items compared to older ones, i.e., we make them heavier with respect to the older items. This is done by multiplying the updates c_i^t with $f(t)$, which is any monotonically increasing function of time t. Thus, instead of updating the sketch with c_i^t we update the sketch with $f(t) \times c_i^t$. The tendency of the sketch is to preserve large values. This inflation thus preserves the accuracy of recent items, after artificial inflation, compared to the older ones. While querying, we apply the de-emphasis process, where we divide the results by $f(t)$, to obtain the estimate of item i at time instance t. The procedure is simple and summarized in Figure 3. In this process, as we show later, the errors of the recent counts are decreased more than the errors of the older counts.

To understand why the pre-emphasis and de-emphasis defined in previous paragraph works, consider the case of CMS with only one hash function h_k, i.e. $d = 1$ and only a one row sketch as shown in Figure 4. The vanilla update procedure is simply adding c_i^t at location $h_k(i,t)$ for every increment c_i^t. During estimation of counts for item i in time interval t we simply report the value of $h_k(i,t)$.

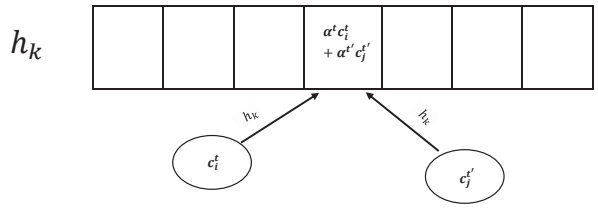

Figure 4: Toy illustration of a collision after Pre-emphasis

Now consider the case when we apply the artificial pre-emphasis and later de-emphasis to cancel it. When there are no collisions, just like the old sketches, we estimate the counts c_i^t exactly. Pre-emphasis and de-emphasis cancel each other and have no effect. Errors occur when there are collisions. For illustration, assume that $h_k(i,t) = h_k(j,t')$, i.e. both c_i^t and $c_j^{t'}$ collide and go to the same location. Suppose we have $t > t'$. The estimate of both c_i^t and $c_j^{t'}$ using vanilla CMS would be simply $c_i^t + c_j^{t'}$ because they both go to the same location and we simply return the counter. Note, we have only one row, and so the minimum is over one value. For the adaptive variant, based on the pre-emphasis and de-emphasis using a monotonic sequence $f(t)$ the estimates of c_i^t and $c_j^{t'}$ would be

$$\widehat{c_i^t} = \frac{f(t)c_i^t + f(t')c_j^{t'}}{f(t)} = c_i^t + \frac{f(t')}{f(t)}c_j^{t'}$$

$$\widehat{c_j^{t'}} = \frac{f(t)c_i^t + f(t')c_j^{t'}}{f(t')} = c_j^{t'} + \frac{f(t)}{f(t')}c_i^t.$$

Since $t > t'$, we will have $f(t) > f(t')$ because of monotonicity of $f(t)$ with respect to t. With vanilla CMS we overestimate c_i^j by $c_j^{t'}$ whereas the overestimation after pre-emphasis and de-emphasis is $\frac{f(t')}{f(t)}c_j^{t'}$ which is strictly smaller than $c_j^{t'}$. On the other hand, the error in $c_j^{t'}$ is $\frac{f(t)}{f(t')}c_i^t$ which is greater than c_i^t, the error with vanilla CMS. Thus, whenever there is a collision, the recent items suffer less compared to older items. This achieves the desired time adaptability in errors, given the same amount of space. The whole idea works for any sequence $f(t)$ which is monotonic in t. We now make this notion more rigorous.

4.2 Adaptive Count-Min Sketch (Ada-CMS)

Ada-CMS requires a monotonically increasing sequence $\{f(t); t \geq 0\}$ for performing the pre-emphasis and de-emphasis. Monotonicity implies $f(t) \geq f(t')\forall t > t'$. Similar to CMS we have a two dimensional cell with depth d and width w. The overall procedure is summarized in Algorithm 2.

Algorithm 2 *Adaptive Count-Min Sketch*

Input: Any monotonic sequence $\{f(t); t \geq 0\}$
$w \leftarrow \lceil \frac{e}{\epsilon} \rceil$
$d \leftarrow \log \frac{1}{\delta}$
$M \leftarrow d \times w$ array initialized to 0

Update c_i^t
for $j = 1$ **to** d **do**
$M(j, h_j(i,t)) \leftarrow M(j, h_j(i,t)) + f(t)c_i^t$
end for

Query for counts of item i **during** t
$\widehat{c_i^t} \leftarrow \min_{j \in \{1,2,\ldots,d\}} \frac{M(j,h_j(i,t))}{f(t)}$
return $\widehat{c_i^t}$

Just like the vanilla CMS described in Section 3, we hash both item and time simultaneously, i.e., our hash functions are from the product of time and items to $[1, w]$. The update and the query use the pre-emphasis and de-emphasis as described in Section 4. Other than this simple modification, the overall algorithm is same. Thus, clearly we have no additional overhead. If we make $f(t) = 1$, then we recover the vanilla CMS algorithm. Thus, Algorithm 2 is a strict generalization of CMS. This simple modification, as we show later, is very powerful and achieves the desired functionality of time adaptability. Furthermore, as shown in Section 4.2.3, we obtain many design choices, which can be tailored for different applications.

4.2.1 Analysis

The analysis follows similar lines with the analysis of vanilla CMS [7] except that we have to deal with the pre-emphasis and de-emphasis factor $f(t)$.

THEOREM 1. *For $w = \lceil \frac{e}{\epsilon} \rceil$ and $d = \log \frac{1}{\delta}$, given any (i,t) we have*

$$c_i^t \leq \widehat{c_i^t} \leq c_i^t + \epsilon\beta^t\sqrt{\mathcal{M}_2^T}$$

with probability $1 - \delta$. Here $\beta^t = \frac{\sqrt{\sum_{t'=0}^{T}(f(t'))^2}}{f(t)}$ is the time adaptive factor monotonically decreasing with t.

Intuition Behind the Proof: We will start by analyzing errors under just one hash function. Note, that if there are no collisions, then the estimation is exact. Collisions are bad events that cause errors (overestimation), but with large enough values of w, random collisions have small probability. This small probability will make the expectations of

error small. Furthermore, repeating the process with d independent hash functions, and finding the best (minimum) count among d repetitions of the experiment, will give the required high probability of concentration.

PROOF. Given i and t, define indicator variable $\mathbb{I}_{k,j}^{t'}$ as

$$\mathbb{I}_{k,j}^{t'} = \begin{cases} 1, & \text{if } (i,t) \neq (j,t') \text{ and } h_k(i,t) = h_k(j,t') \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$\mathbb{E}[\mathbb{I}_{k,j}^{t'}] = Pr\left(h_k(i,t) = h_k(j,t')\right) = \frac{1}{w}. \tag{2}$$

Let us consider a single row estimate of c_i^t corresponding to hash function h_k, i.e. row k, call it $\widehat{c_{k,i}^t}$.

$$\widehat{c_{k,i}^t} = \frac{1}{f(t)}\left[f(t)c_i^t + \left[\sum_{t'=0}^{T}\left(f(t') \times \sum_{j=1}^{N}\mathbb{I}_{k,j}^{t'}c_j^{t'}\right)\right]\right] \geq c_i^t. \tag{3}$$

$\widehat{c_{k,i}^t} > c_i^t$ is true for all k and hence also true for the minimum. For the other side of the bound we have,

$$\mathbb{E}[\widehat{c_{k,i}^t} - c_i^t] = \frac{1}{f(t)}\mathbb{E}\left[\sum_{t'=0}^{T}\left(f(t) \times \sum_{j=1}^{N}\mathbb{I}_{k,j}^{t'}c_j^{t'}\right)\right] \tag{4}$$

$$= \frac{1}{f(t)}\sum_{t'=0}^{T}\left(f(t') \times \sum_{j=1}^{N}\frac{1}{w}c_j^{t'}\right) - \frac{c_i^t}{w} \tag{5}$$

$$\leq \frac{1}{wf(t)}\sum_{t'=0}^{T} f(t')C^{t'} \tag{6}$$

$$\leq \frac{1}{w}\left(\frac{\sqrt{\sum_{t'=0}^{T}(f(t'))^2}}{f(t)}\right)\sqrt{\mathcal{M}_2^T} = \frac{\beta^t}{w}\sqrt{\mathcal{M}_2^T} \tag{7}$$

In getting Equation 5 we used linearity of expectation. Since $\mathbb{I}_{k,i}^t = 0$, it created a hanging term $\frac{c_i^t}{w}$, dropping which gives the next inequality instead of equality. The next step is Cauchy-Schwartz and the definition of $\mathcal{M}_2^T$. From Markov's inequality we have:

$$Pr\left(\widehat{c_{k,i}^t} - c_i^t \geq \epsilon\beta^t\sqrt{\mathcal{M}_2^T}\right) \leq \frac{1}{\epsilon}\frac{1}{w} \leq e \tag{8}$$

Our final estimate is the minimum over d estimates which are independent because of pairwise independence of h_k. So for the final estimate $\widehat{c_i^t}$ we have

$$Pr\left(\widehat{c_i^t} - c_i^t \leq \epsilon\beta^t\sqrt{\mathcal{M}_2^T}\right) \geq 1 - e^d = 1 - \delta. \tag{9}$$

as required. The monotonicity of β^t follows directly from the monotonicity of $f(t)$ in the denominator. □

Remarks on Theorem 1: We can see that since β^t is monotonically decreasing the upper bound on errors, given by Theorem 1, decreases as time t increases. Thus, for point queries we obtain time adaptive error guarantees. For a vanilla CMS sketch we can get the upper bound in terms of the L_1 norm (see [7] for details) which is simply the sum Σ^t. Here, we get the upper bound in terms of weighted sum (Equation 6) which is further upper bounded by $\sqrt{\mathcal{M}_2^T}$ due to Cauchy-Schwartz inequality.

Dealing with Overflows: Due to artificial pre-emphasis, it is possible that the counters in the sketch can overflow sooner (particularly in the exponential) than vanilla CMS. A simple way to deal with this occasional overflow is to periodically divide all counters by a constant to keep the overflow under control. If we know beforehand a safe constant to divide by, we can always divide even before adding elements to the sketch. During estimation we can multiply the outputs by those known constants.

Before we look into choices of $f(t)$ and start contrasting Ada-CMS with standard CMS for time adaptive data summarization, we emphasize why Ada-CMS is free from the set of shortcomings, inherent to Hokusai, described in Section 2.6.

4.2.2 Advantages of Ada-CMS over Hokusai

Since Ada-CMS only keeps one sketch and use pre-emphasis and de-emphasis to achieve time adaptability we automatically get rid of the first problem mentioned in Section 2.6. If certain time interval t is empty then there is no addition to the Ada-CMS sketch. Therefore, there are no extra collisions. The previous counters and hence their accuracies are unchanged. We do not have any shrinking overhead. The update and the estimation complexity of Ada-CMS is exactly same as that of CMS.

As argued in Section 3, we are dealing with $N \times T$ distinct items. The error guarantee shown in Theorem 1 holds for any point query with probability $1-\delta$. To ensure this upper bound for all $N \times T$ items with probability $1-\delta$, we need to choose $d = \log\frac{NT}{\delta}$ instead of $\log\frac{1}{\delta}$ and further take the union bound. Thus, with Ada-CMS we only need

$$dw = O(\frac{e}{\epsilon} \times \log\frac{NT}{\delta}) = O\left(\frac{(\log N + \log T - \log\delta)}{\epsilon}\right)$$

space to answer all possible point queries with provable time adaptive guarantees. On the other hand, Hokusai needs one sketch for every time instance t making it scale linearly with time $O(T)$, which is prohibitively expensive in practice.

We will further see in the next Section that we can handle a variety of design choices by exploiting the flexibility that we have in choosing the monotonic sequence $f(t)$. Later during evaluation, in Section 6, we will observe that the errors of Ada-CMS are smooth over time unlike Hokusai which abruptly changes errors over time.

4.2.3 Comparison with Vanilla Count-Min Sketch and Design Choices

Our aim is to distribute errors more smartly than CMS which has no information about time. In particular, we want errors for the recent counts to be very accurate while for the older counts we are ready to tolerate extra errors. In this section, we show that by appropriately choosing the sequence $f(t)$ we can obtain a variety of error distributions.

One major question that we are interested in is "Given a fixed space and current state of time T, what are the values of time $t \leq T$ where Ada-CMS is more accurate than vanilla CMS ?" We know that if we put $f(t) = 1$ for all t in Ada-CMS, then we recover the vanilla CMS algorithm. This observation comes handy for theoretical comparisons and leads to the following corollary:

COROLLARY 1. *For same w and d, the expected error in c_i^t with Ada-CMS is less than the expected error of Vanilla*

CMS if and only if

$$\sum_{t'=0}^{T} \frac{f(t')}{f(t)} C^{t'} \leq \sum_{t'=0}^{T} C^{t'} \tag{10}$$

PROOF. Immediate from combining Equation 5 with the fact that the final estimate is simply the minimum. □

Due to the monotonicity of $f(t)$ the condition is always true for $t = T$. For $t < T$, the left hand term is weighted sum and $\frac{f(t')}{f(t)} < 1 \quad \forall t' < t$. Thus, for t close to T the left hand side is likely to be smaller.

To get a sense of quantifications, a convenient way for comparison is to compare the upper bounds i.e., β^t. For Vanilla CMS we simply substitute $f(t) = 1$ and get $\beta^t = \sqrt{T}$. Therefore, for all time instance $\geq t$ such that

$$\frac{\sqrt{\sum_{t'=0}^{T}(f(t'))^2}}{f(t)} \leq \sqrt{T} \quad \text{or} \quad \frac{\sum_{t'=0}^{T}(f(t'))^2}{T} \leq (f(t))^2 \tag{11}$$

we get a better upper bound on errors with Ada-CMS for c_i^t compared to vanilla CMS. For instance, if we choose $f(t) = t + 1$ the above condition becomes

$$t \geq \sqrt{\frac{(T+1)(2T+1)}{6}} \geq \sqrt{\frac{1}{3}} T \simeq 0.57T \tag{12}$$

Thus, for recent half of the time we are more accurate than vanilla CMS.

Another interesting choice is exponential, i.e. $f(t) = a^t$ for some $a > 1$. With this we achieve better accuracy than CMS for all (assuming $a^{T+1} \gg 1$)

$$t \geq \frac{\log \frac{a^{T+1}-1}{(a-1)T}}{2 \log a} \simeq \frac{T+1}{2} - \frac{\log T + \log(a-1)}{2 \log a} \tag{13}$$

We can achieve finer control by more complicated choices of $f(t)$. To illustrate a reasonable scenario, suppose we want the errors with Ada-CMS to be never off by a factor γ away from that of vanilla CMS $\forall\, t$. This ensures that we guarantee accuracy within factor γ of what the original CMS would achieve to even very old heavy hitters. In addition, we want to be more accurate than CMS on all recent time $t > K$, for some desirable choice of K. Note that $\beta^t = \sqrt{T}$ for CMS. All these can be translated into the following constraints

$$\beta^t \leq \gamma\sqrt{T} \quad \forall t > 0 \quad \text{and} \quad \beta^t \leq \sqrt{T} \quad \forall t > K. \tag{14}$$

Because β^t is monotonically decreasing, the above condition is also satisfied if

$$\beta^0 = \gamma\sqrt{T}; \quad \beta^K = \sqrt{T} \quad \text{or}$$
$$\frac{\sum_{t'=0}^{T}(f(t'))^2}{T} = \gamma^2 (f(0))^2; \quad \frac{\sum_{t'=0}^{T}(f(t'))^2}{T} = (f(K))^2$$

To simplify let us choose $f(t) = \sqrt{at^2 + bt + 1}$, and after expansion we are left with the following system of simultaneous equation for a and b.

$$a\left[\frac{1}{6}(T+1)(2T+1)\right] + b\left[\frac{1}{2}(T+1)\right] = \gamma^2 - 1 \tag{15}$$

$$a\left[\frac{1}{6}(T+1)(2T+1) - K^2\right] + b\left[\frac{1}{2}(T+1) - K\right] = 0 \tag{16}$$

This system leads to the following choice of a and b

$$a = \frac{(\gamma^2-1)(B-K)}{K(BK-A)}; \quad b = \frac{(\gamma^2-1)(A-K^2)}{K(A-BK)} \tag{17}$$

$$A = \frac{(T+1)(2T+1)}{6}; \quad B = \frac{T+1}{2} \tag{18}$$

We require one more condition that $at^2 + bt + 1 > 0$, which is always true if $a > 0$ and $b > 0$. These two conditions is satisfied if $\gamma > 1$ and $\sqrt{\frac{(T+1)(2T+1)}{6}} > K > \frac{T+1}{2}$. $\gamma > 1$ is a very natural condition because $\gamma < 1$ would mean that we are universally better, than vanilla CMS for all time t. We cannot hope to achieve that, given the near optimality of CMS sketches.

Some Remarks: We have shown that there is significant room of distributing the errors in CMS more smartly using the idea of pre-emphasis and de-emphasis. The flexibility in engineering the monotonic sequence $f(t)$ makes it applicable to a wide variety of situation at hand. The choice of pre-emphasis and de-emphasis transformations along with the sketching parameters, thus, unifies the trade-off between memory, accuracy and "time adaptability" in a single generalized framework, which is very exciting.

Here, we have focused on giving importance to the more recent time interval t. The idea of pre-emphasis and de-emphasis is more general and not just limited to time. For instance, if we want to estimate more accurately a set of items coming from some important source, the idea of pre-emphasis and de-emphasis is naturally applicable, as well.

4.2.4 Extension to Other Popular Sketches

There are two major variants of CMS in literature: (i) CMS with conservative updates [11] and (ii) CMS with skipping [3]. These two variants can be naturally extended with the idea of pre-emphasis and de-emphasis. Count Sketches [6] which predate CMS is another popular algorithm in practice and it is very much similar in nature to CMS. The idea of pre-emphasis and de-emphasis is directly applicable to Count Sketches as well for obtaining adaptive error guarantees. The details and analysis is analogous to Ada-CMS and so we do not repeat them here. To demonstrate the generality of pre-emphasis and de-emphasis idea, we provide time adaptive variant of Lossy Counting (LC) which is a very popular deterministic algorithm for counting heavy hitters, and it is very different in nature from CMS.

4.3 Adaptive Lossy Counting (Ada-LC)

In line with Ada-CMS we can use pre-emphasis and de-emphasis idea to obtain an adaptive variant of the popular lossy counting (LC) algorithm. Ada-LC, just like vanilla LC, needs one map M of size l. To perform pre-emphasis and de-emphasis, for obtaining adaptability over time, we again need a monotonically increasing sequence $f(t)$.

Just like Ada-CMS, before adding an element c_i^t to the map we multiply it by $f(t)$. If the map is full then we remove the minimum element from the map to create space. While estimating c_i^t, we just report the count of the element in the map divided by $f(t)$ (de-emphasis). The overall procedure is summarized in Algorithm 3. A more practical algorithm is to delete all elements which are less than minimum plus some predefined threshold.

Algorithm 3 *Adaptive Lossy Counting*

Requirement: Any monotonic sequence $\{f(t); t \geq 0\}$
$l \leftarrow \lceil \frac{1}{\epsilon} \rceil$
$M \leftarrow Empty$
$min \leftarrow 0$

Update c_i^t
$M[(i,t)] \leftarrow M[(i,t)] + f(t)c_i^t$
if M is Full **then**
  $min = min_{a \in M} M[a]$
  for $a \in M$ **do**
    if $M[a] = min$ **then**
      Delete M[a]
    end if
  end for
end if

Query for counts of item i **during** t
if M contains (i,t) **then**
  return $\widehat{c_i^t} = \frac{M[(i,t)]}{f(t)}$
else
  return 0
end if

4.3.1 Analysis

The intuition why pre-emphasis and de-emphasis work for Ada-LC is easy to see. Since recent counts are inflated during pre-emphasis they are unlikely to be deleted compared to the older counts and hence their values are preserved more accurately in the map. This simple idea is also amenable to analysis and gives provable time adaptive error bounds.

Unlike the Ada-CMS, Ada-LC is deterministic and so the analysis does not involve probability.

THEOREM 2. *If $l = \lceil \frac{1}{\epsilon} \rceil$ then the Ada-LC returns an estimate $\widehat{c_i^t}$ such that*

$$c_i^t \geq \widehat{c_i^t} \geq c_i^t - \epsilon \beta^t \sqrt{\mathcal{M}_2^T}$$

where $\beta^t = \frac{\sqrt{\sum_{t'=0}^T (f(t'))^2}}{f(t)}$ is the time adaptive factor and it is same as in Theorem 1.

PROOF. The total amount of increment to M is all items in the stream along with the pre-emphasis $\sum_{t=0}^T f(t)C^t$. Let $min_{i,t}$ be the minimum value after the insertion of c_i^t. Then in the worst case all counters can be $min_{i,t}$ and we decrement $l \times min_{i,t}$ in this iteration. Thus, the maximum possible total decrement is $l \sum_{t=0}^T \sum_{i=1}^N min_{i,t}$. Since, the total decrement is less than the total increment we have

$$\sum_{t=0}^T f(t)C^t - l \sum_{t=0}^T \sum_{i=1}^N min_{i,t} \geq 0 \quad or$$

$$\sum_{t=0}^T \sum_{i=1}^N min_{i,t} \leq \frac{1}{l} \sum_{t=0}^T f(t)C^t \leq \epsilon \left(\sqrt{\sum_{t'=0}^T (f(t'))^2} \right) \sqrt{\mathcal{M}_2^T}$$

The last step is Cauchy-Schwartz. Thus, every counter has decreased by at most $\epsilon \left(\sqrt{\sum_{t'=0}^T (f(t'))^2} \right) \sqrt{\mathcal{M}_2^T}$. Since we always decrement any count in the map and increment c_i^t as $f(t)c_i^T$, we have

$$f(t)c_i^t \geq M[(i,t)] \geq f(t)c_i^t - \epsilon \left(\sqrt{\sum_{t'=0}^T (f(t'))^2} \right) \sqrt{\mathcal{M}_2^T}$$

Dividing by $f(t)$ leads to the desired expression. □

Since the error bound has the time adaptive expression equal to $\epsilon \beta^t \sqrt{\mathcal{M}_2^T}$ which is same as the expression for the Ada-CMS, we can directly borrow all the design choices shown in Section 4.2.3 for choosing $f(t)$.

5. RANGE QUERIES

In practice, we are typically interested in range queries over time. For example, count-based features are common in click-through rate prediction, such as counts of clicks in past 24 hours on a given ad. Define $c_i^{[t_i,t_2]}$ to be the count of item i during any specified time interval $[t_1, t_2]$. One naive estimate of this range query is simply the sum of all point estimates leading to a crude estimator:

$$\widehat{c_i^{[t_1,t_2]}} = \sum_{t=t_1}^{t_2} \widehat{c_i^t} \tag{19}$$

An obvious problem with this estimator is that the summation involves too many terms. The errors grow linearly with the length of the interval.

A well know solution [7] is to store multiple sketches for different size of dyadic intervals. The idea is to keep $\log T$ sketches, one each for the intervals of the form $[t2^k+1, ..., (t+1)2^k]$ (of size 2^k) with $k = 0, 1, ..., \log T$. When $k = 0$ this consists of the usual sketch for every time instance t or the interval $[t, t]$, when $k = 1$, these sketches keep estimates of two consecutive time intervals $[t, t+1]$, etc. The key observation, made in [7], is that any interval $[t_1, t_2]$ can be decomposed as a sum of at most $2 \log T$ dyadic intervals. For instance interval $[10, 20]$ can be decomposed into union of four disjoint dyadic intervals $[10, 10]$, $[11, 12]$, $[13, 16]$, $[17, 20]$. Thus, to estimate $c_i^{[10,20]}$, we only need 4 point queries compared to 11 with the crude estimator given by equation 19. The errors in the worst case grow as $O(\log T)$ instead of linearly in the length of the interval (or range). The price that we pay in terms of time and memory, for answering any range query, is an increase of a factor of $O(\log T)$. The overall complexity is still logarithmic in both total time T and total items N.

The idea of pre-emphasis and de-emphasis for range queries is quite straightforward except for one subtlety. We need to keep $\log T$ different adaptive sketches, one for each type of dyadic range, and use the same idea of decomposing ranges into at most $2 \log T$ intervals. However, we need to ensure that every time instance $t \in [t2^k+1, ..., (t+1)2^k]$, for sketches corresponding to this type of dyadic interval, gets the same pre-emphasis and later de-emphasis. For example, if we are keeping sketches to count intervals of size 2, i.e. of the form $[2t+1, 2t+2]$, we have to give same pre-emphasis to $2t+1$ and $2t+2$. This is because we cannot have a well defined de-emphasis for recovering $c_i^{[2t+1,2t+2]} = c_i^{2t+1} + c_i^{2t+2}$ from $f(2t+1)c_i^{2t+1} + f(2t+2)c_i^{2t+2}$ unless $f(2t+1) = f(2t+2)$. Implementation wise this is very simple: we only need to make $2t+1$ and $2t+2$ indistinguishable by changing t to $t' = \lceil t/2 \rceil$ (in general $t' = \lceil t/2^i \rceil$ for sketches corresponding to dyadic interval $[t2^i + 1, ..., (t+1)2^i]$).

5.1 Analysis

We focus our analysis on range queries with Ada-CMS, a very similar analysis applies to Ada-LC.

Define $t^k = \lceil \frac{t}{2^k} \rceil$. With this distinction, all $t \in [t2^k + 1, ..., (t+1)2^k]$ have can be denoted by the same value t^k. We keep one Ada-CMS sketch for intervals of form $[t2^k + 1, ..., (t+1)2^k]$ (of size 2^k) with $k = 0, 1, ..., \log T$. The point queries for this sketch $c_i^{t^k}$ directly give the count of interval $[t2^k + 1, ..., (t+1)2^k]$. With slight abuse of notation we will also use $t^k = [t2^k + 1, (t+1)2^k]$ to denote any dyadic range.

With a similar line of analysis as Theorem 1 it can be shown that if $w = \lceil \frac{e}{\epsilon} \rceil$ and $d = \log \frac{1}{\delta}$, then with probability $1 - \delta$ the following holds, for a sketch corresponding to any dyadic range of the form $[t2^k + 1, ..., (t+1)2^k]$:

$$c_i^{t^k} \leq \widehat{c_i^{t^k}} \leq c_i^{t^k} + \epsilon \beta^{t^k} \sqrt{\mathcal{M}_2^T}, \tag{20}$$

where

$$\beta^{t^k} = \frac{\sqrt{\sum_{t=0}^{T} (f(t))^2}}{f(t^k)} = \frac{\sqrt{\sum_{t=0}^{\lceil \frac{T}{2^k} \rceil} 2^k (f(t^k))^2}}{f(t^k)} \tag{21}$$

is a monotonically decreasing function of t^k.

Let $I_{dyd}^{[t_1,t_2]}$ be the set of all dyadic intervals that are contained in $[t_1, t_2]$. Formally,

$$I_{dyd}^{[t_1,t_2]} = \{[x, y] : [x, y] \subseteq [t_1, t_2] \ \& \ [x, y] = [t2^k + 1, (t+1)2^k]\} \tag{22}$$

for some $t \in \mathbb{Z}$ and $k \in \{0, 1, ..., \log T\}$.

We now define

$$B^{[t_1,t_2]} = \max_{t^k \in I_{dyd}^{[t_1,t_2]}} \beta^{t^k}. \tag{23}$$

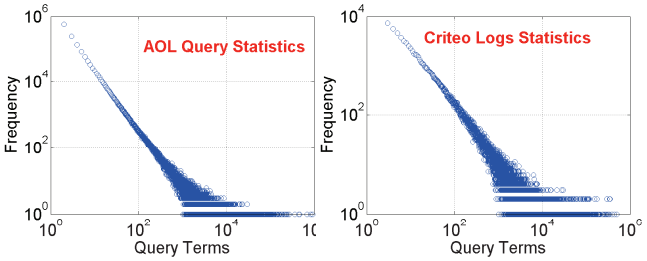

Figure 5: Frequency of items observed over the entire stream. The trends clearly depicts a power law as expected in real environments. The x-axis display items grouped by Matlab's tabulate function.

With this $B^{[t_1,t_2]}$ we have the following:

THEOREM 3. *Given $w = \lceil \frac{e}{\epsilon} \rceil$ and $d = \log \frac{1}{\delta}$, then with probability $1 - \delta$ we have for any given range $[t_1, t_2]$*

$$c_i^{[t_1,t_2]} \leq \widehat{c_i^{[t_1,t_2]}} \leq c_i^{[t_1,t_2]} + 2\epsilon B^{[t_1,t_2]} \sqrt{\mathcal{M}_2^T} \log T \tag{24}$$

PROOF. The proof follows from the fact that any interval $[t_1, t_2]$ can be written as union of at most $2 \log T$ disjoint dyadic intervals. For each of those dyadic intervals t^k we must have $t^k \subseteq [t_1, t_2]$. The error for estimating that dyadic range is upper bounded by $\epsilon \beta^{t^k} \sqrt{\mathcal{M}_2^T} \leq \epsilon B^{[t_1,t_2]} \sqrt{\mathcal{M}_2^T}$. This follows from the definition of $B^{[t_1,t_2]}$. The lower bound is trivial because we always overestimate. □

Remarks: The factor of $\log T$ appears in the upper bound along with some constants. This is analogous to upper bounds for range queries with vanilla CMS [7].

6. EXPERIMENTAL EVALUATION

Code: Algorithms were implemented in C#, with all experiments performed on an Intel Xeon 3.10 GHz CPU.

Datasets: Two real-world streaming datasets were used, *AOL* and *Criteo*, covering two important applications: online search and advertisement.

AOL: AOL query logs consist of 36 million search queries collected from 650 thousand users over 3 months. The dataset is comprised of search queries with associated timestamps, containing 3.8 million unique terms.

Criteo: Criteo conversions dataset contains feature values and conversion feedback for clicked display ads sampled over a two-month period [5]. Every ad is associated with a timestamp and 9 categorical terms hashed for anonymity, for a total of 150k unique hashed categorical terms.

Figure 5 plots the frequency of terms for both datasets in decreasing order on log-log scale. We use the Matlab tabulate function, which automatically computes the frequency and groups the plots. The plots reveal the power-law phenomenon common among real-world frequency distributions. An important problem in a variety of real applications is to keep track of the counts of important terms over time, in a streaming setting under a memory budget.

In these experiments, we focus on *time adaptability*: methods are compared by their accuracy on more recent counts, while we expect them to permit higher error rates for counts in distant past.

Competing Methodologies: Count-min sketching (CMS) is a widely popular algorithm in industrial practice. Our aim is to evaluate the effects of pre-emphasis and de-emphasis on CMS and compare it with HOKUSAI [20] algorithm that aims to make error rates time-adaptive. Since HOKUSAI is also built on CMS, we focus our comparison on the novel variants of CMS proposed in this paper. We omit comparisons to Lossy Counting in this section because, unlike CMS and Hokusai, it is deterministic, slow, and difficult to parallelize, due to the repeated linear-space-complexity creation step that executes every time when the map is full (see Algorithm 3 and Section 6.2). Therefore, for clarity, we defer the comparisons of Lossy Counting to its adaptive variants to Appendix C.

For comparisons, we keep track of all discrete items of interest (3.8 million in case of AOL and 150k for Criteo) over time using the following variants of CMS sketching schemes:

1. **Hokusai:** this algorithm was proposed in [20] for time-adaptive counting, based on CMS of different sizes for each time instance, described in Algorithm 1 and Section 2.5.

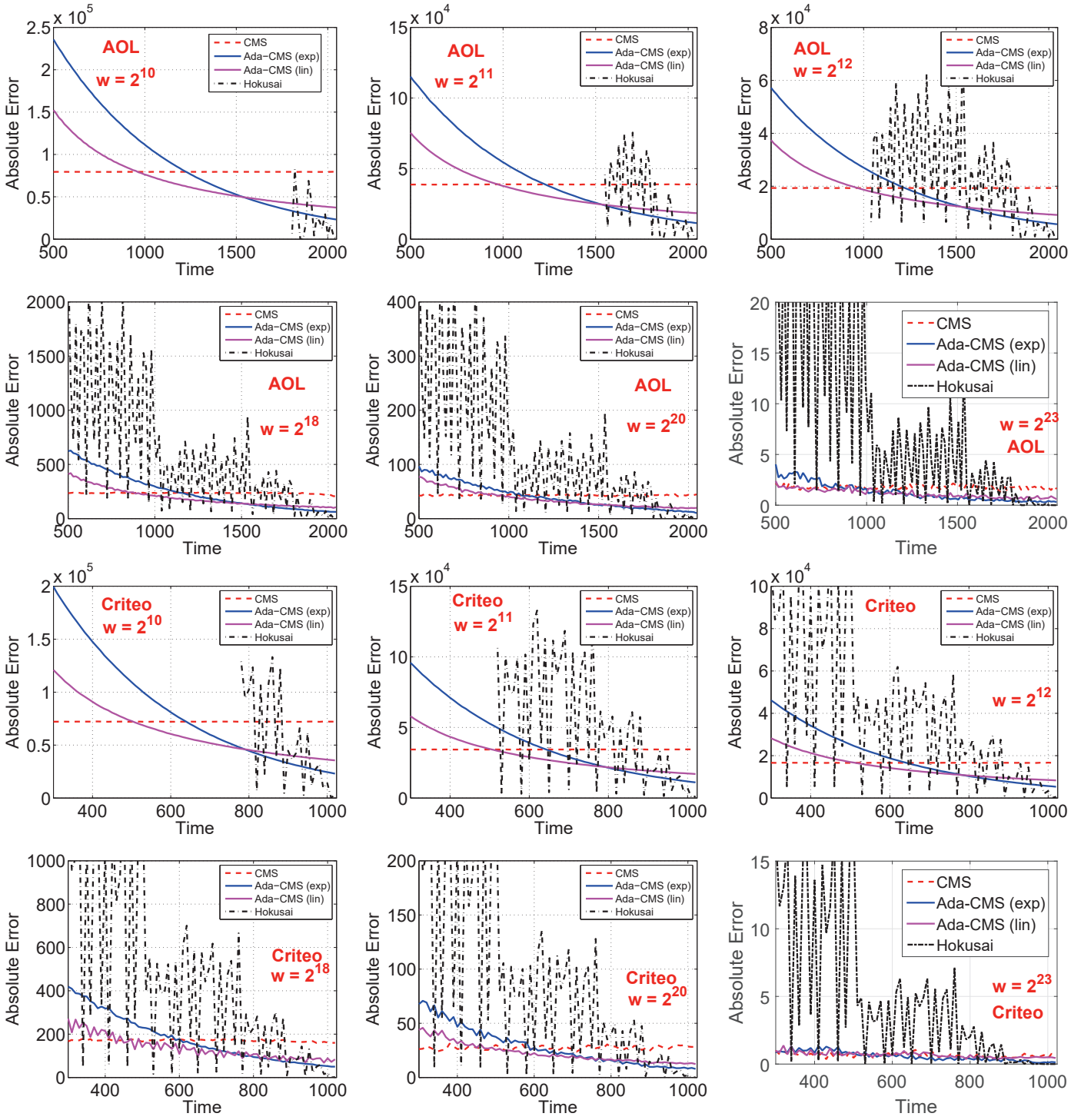

Figure 6: Average absolute error over time for different sketching scheme with varying sizes of range w (memory). A value of $d = 4$, (also used in [20]), was fixed in all the experiments. We ensured that all algorithms take exactly the same amount of memory while comparisons. CMS does not have any time adaptability. Hokusai keeps different sketches for all time intervals and therefore the error fluctuates. For smaller w ($w \leq 2^{12}$), Hokusai does not have enough sketches for all t. Both variants of Ada-CMS show the desired time adaptability, validating our claims, and is significantly better than Hokusai.

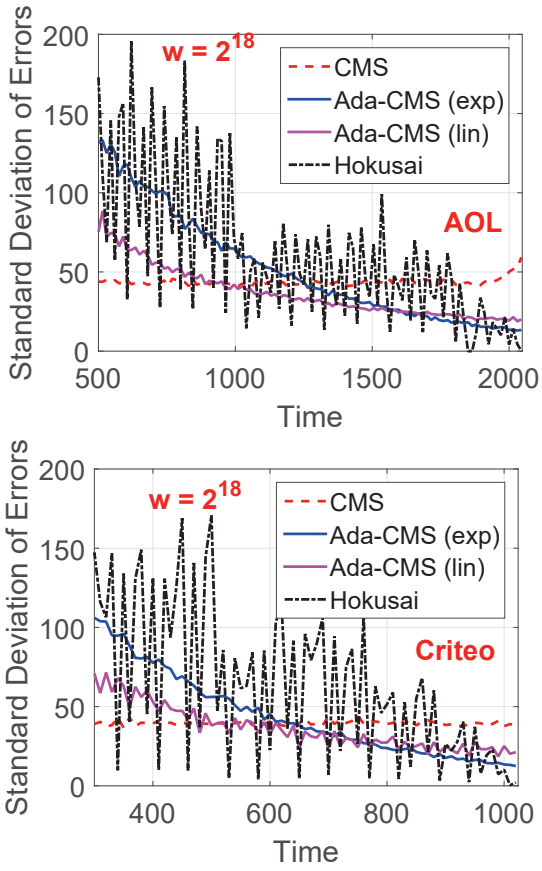

Figure 7: Standard deviation of errors for $w = 2^{18}$.

2. **CMS:** the basic CMS with the modification described in Section 3. Because it is not time-adaptive, it will serve as a baseline for the other algorithms.

3. **Ada-CMS (lin):** the proposed adaptive variant of CMS with pre-emphasis and de-emphasis using linearly increasing sequence $f(t) = a \times t$ for a fixed a. Please see Algorithm 2 and Section 4.2 for more details. All experiments used $a = 0.5$.

4. **Ada-CMS (exp):** the proposed adapative variant of CMS with pre-emphasis and de-emphasis using exponentially increasing sequence $f(t) = a^t$ for a fixed a. All experiments used $a = 1.0015$.

We use time granularity of one hour, thus each time instance t covers an hour of data. This yields around 2000 time instances for the AOL dataset, and around 1000 time instances for Criteo. We use the four competing algorithms to summarize the datasets. For *AOL* consisting of 3.8 million terms over more than 2000 time instances, the effective number of items is around $3.8M \times 2000 \simeq 8 \times 10^9$ which is a reasonably big number. For *Criteo*, the effective number of items is around $150k \times 1000 \simeq 1.5 \times 10^8$.

6.1 Accuracy-Memory Trade-off

Approximate counting algorithms provide a trade-off between estimation accuracy and the amount of space used. A superior algorithm provides a better tradeoff between accuracy and memory. Therefore, we compare the accuracies of all the algorithms for a fixed amount of space.

In the experiments, we kept the sketch depth of $d = 4$ fixed, following the setup used in [20]. This choice fixes the value of probability guarantee $1 - \delta$ to a constant value of $1 - \frac{1}{e^d}$ for all the four sketching schemes. To see the effect of variations on the sketch sizes, we varied the value of sketch width w in $\{2^{10}, 2^{11}, ..., 2^{23}\}$. For HOKUSAI, we chose the sketch size for each time instance so that the total amount of memory (the total size of all sketches) is exactly the same as the amount of memory used by the other algorithms. To ensure reasonable estimate quality, we verified that every time instance gets sketch size of $w \geq 4$, otherwise it simply returns the sum of stream over that time period. In some cases, ($w \leq 12$ for AOL and $w \leq 11$ for Criteo) we cannot have enough sketches, of any reasonable size, for all time spans. For such cases, we only report time instances for which Hokusai can make predictions. This is one of the shortcomings of Hokusai in that it requires a large number of sketches to keep track of all time instances because it needs one sketch for each of them.

In real settings, we are interested in estimates for items with high counts. For generating queries, we chose the top 1000 terms from both datasets based on their frequencies over the entire stream as items of interest. We then estimate the counts of these 1000 items for all the time intervals (2000 for AOL and 1000 for Criteo) by issuing point queries to the sketches. We compare the estimates of these counts with the true (gold standard) counts. The plot of the mean absolute error, of 1000 frequent items, over time for all the four competing schemes, is shown in Figure 6. We only highlight $w = \{2^{10}, 2^{11}, 2^{12}, 2^{18}, 2^{20}, 2^{23}\}$ to illustrate accuracy trends for high memory (2^{18-23}), as well as for low memory (2^{10-12}) regions. It should be noted that each of the plots shows accuracy for a given, fixed amount of memory for all the algorithms. In addition, we also illustrate the statistical significance of these numbers by showing the standard deviation of error estimates over time for $w = 2^{18}$ for both the datasets in figure 7. The standard deviation plots for other values of w exhibit analogous behavior.

Observations: the plots clearly show that the errors (both mean and standard deviation) for HOKUSAI fluctuate with time, producing significantly worse estimtes than the alternative sketches for both datasets. The fluctuations are expected because HOKUSAI keeps different CMS sketches for each time instance. Such fluctuations were also observed in [20]. It is known than the accuracy of CMS decreases with the total number of items added to the sketch. Therefore, for a busy time interval we can expect bad accuracy compared to non-busy intervals, leading to fluctuation in errors over time. With limited space $w \leq 2^{12}$, it is not possible to keep any reasonable sized sketch for all time intervals. For instance, when $w = 2^{10} \leq T \simeq 2000$, we cannot allocate any reasonable memory for all the time intervals. For such cases, we only report the time intervals we can track with HOKUSAI. Therefore, for those cases ($w < 2^{12}$), we do not see any values with HOKUSAI for all time instances t.

Baseline CMS does not provide any time adaptability, thus its error rates stay constant over time as expected. In contrast, the ADA-CMS results clearly demonstrate the time adaptive behavior for both mean and standard deviation. For recent time intervals, it is significantly more accurate compared to CMS and thereafter errors grow smoothly with decrease in time. Since CMS and Ada-CMS has only one

sketch, we do not see the undesirable fluctuations in the errors observed with Hokusai.

The difference between Ada-CMS (lin) and Ada-CMS (exp) clearly demonstrates the effect of choosing different forms for $f(t)$. With Ada-CMS (exp), the errors for the recent time intervals are better than those of Ada-CMS (lin) but grow faster than Ada-CMS (lin) with time on both the datasets. This clearly demonstrates that, in addition to time adaptability of errors, we have control over the rate of growth of errors over time. We have only shown linear and exponentially increasing sequences for demonstration, but can obtain a whole spectrum of errors by choosing various forms of $f(t)$. Irrespective of the growth rate, the errors and standard deviations with Ada-CMS are significantly less those of Hokusai for all experiments.

As we double the memory provided to the algorithms, the errors halve as expected. It is clear that with more memory the algorithms become more accurate, but Ada-CMS retains advantage over Hokusai, demonstrating a consistent trend. CMS and its variants are almost exact for $w = 2^{23}$, whereas Hokusai loses accuracy, especially for older counts.

6.2 Performance Comparisons

One of the major concern with Hokusai, as mentioned in Section 2.6, is the overhead of the shrinking operations. Not only does (the *item aggregation* scheme of) Hokusai requires one Count-min Sketch A^t for every time instance t but every transition of time from t to $t+1$ requires halving the size of $\log t$ many sketches. The halving of a sketch A^t requires a linear scan of all the cells in A^t, leading to poor scaling of Hokusai with the growing sketch sizes.

In this section, we compare the computational overheads of inserting streaming data into the sketches. Table 1 and Table 2 summarize the average running time, in seconds, of insertion for the *AOL* and *Criteo* datasets, respectively, using different sketching schemes. The averages are taken over 10 independent runs. Lower times indicate faster processing rate (throughput) of the sketching scheme. To further illustrate the effect of memory consumption (sketch size), we performed independent experiments with varying memory sizes. Results are shown for sketch widths $w = \{2^{20},\ 2^{22},\ 2^{25},\ 2^{27},\ 2^{30}\}$.

The results reveal how slow Lossy Counting (LC) is in general. To create space for the incoming items, once the map gets full, low-count elements must be purged; if we only remove the minimum element as suggested by the classical algorithm, then the algorithm insertion does not even finish in 2 -3 days. We therefore implemented a more practical version, in which, once the map gets full, we remove the items having smallest 10% of counts from the map. With this modification, LC and Ada-LC run in reasonable time.

The numbers clearly show the poor scalability of Hokusai with increasing sketch size. As the sketches increase, shrinking overhead in Hokusai with every increment in t makes the algorithm extremely slow. In contrast, for CMS and Ada-CMS, increases in sketch size have negligible affect on the processing time, which is not surprising given that the sketch size has no effect on the insertion time of the algorithm.

LC and its adaptive variants shows the opposite trend. LC and Ada-LC are deterministic algorithms (no hashing is used) that are hard to parallelize. When memory is reduced, the map quickly runs out of space and requires removal of smaller elements, a linear time operation. Therefore, under smaller memory budget, LC and Ada-LC are very slow. As the memory increases, space creation is less frequent, and LC and Ada-LC exhibit faster running times.

Table 1: Time (seconds) of summarizing the full *AOL* dataset using the respective sketches of different widths w. The numbers are averaged over 10 independent runs. Lower time indicates faster processing rate (throughput) of the sketching scheme.

	2^20	2^22	2^25	2^27	2^30
CMS	44.62	44.80	48.40	50.81	52.67
Hoku	68.46	94.07	360.23	1206.71	9244.17
ACMS_lin	44.57	44.62	49.95	52.21	52.87
ACMS_exp	68.32	73.96	76.23	82.73	76.82
LC	1334.843	578.70	268.67	267.48	253.40
ALC_lin	3184.40	1576.46	326.23	289.54	284.09
ALC_exp	1789.07	955.29	286.50	283.14	272.98

Table 2: Time (seconds) of summarizing the full *Criteo* dataset using sketches of different widths w. The numbers are averaged over 10 independent runs. Lower time indicates faster processing rate (throughput) of the sketching scheme.

	2^20	2^22	2^25	2^27	2^30
CMS	40.79	42.29	45.81	45.92	46.17
Hoku	55.19	90.32	335.04	1134.07	8522.12
ACMS_lin	39.07	42.00	44.54	45.32	46.24
ACMS_exp	69.21	69.31	71.23	72.01	72.85
LC	794.45	415.15	393.27	390.15	385.85
ALC_lin	3074.51	1472.34	432.14	398.21	393.10
ALC_exp	1291.82	821.27	388.06	372.15	365.18

CMS is popular in practice due its ease, parallelization and small overhead. All these properties are mimicked by the Ada-CMS variants proposed in the paper, which are strict generalization of CMS. By performing time-adapative sketching via pre-emphasis and de-emphasis, Ada-CMS avoids the overheads of Hokusai, and yields significant improvements in ability to trade off accuracy and memory, as shown in Section 6.1.

7. CONCLUSIONS

We found a mathematical opportunity that makes two widely popular sketching techniques temporally adaptive, thereby making them more suitable for massive data streams with drift over time. Our proposed technique, Ada-sketch, provides time-adaptive sketching that integrates the powerful idea of pre-emphasis and de-emphasis of with existing sketching techniques. As a result, we obtain strict generalization of classic sketching algorithms, unifying the trade-off between memory, accuracy and time adaptability in a single framework. This generalization comes with no additional overhead, and our proposal is simple to implement.

The proposed integration of sketches with pre-emphasis and de-emphasis, as we demonstrate, posseses strong theoretical guarantees on errors over time. Experiments on real datasets support our theoretical findings and show significantly superior accuracy and runtime overhead compared to the recently proposed Hokusai algorithm. We hope that our proposal will be adopted in practice, and it will lead to further exploration of the pre-emphasis and de-emphasis idea for solving massive data stream problems.

8. REFERENCES

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29. ACM, 1996.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2002.

[3] S. Bhattacharyya, A. Madeira, S. Muthukrishnan, and T. Ye. How to scalably and accurately skip past streams. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 654–663. IEEE, 2007.

[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[5] O. Chapelle. Modeling delayed feedback in display advertising. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1097–1105. ACM, 2014.

[6] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Automata, Languages and Programming*, pages 693–703. Springer, 2002.

[7] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[8] G. Cormode, S. Tirthapura, and B. Xu. Time-decayed correlated aggregates over data streams. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2(5-6):294–310, 2009.

[9] R. Dolby. Noise reduction systems, Nov. 5 1974. US Patent 3,846,719.

[10] C. Estan and G. Varghese. Data streaming in computer networking.

[11] C. Estan and G. Varghese. *New directions in traffic measurement and accounting*, volume 32. ACM, 2002.

[12] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[13] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets.

[14] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB*, volume 1, pages 79–88, 2001.

[15] T. Graepel, J. Q. Candela, T. Borchert, and R. Herbrich. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 13–20, 2010.

[16] D. Hillard, S. Schroedl, E. Manavoglu, H. Raghavan, and C. Leggetter. Improving ad relevance in sponsored search. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 361–370. ACM, 2010.

[17] O. Kennedy, C. Koch, and A. Demers. Dynamic approaches to in-network aggregation. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 1331–1334. IEEE, 2009.

[18] G. Krempl, I. Žliobaite, D. Brzeziński, E. Hüllermeier, M. Last, V. Lemaire, T. Noack, A. Shaker, S. Sievi, M. Spiliopoulou, et al. Open challenges for data stream mining research. *ACM SIGKDD Explorations Newsletter*, 2014.

[19] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357. VLDB Endowment, 2002.

[20] S. Matusevych, A. Smola, and A. Ahmed. Hokusai-sketching streams in real time. In *UAI*, 2012.

[21] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, et al. Ad click prediction: a view from the trenches. In *KDD*, 2013.

[22] S. Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.

[23] M. Richardson, E. Dominowska, and R. Ragno. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*, pages 521–530. ACM, 2007.

[24] S. V. Vaseghi. *Advanced digital signal processing and noise reduction*. John Wiley & Sons, 2008.

[25] Q. Yang and X. Wu. 10 challenging problems in data mining research. *International Journal of Information Technology & Decision Making*, 2006.

APPENDIX

A. RECONSTRUCTION EXPERIMENTS

In response to reviewer's feedback we will demonstrate the ability of Ada-CMS to preserve a signal in this section and use of Ada-CMS sketches for the reconstruction of time series. We use the same sketches used to summarize the whole AOL and Criteo dataset from the Experiments in Section 6. For illustration, we summarize the whole dataset using Ada-CMS with $w = 2^{21}$ and then reconstruct the counts of top 3 heavy hitter over time. We plot the actual counts and the estimated counts for both the dataset in Figure 8. We can clearly see that the two plots, despite burstiness, are almost indistinguishable from each other, showing near perfect reconstruction.

B. EVALUATION OF RANGE QUERIES

In this section we evaluate the four sketching schemes for estimating the counts over a range of given time interval $c_i^{[t_1,t_2]}$ as described in Section 5. We use the dyadic interval idea to decompose any given range $[t_1, t_2]$ into at most $2 \log T$ point queries. We need to store $\log T$ different sketches for answering range queries. We used the same values of parameters as before. From the previous section, we knew that small w is not sufficient for Hokusai to keep track of all time intervals. Therefore we only varied w in $\{2^{13}, 2^{14}, ..., 2^{20}\}$. Given a fixed value of w, we ensured that the total memory used by any of the four algorithms is same to ensure fair comparisons.

We again chose the same top 1000 frequent terms, used for point query experiments, as items of interest. This time we are interested in keeping track of the counts of terms per day

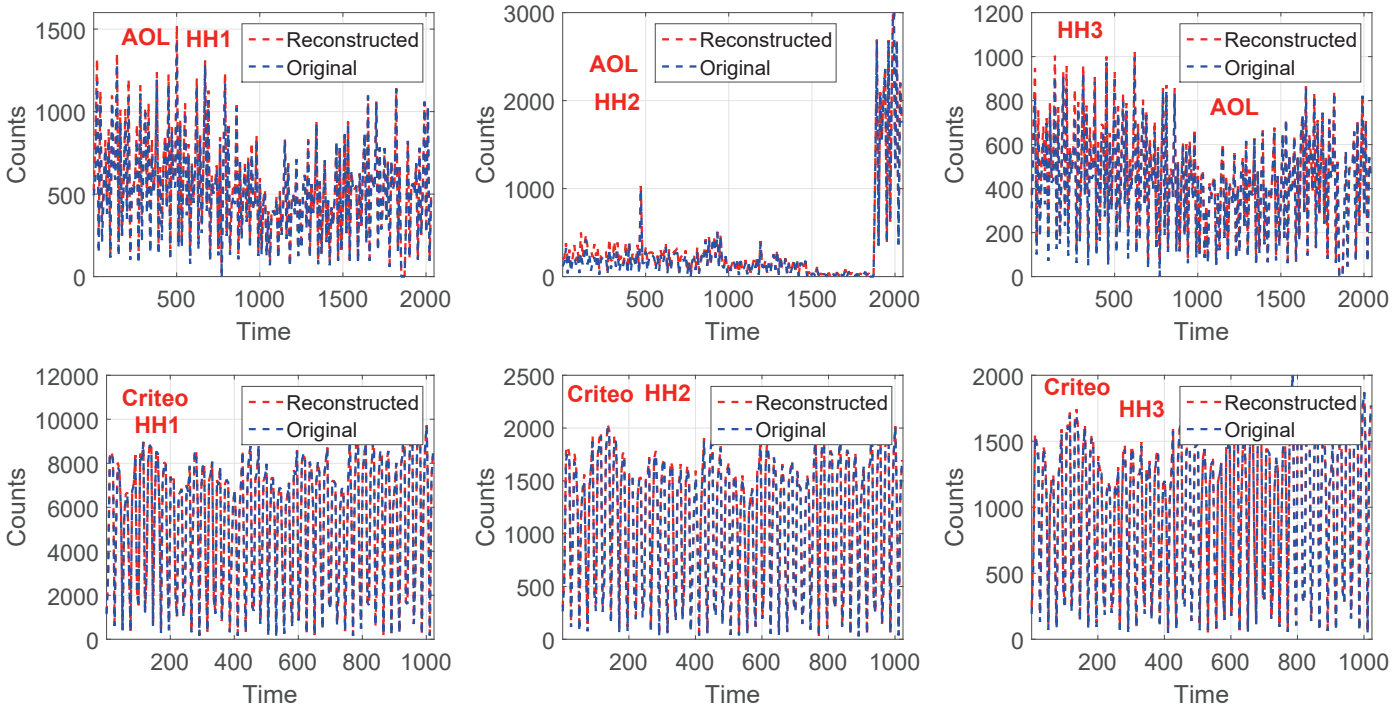

Figure 8: Actual and reconstructed counts of top 3 heavy hitters (HH) over time for both the datasets. The plots almost overlap.

for over 80 days with AOL and around 40 days with Criteo. We again set the time granularity of one hour, thus every t denotes an hour. So for estimating the counts of a given day, we are estimating intervals of size 24. We computed the gold standard error and plot the mean absolute error of the estimates returned by the four sketching schemes of interest. The mean was taken over the chosen 1000 terms of interest. The result is summarized in Figure 9. The errors with Hokusai are significantly higher than the errors of all other techniques, making it difficult to visualize the difference between CMS and Ada-CMS in the same graph; therefore we show the results of CMS and Ada-CMS in a separate Figure 10 to make the difference more visible.

Observations: We can clearly see that the errors with Hokusai are significantly worse with range queries compared to CMS and Ada-CMS on both the datasets. This is expected because range query estimate is sum of many point query estimates. Therefore, the errors add up. We know from the previous section that Hokusai is worse for point queries and this difference blows up for range queries. Ada-CMS again, as expected, shows the adaptive behavior, which clearly supports our claims. Also, we have the additional option to choose different $f(t)$ for different dyadic ranges but to keep things simple we only use the same values of $f(t)$, as used for point queries, in this experiment for all the sketches.

We can see that there are small fluctuations with CMS and Ada-CMS for range queries. It should be noted that the upper bounds shown in Theorem 3 comes from the fact that every interval can be decomposed into at most $2 \log T$ dyadic intervals. The term $2 \log T$ is the worst case and in many cases it is less, depending on the interval itself. For example, the intervals of size 10 $[9, 18]$ can be decomposed into just two dyadic ranges $[9, 16]$ and $[17, 18]$, whereas $[10, 19]$ has decomposition into 4 intervals $[10, 10]$, $[11, 12]$, $[13, 16]$, $[17, 18]$ and $[19, 19]$. The error estimates of each interval sum up. This explains the fluctuation in the Figure 10 for estimating range queries with CMS and Ada-CMS, which is universal for the dyadic interval trick. The global trend in errors is clearly adaptive.

C. MEMORY VS ACCURACY TRADEOFF FOR LC AND ADA-LC

In this section, we compare the accuracy of LC and its adaptive variants Ada-LC (lin) and Ada-LC (exp), over time, with a given fixed amount of memory. The experimental setup is same as described in Section 6.1, except that we replace the CMS variants of sketching algorithms with their LC counterparts. We also used the same $f(t)$ for linear and exponential variants. It should be noted that LC and Ada-LC keep a map for $(item, time)$ pairs. If the pair is present, then the estimation is very accurate for this $(item, time)$ pair, otherwise LC returns zero. LC and its variant always underestimate the counts, whereas with CMS and its variants we always overestimate. As noted in Section 6.2, LC and its variants are very slow. We use a more practical implementation in which we remove smallest 10% of elements from the map to make it run in reasonable time.

After summarizing the dataset with the LC and Ada-LC sketches, we compare the estimates of counts of 1000 frequent items over time with the corresponding true gold standard counts. The plot of the mean absolute error over time for LC and Ada-LC schemes, is shown in Figure 11. To show the variation with respect to memory we show plots for three different values of $w = \{2^{16}, 2^{17}, 2^{18}\}$.

Observations: We can see that Ada-LC, on an average, gets more hits for recent time interval compared to LC. For very old time instances, LC is better on an average.

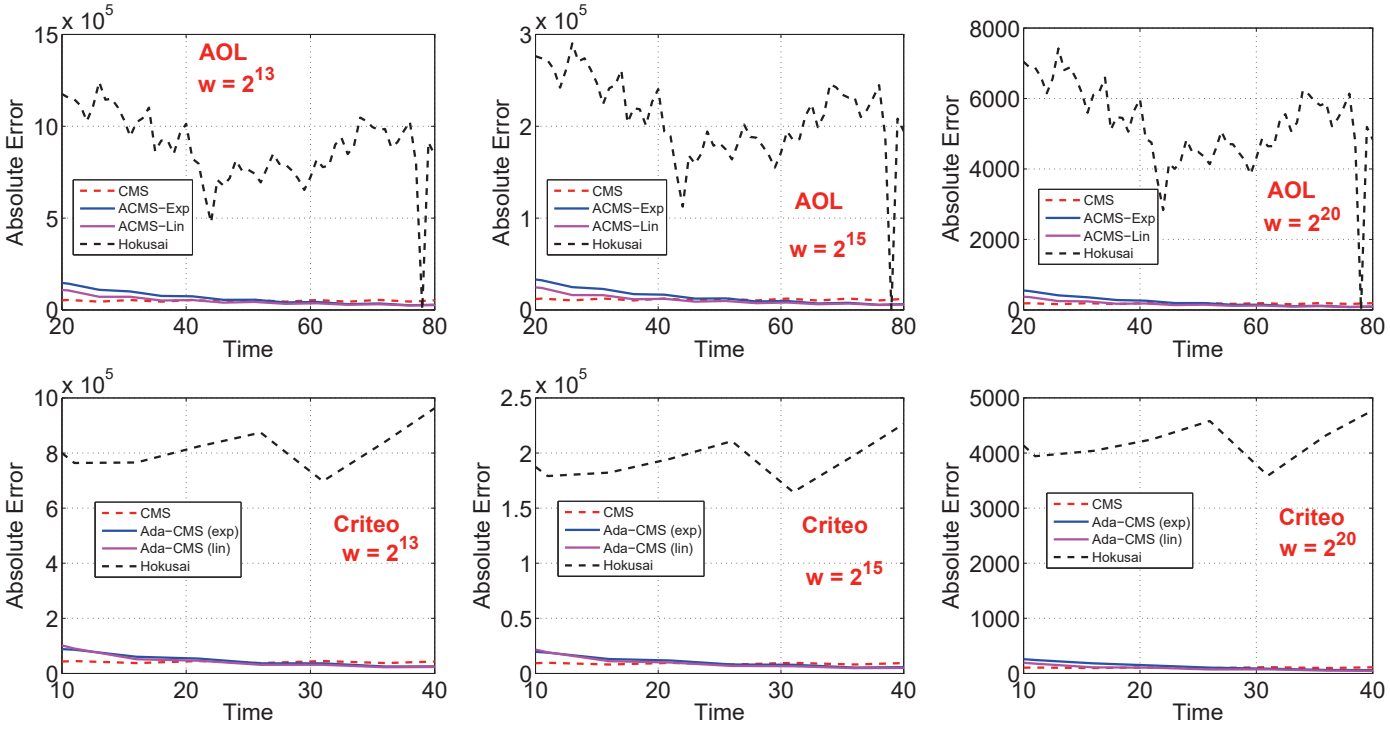

Figure 9: Average absolute error for estimating range queries with time for all the four competing schemes. Hokusai has significantly worse errors. See Figure 10 for a better visualization of comparisons between the errors of CMS and Ada-CMS.

This clearly demonstrates the adaptive nature of the Ada-LC compared to LC. The trends are consistent. Again Ada-LC (exp) is more aggressive than Ada-LC (lin) as expected. Thus, the adaptive variants, which are strict generalization of classical LC algorithm, achieve the desired time adaptability, which is not surprising given our theoretical results.

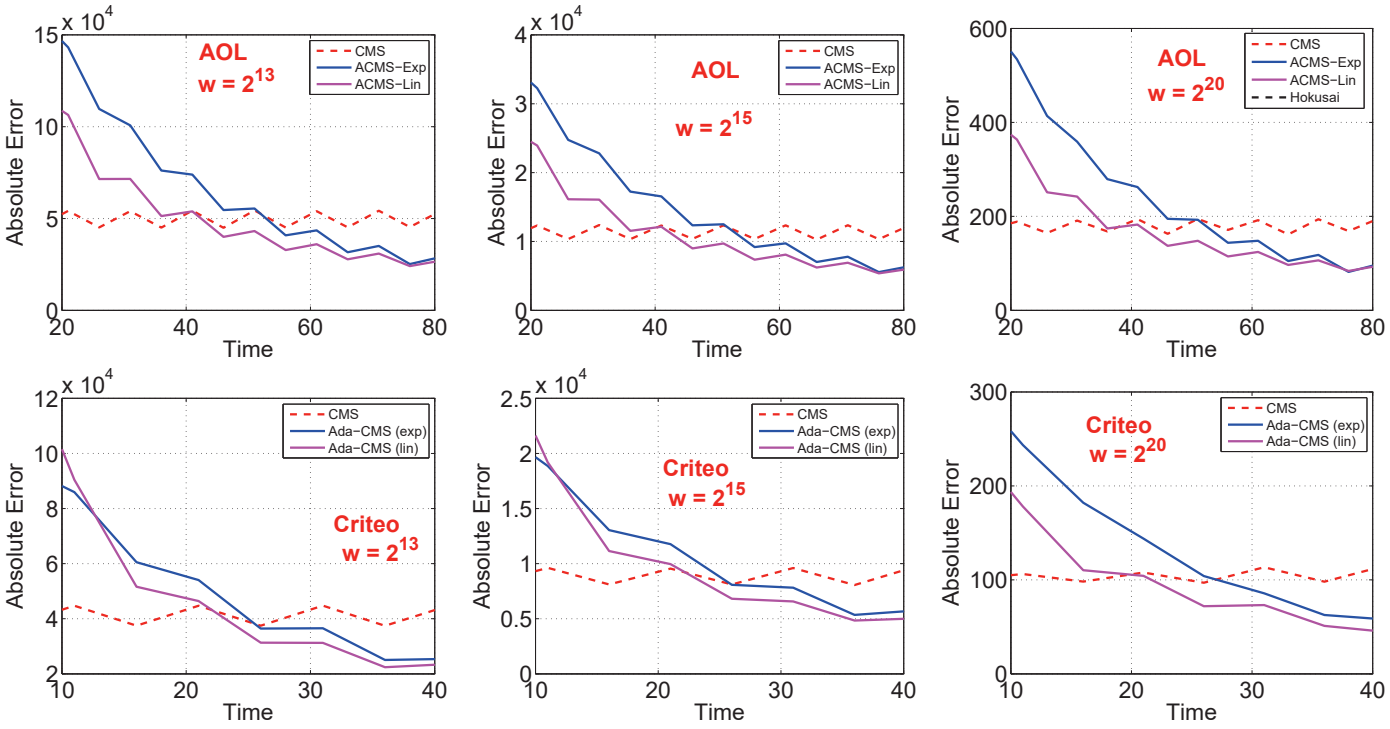

Figure 10: Average absolute error over time, for estimating range queries, with CMS and Ada-CMS. The global trend clearly shows the adaptive nature of Ada-CMS with time

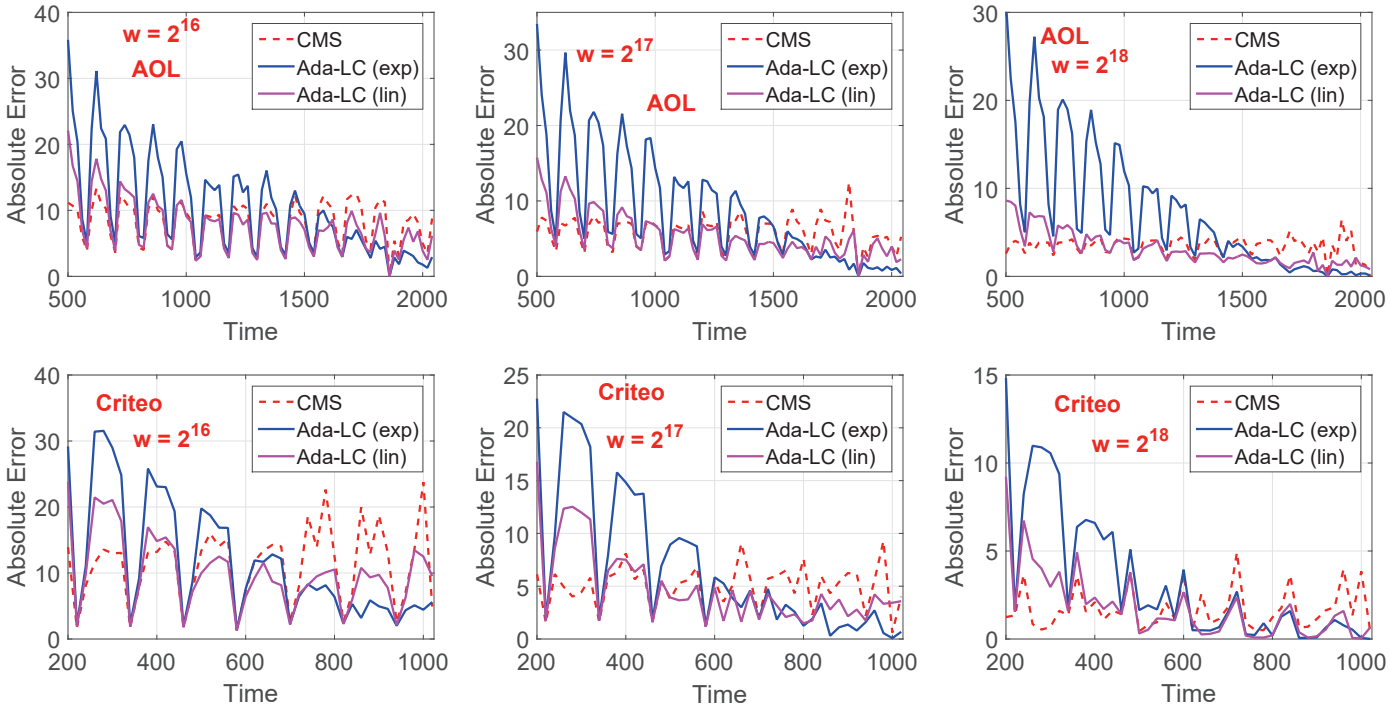

Figure 11: Average absolute error for estimating range queries with time for LC and its adaptive variants. On an average Ada-LC clearly shows adaptive behaviors.