Space-efficient Online Approximation of Time Series Data: Streams, Amnesia, and Out-of-order

Sorabh Gandhi, Luca Foschini, Subhash Suri
Department of Computer Science
UC Santa Barbara
{sorabh, foschini, suri}@cs.ucsb.edu

Abstract— In this paper, we present an abstract framework for online approximation of time-series data that yields a unified set of algorithms for several popular models: data streams, amnesic approximation, and out-of-order stream approximation. Our framework essentially develops a popular greedy method of bucket-merging into a more generic form, for which we can prove space-quality approximation bounds. When specialized to piecewise linear bucket approximations and commonly used error metrics, such as L_2 or L_∞, our framework leads to provable error bounds where none were known before, offers new results, or yields simpler and unified algorithms. The conceptual simplicity of our scheme translates into highly practical implementations, as borne out in our simulation studies: the algorithms produce near-optimal approximations, require very small memory footprints, and run extremely fast.

I. INTRODUCTION

Analyzing an online stream of measurements or transactions is a fundamental problem in database systems. Indeed, with the rapid growth and adoption of various tracking and monitoring technologies, database systems are under constant pressure to process in real-time the large volumes of data feeds that are generated by sensors embedded in environments, network infrastructures, financial markets, and so on. In many of these applications, the data is best thought of as a *time series*— an *ordered* sequence of measurements, where the order can be temporal as stock trades in financial markets, series of sensor measurements, retail transactions, etc., or spatial as the linear order defined by isocontours in some physical sensor measurements, or some other externally imposed order.

The goal of an online time-series approximation scheme is to summarize the data in one-pass and construct an approximate representation that can support a variety of time-series queries with bounds on worst-case error. Given the importance of time-series data, many approximation methods are known [6], [15], [17], [19], [25], although research in *online* approximation and worst-case error bounds is relatively recent [3], [10], [11], [12], [13]. Some of these schemes such as those based on wavelets can require an expensive *post-processing* phase to reduce a larger set of wavelet co-efficients to a B-bucket approximation. This includes the current theoretical best scheme due to Guha [10], [11] which achieves $(1 + \varepsilon)$ approximation but requires $O(B^3 \varepsilon^{-2} \log^2 n)$ time post-processing to cull down the $O(B \log n)$ wavelet coefficients into a B-bucket representation. In this work, we are particularly interested in combinatorial approximation schemes that maintain a *query-ready* synopsis in real-time— that is, at all times, the approximation is available for query processing with no additional computational overhead. In this regard, our approach is similar to the methods of [3], [24]. As discussed by Guha [10], when wavelets are used directly as a queriable structure [5], one only gets an approximation quality of the form $(O(1), \log n)$ in the standard (α, β) terminology; that is, α times the error of the optimal scheme using β times the optimal number of buckets. In order to achieve $(O(1), O(1))$ approximation, one requires the expensive post-processing mentioned earlier [10], [22].

Our main contribution is the introduction of a *generic* geometric method for approximating a one-dimensional signal that has the following properties: (1) at all times, it maintains a *query-ready* piecewise linear approximation of the time-series seen so far; and (2) as long as the error metric satisfies some weak conditions (non-negativity, monotonicity and additivity), the approximation provides (α, β)-type worst-case optimality bounds. More specifically, we show that a simple *greedy bucket-merge* method that has been used in specific settings (maximum error [3] and euclidean distance based histograms [16] and maximum error amnesic approximation [24]), sometimes without any accompanying worst-case error analysis, can be *abstracted* to a generic form and applied more broadly with worst-case approximation bounds where none were known before [16], [24], and simpler and unified algorithms in other cases. In particular, we show that, besides the data stream model, the new generic scheme is easily adapted to two other variants of time-series approximation that have received significant attention lately: *out-of-order* streams and *amnesic* approximation.

The *out-of-order* stream model is motivated by the asynchrony in many distributed and networked data processing applications [8], such as sensor networks. When the data is produced at dispersed locations and then transmitted to a collection or processing node, data values may not arrive in the strict time-order in which they were recorded due to various networking delays and asynchronous communication. An approximation scheme for the out-of-order stream should be tolerant of a small number of *gaps* in the arrival sequence. We show that our approximation scheme works for the out-of-order time series model, increasing the memory requirement only by an additive term $O(k)$, where k is the number of gaps in the time series at the current time.

An *amnesic* approximation is motivated by the fact that in

most applications of time-series, *the value of data decreases with age* [23], [24]. An analyst is likely to be most concerned with the recent data, and as the data gets old, his tolerance for the error of approximation grows. Thus, given a fixed memory budget, a natural way to optimize its use in the time-series approximation is to grow the error tolerance with age, so that progressively smaller fraction of the representation is spent on older data. While there have been several models for aging the data (exponential time decay, polynomial time decay, etc), our approach is general, and works with any given *monotone* amnesic function. Specifically, given a piecewise constant monotone *amnesic function*, describing the error tolerance over the entire history, our scheme achieves a constant factor approximation. We begin with some notation to introduce our problems, followed with a statement of our results.

A. Preliminaries and Contributions

We assume that a time series is given as a sequence of value $S = <v_1, v_2, \ldots, v_n>$, where each value v_i is associated with a *timestamp* t_i. For our purposes, the timestamps are an abstract entity with monotonically increasing values. We are not concerned with whether the timestamps reflect the time at which the data values are generated (source) or the time at which they are processed (sink)—any consistent view of timestamps will suffice. In the stream model, we only require that before the item with timestamp t_i arrives, all items with timestamps $t_j < t_j$ have arrived, but this assumption is relaxed in the out-of-order stream model.

We consider the problem of maintaining an approximation of the time series. A popular approximation method is to partition the data into consecutive blocks called *buckets* and represent every bucket by a single value, called a piecewise constant (PC) approximation. Instead, if we approximate the values of each bucket by a linear function (line segment), we get the piecewise linear (PL) approximation. The quality of an approximation is measured by error between the true time series and its approximated version, and L_p norm is a popular form of measuring the error. Specifically, the L_p norm error of the approximation is defined as follows:

$$E_p = \left(\sum_{i=1}^{n} |v_i - \hat{v}_i|^p \right)^{1/p} \qquad (1)$$

where $\hat{v}_i$ is the approximation for value v_i. The L_∞ norm is defined as:

$$E_\infty = \max_{i=1}^{n} |v_i - \hat{v}_i| \qquad (2)$$

Our primary focus is on L_∞ and L_2 norms.

The performance of our algorithms are expressed using the (α, β) approximation guarantees, meaning we achieve α times the error of the optimal scheme using β times the optimal number of buckets. Our main results are summarized below.

1) We show that a generic bucket-merge scheme achieves a $(2, 4)$ approximation for a broad class of error metrics that satisfy non-negativity, monotonicity and additivity.
2) For commonly used metrics such as L_2 and L_∞ and piecewise constant or piecewise linear approximations,

our scheme admits a simple and highly efficient streaming implementation that compares favorably with the existing methods. In particular, for the L_2 error, our algorithm maintains a B bucket approximation using $O(B)$ working space and $O(\log B)$ update time. While our bounds are worse than the best known theoretical result [10], [11], our scheme delivers comparable approximation quality in simulations, *with the advantage of being significantly faster.*
3) We show that our scheme easily adapts to the amnesic model of time series [24]. In particular, this adaptation called AMNESIC-MERGE achieves $(1+\varepsilon, 3)$ approximation using $O(\varepsilon^{-1/2}B \log(1/\varepsilon))$ memory and $O(\log B + \varepsilon^{-1/2}\log(1/\varepsilon))$ per item update time for the piecewise linear approximation. For piecewise constant approximation AMNESIC-MERGE achieves a $(1, 3)$ approximation using $O(B)$ memory and $O(\log B)$ per item update time. (This problem is called the UAA problem in [24], using piecewise constant monotonic amnesic functions.)
4) We also show that our scheme extends easily to out-of-order streams, where the data values may not arrive in their intended (stamped) order. Our scheme maintains piecewise approximations (PC or PL) for only the data that have arrived while still ensuring worse-case bounds on the error of approximation. (The need to bound approximation error requires that no bucket should include a "missing value" because otherwise an adversary can force an unbounded error by assigning arbitrarily large values to the missing data in the future.) Our variant, called the FRAGMENT-MERGE achieves $(\sqrt{2}, 5)$ approximation for the L_2 error using $O(B)$ space and $O(\log B)$ update time, and $(1+\varepsilon, 3)$ approximation for L_∞ error using $O(\varepsilon^{-1/2}B \log(1/\varepsilon))$ space and $O(\log B + \varepsilon^{-1/2}\log(1/\varepsilon))$ update time.
5) We perform extensive simulations to validate the empirical performance of our schemes on a variety of synthetic and real-world datasets. Our results confirm the space and time efficiency of our algorithms, which together with its simplicity and generality make it well-suited for many applications.

Our paper is organized as follows. In Section II, we discuss related work. In Section III, we describe the GENERIC-MINMERGE algorithm and its error analysis. Section IV, V, and VI discuss applications of this generic technique: stream histograms, amnesic approximations and out-of-order streams. In Section VII, we discuss our simulation results.

II. RELATED WORK

There is a large body of research on streaming histograms in the database community. Due to space constraints, we are able to mention only those that are closely related to our work. A non-streaming dynamic programming scheme for constructing a B-bucket histogram was originally proposed by Jagadish et al. [14], with time complexity $O(n^2B)$ and space $O(nB)$, which was subsequently improved by Guha [10] to $O(n^2B)$ time and $O(n)$ space. Currently, the best worst-case bounds for streaming histogram construction are due to

Guha and his colleagues [9], [10], [11], [12], [13]. Their techniques tend to maintain a summary structure of small memory, require $O(1)$ processing time per item, and then require some post processing at the end to compute the final approximation. Specifically, for the L_2 metric, they achieve a $(1 + \varepsilon, 1)$ approximation in $O(n + B^3 \varepsilon^{-2} \log^2 n)$ time with $O(B \varepsilon^{-2} \log n \log \varepsilon^{-1})$ space [10]. A very recent result of Guha [11] removes the dependence on $\log n$ from the space bounds.

Since many real-time applications deal with an intermixed sequence of updates and queries, an expensive post-processing before each query can be a bottleneck. As a result, we seek representations that dynamically maintain an approximation, always ready to process queries. In [3], Buragohain et al. propose fast lightweight algorithms for query-ready approximations, but their schemes only works for the L_∞ error metric. For other error metrics we point the reader to [10] as a good survey for existing techniques. Many time-series approximation scheme use a natural version of greedy bucket merging [24], [16], [3] in which the algorithm repeatedly greedily merges two buckets whenever it needs to allocate a new bucket. In particular, the GRAP-R algorithm proposed for the L_2 metric in [24] is very similar to the merging scheme proposed in our paper, but it lacks a worst-case error analysis. In the computational geometry community, several algorithms are known for curve simplification using Hausdorff and Fréchet metrics, but their focus is not on time series.

The amnesic approximation problem was proposed in [24], where the authors perform an extensive study and show very good results in practice. But they offer no worst case guarantees. In fact, our merge scheme is a minor variation (controlling when the merge happens) but we can prove worst-case approximation bounds for our version.

Out-of-order stream processing has been a topic of interest in many recent works in the database community. There have been new architectures proposed for handling out of order processing [20], [21] for a variety of problems, with the main concern being memory footprint. Many researchers have proposed heuristics [2], [26] and guaranteed algorithms [4], [8] for answering various types of queries for out-of-order data including quantiles, heavy hitters, sliding window counts. We solve the problem of maintaining histograms over out-of-order streams which is a well known aggregation technique for answering queries, and we show that the memory footprint required for our technique is very small.

III. THE GENERIC-MINMERGE

We want to compute a B bucket approximation of a time series of values $S = <v_1, v_2, \ldots, v_n>$, where B is an input parameter. The piecewise constant (resp. linear) approximation means that the values in each bucket are approximated by a single value (resp. linear function). The *error of the approximation* is typically measured by the L_p norm distance between the original time series and its approximated version. We will use the notation $e(S, b)$ to denote the optimal error in approximating the time series S using b buckets; the specific L_p norm and the approximation type (constant or linear)

will be clear from the context. We begin with an abstract formulation of the problem that attempts to capture the essence of a *greedy bucket-merge* that is used in practice. We say that the piecewise approximation (constant or linear) for an error metric is *well-behaved* if the following three conditions hold:

1) *Nonnegativity*: The error is always non-negative.

$$e(S', b) \geq 0, \quad S' \subseteq S, \quad b \geq 1 \qquad (3)$$

2) *Monotonicity*: The error of approximating $S_1 \cup S_2$ using one bucket is at least as large as the error in approximating S_1 and S_2 separately with one bucket each.

$$e(S_1 \cup S_2, 1) \geq e(S_1, 1) + e(S_2, 1) \qquad (4)$$

Figure 1 illustrates this property with an example.

3) *Additivity*: The error of a B-bucket approximation is the sum of the errors of the individual buckets. (Please see the remark below.)
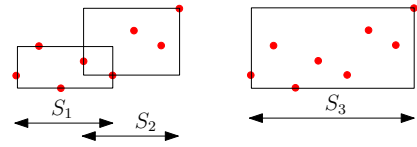

Fig. 1. Illustrating monotonicity, using L_∞ error. The error for the combined set with one bucket is at least as large as the error of two subsets.

Remark: One can verify that PC or PL approximations using all L_p norms (including L_∞) are well-behaved—simply use the pth power of the L_p norm as the error e to see that additivity and monotonicity hold. In the limit as $p \to \infty$, this holds for the L_∞ norm as well, but one can also replace sum with max in appropriate places in the proofs to get results for the L_∞ norm as well. The Hausdorff metric is also well-behaved, but the Fréchet is not [1].

A. The Algorithm

Algorithm 1 GENERIC-MINMERGE (S, B)

1: $\mathcal{M} = \emptyset$
2: **for** $next\ v_i \in S$ **do**
3:    Allocate new bucket u
4:    $\mathcal{M} = \mathcal{M} \bigcup \{u\}$
5:    **if** $|\mathcal{M}| \geq 4B$ **then**
6:       $\{v_1, v_2\} = $ BEST-ADJPAIR$(\mathcal{M})$
7:       $v := $ MERGE(v_1, v_2)
8:       $\mathcal{M} = \mathcal{M} \setminus \{v_1, v_2\}$
9:       $\mathcal{M} = \mathcal{M} \bigcup \{v\}$
10:    **end if**
11: **end for**

The generic merge scheme, denoted GENERIC-MINMERGE is explained in pseudo-code in Algorithm 1. It takes as input a time series S and B, the number of buckets. At any time, the buckets impose a partition of S into intervals, where two buckets are *adjacent* if they represent two consecutive intervals of this partition. Our approximation maintains a partition with at most $4B$ buckets. When a new item arrives, it is placed
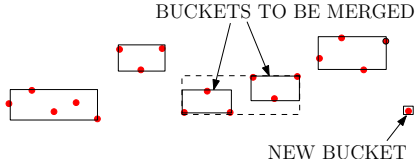
Fig. 2. Illustrating Min-Merge. The new item is put in a new bucket and the two buckets whose merger leads to the least error bucket are merged (shown with a dashed box).

in its own (newly allocated) bucket. *If this new bucket causes the number of buckets to exceed 4B, we shrink the number of buckets by one by merging an appropriate pair of adjacent buckets. In particular, we merge those two adjacent buckets that result in a bucket with smallest error.* This is achieved by functions BEST-ADJPAIR and MERGE. An illustration of one iteration of the algorithm is shown in Figure 2. In the next section we prove that this scheme achieves the $(2, 4)$ approximation.

B. Analysis of Generic-Minmerge (GM)

We begin by stating a crucial property, called *Min-Merge* maintained by the buckets produced by the GM algorithm at any time. *An approximation P is said to obey the Min-Merge property if the approximation error of a bucket formed by merging any two adjacent buckets in P is greater than (or equal to) the approximation error of any of the buckets before the merge.* Then by Algorithm 1 we have the following, whose proof is omitted from this abstract due to space constraints.

Lemma 1: The buckets in the GENERIC-MINMERGE algorithm obey the Min-Merge property.

We now prove the main result of this section.

Theorem 1: For any well-behaved error metric, GENERIC-MINMERGE achieves a $(2, 4)$ approximation.

Proof: We present our proof when the objective is to minimize the *sum* of the bucket errors; the proof for the *maximum* error follows by changing summation to max. Let E^* denote the optimal error of approximation using B buckets, and let E denote the error achieved by the GM algorithm using $4B$ buckets. We will show that

$$E \leq 2E^*. \tag{5}$$

For ease of reference, we will denote the buckets in the optimal solution as optimal buckets, and the buckets of the latter as the merge buckets: in both solutions, the buckets partition the set into intervals. Now, the B optimal buckets can *intersect* at most $B-1$ merge buckets, where the intersection means that an endpoint of the optimal bucket lies in the interior of the corresponding merge bucket. Let us call a merge bucket intersected by the optimal bucket a *split bucket*, and let $b_s \leq B-1$ be the number of split buckets. Let E_s be the total error associated with these b_s split buckets. We call the remaining merge buckets the *non-split buckets*, and let E_n be the total error associated with these buckets. Now, since the split and non-split buckets account for all the buckets of the GM algorithm, we have

$$E \leq E_s + E_n \tag{6}$$

(This is an equality for L_p norms, for finite p, and the max can be substituted for the sum for the L_∞ norm.)

We now argue that

$$E_n \leq E^* \tag{7}$$

Let $\{T_1, T_2, \ldots, T_B\}$ denote the partition of S by the B optimal buckets. That is, T_i is the subset approximated by the ith bucket in the optimal representation. Similarly, let $\{T'_1, T'_2, \ldots, T'_{b_n}\}$ denote the sets approximated by the b_n non-split buckets of GM. Observe that for every j, we must have an i such that $T'_j \subseteq T_i$, although some T_i sets may not have any T'_j as subset. Let Y be the set of indices such that $i \in Y$ iff for some j, $T'_j \subseteq T_i$. Thus, we have

$$\sum_{i \in Y} e(T_i, 1) \geq \sum_{j=1}^{b_n} e(T'_j, 1) \tag{8}$$

This follows from the monotonicity property because each non-split bucket lies inside some optimal bucket. Finally, since $E^* \geq \sum_{i \in Y} e(T_i, 1)$, we get that $E_n \leq E^*$.

Next, we argue that $E_s \leq E^*$. If $\{T''_1, T''_2, \ldots, T''_{b_s}\}$ are the subsets approximated by the split buckets, then we clearly have that

$$E_s = \sum_{i=1}^{b_s} e(T''_i, 1) \tag{9}$$

Call a pair of adjacent non-split buckets a *witness pair* if those buckets lie in a common optimal bucket. Call a set of witness pairs *disjoint* if no two pairs share any (non-split) buckets. We will argue shortly that there always exists a set of b_s disjoint witness pairs, but for now let's assume that and complete the proof that $E_s \leq E^*$.

Let us denote the set of values approximated by the ith disjoint witness pair by the set X_i. Since the buckets obey the Min-Merge property (Lemma 1), we must have

$$\sum_{i=1}^{b_s} e(X_i, 1) \geq \sum_{i=1}^{b_s} e(T''_i, 1) \tag{10}$$

Each of these X_i sets is a subset of some set T_j, where some sets might have multiple X_is as subsets. Let Z be the set of indices j such that $j \in Z$ iff T_j has some X_i set as its subset. Then, using monotonicity we get,

$$\sum_{j \in Z} e(T_j, 1) \geq \sum_{i=1}^{b_s} e(X_i, 1) \tag{11}$$

By the non-negativity and additivity of the error metric, we have

$$E^* \geq \sum_{i \in Z} e(T_i, 1) \tag{12}$$

Using equations 9, 10, 11, 12 we get,

$$E_s \leq E^* \tag{13}$$

Remember that b_s is at most $B-1$. Now we show that our assumption of there being at least b_s disjoint witness pairs is true. Let us say not, and without loss of generality let us say there are exactly $B-2$ disjoint witness pairs. There

are a total of $4B$ buckets, among which there are at most $B-1$ split buckets. Hence we are left with at least $3B+1$ non-split buckets. Among these there are only $B-2$ disjoint witness pairs (by assumption). Hence we are left with at least $3B+1-(2B-4) = B+5$ buckets. But, we have only B optimal buckets, hence by the pigeon hole principle, there must be at least one more disjoint witness pair, which makes our assumption incorrect.

Equations 6, 7 and 13 give us the 2 approximation on the error, we used $4B$ buckets, where B is the optimal, and this completes the proof that the GM algorithm gives a $(2, 4)$ approximation if the error metric is well behaved. ∎

The error analysis of the generic merging scheme holds for all well-behaved error metrics, which include all L_p norms as well Hausdorff distance. In fact, for the L_p norms we can show the following improved bound.

Theorem 2: For the L_p error metric, GENERIC-MINMERGE achieves a $(2^{1/p}, 4)$ approximation.

The proof is omitted due to space constraints.

Remark: A streaming implementation of GM requires a space-efficient implementation of the two key functions: BEST-ADJPAIR and MERGE. In the following, we show that for the two most popular norms (L_2 and L_∞), these functions can be implemented in the streaming model with small memory footprints and can also be extended to amnesic and out-of-order models. On the other hand, we do not know how to realize these functions for other L_p norms in the streaming model: for instance, computing the optimal L_1 approximation of a time series requires maintaining (and updating) the *median* of streaming subsets. Whether one can use *approximations* of the bucket errors and still obtain the global error bound for the generic minmerge remains an interesting open question.

IV. HISTOGRAMS FOR L_2 AND L_∞

To evaluate the functions BEST-ADJPAIR and MERGE in the streaming model, we need to compute the L_2 or L_∞ error for the union of two buckets without explicit knowledge of the values in those buckets. For the L_∞ error, Buragohain et al. [3] show how to implement the greedy bucket-merge in the streaming setting. However, their technique and analysis is limited to L_∞ error only, which can be computed by maintaining the axis-aligned bounding box for PC histogram and (approximate) convex hulls for the PL histogram. Because of their specialized application, Buragohain et al. [3] are able to obtain the following tighter analysis for the greedy bucket-merge:

Theorem 3 ([3]): For the L_∞ error metric,

- the GM algorithm achieves a $(1, 2)$ approximation for the streaming PC histogram using $O(B)$ space and $O(\log B)$ update time for each value.
- the GM algorithm achieves a $(1+\varepsilon, 2)$ approximation for the streaming PL histogram using $O(\varepsilon^{-1/2} B \log(1/\varepsilon))$ memory and $O(\log B + \varepsilon^{-1/2} \log(1/\varepsilon))$ update time for each value.

In the following, we show that our generic framework allows us to extend these results to L_2 norm. In particular, we show that the L_2 error for each bucket can be encoded in $O(1)$ space, and this representation allows us to compute the L_2 error of the bucket obtained by merging two adjacent buckets in $O(1)$ time. Specifically, let us consider two adjacent buckets, representing the sets of items S_1 and S_2, where $S_1 \cap S_2 = \emptyset$. Suppose the sequence of values is $v_1, v_2, \ldots, v_n$, with associated time stamps $t_1, t_2, \ldots, t_n$. Then, it is well-known that the optimal linear approximation under the L_2 error for the sequence $v_1, v_2, \ldots, v_n$ is achieved by the *least squares line* fitting the two-dimensional set of values $(t_1, v_1), (t_2, v_2), \ldots, (t_n, v_n)$. The slope and the intercept of this line can be computed by the following four sums: $\sum v_i$, $\sum v_i^2$, $\sum t_i$ and $\sum v_i t_i$, which can be maintained in $O(1)$ space and updated for the union of two adjacent buckets in $O(1)$ time. The proof of the following can be found in [7] and [24].

Lemma 2: Given two adjacent intervals sets S_1 and S_2 and their optimal least square approximations, we can obtain the optimal least square approximation for $S_1 \cup S_2$ in $O(1)$ time and $O(1)$ space. The exact value of the L_2 norm error of $S_1 \cup S_2$ can be computed from the errors of the buckets for S_1 and S_2 in $O(1)$ time and $O(1)$ space.

With this result, we can evaluate the key functions of the GM algorithm efficiently in the streaming model. Specifically, MERGE can be implemented $O(1)$ time by storing $O(1)$ information about each bucket, while BEST-ADJPAIR function can be implemented using a heap in $O(\log B)$ update time per value. Thus, we have the following result.

Theorem 4: Given a time-series, we can compute a streaming $(\sqrt{2}, 4)$ approximation of its B-bucket PC or PL histogram under the L_2 norm using $O(B)$ space and $O(\log B)$ update time for each value.

V. L_∞ AMNESIC APPROXIMATION

In an *amnesic* (or, time decaying) approximation, recent data are approximated more faithfully than older data. In other words, the approximation error is allowed to grow with the age of the data. The streaming amnesic approximations have received considerable attention lately as histogram summaries have become an important tool for online analysis of data [23], [24]. We show that our generic minmerge framework is well-suited for amnesic approximation as well.

An amnesic representation is controlled by dividing the data into *age groups*, where each age group is assigned an upper bound on the approximation error. We take a general approach and assume that the amnesic representation is specified as a piecewise constant function, which is monotone, meaning that the approximation errors grow with age. In particular, a monotone piecewise constant amnesic function partitions the time series into time-intervals $\{[1, \tau_1], (\tau_1, \tau_2], \ldots, (\tau_{m-1}, \tau_m]\}$, with each interval $(\tau_i, \tau_{i+1}]$ associated with an error tolerance e_{i+1}, where $e_i \leq e_{i+1}$. Thus, the amnesic function $f(x)$ can be described as:

$$f(x) = \left\{ \begin{array}{ll} e_1, & 1 \leq x \leq \tau_1 \\ e_2, & \tau_1 < x \leq \tau_2 \\ \vdots & \\ e_m, & \tau_{m-1} < x \leq \tau_m \end{array} \right\} \qquad (14)$$

See Figure 3 for an example—the top part shows an example stream approximated within the error bounds specified by the amnesic error function shown at bottom.
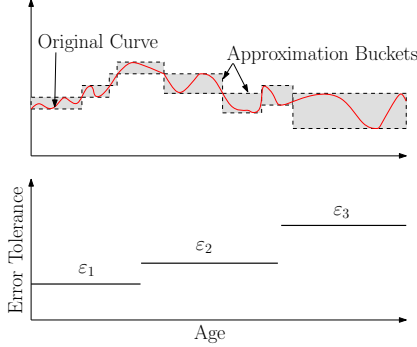

Fig. 3. Amnesic approximation. The top figure shows increasing bucket heights (L_∞ error) with increasing data age, and the bottom figure shows a monotone amnesic error function, with 3 age groups.

Next, we describe the amnesic version of our generic merge algorithm and prove its error bounds. We consider only the L_∞ norm.

A. Amnesic-Merge (AM) Algorithm

Let the number of age groups in the amnesic function be m, where we assume that m is a small constant and $m = o(B)$. Then, our problem is the following: *Given a monotone piecewise constant amnesic function $f()$, and a stream $S = < v_1, v_2, \ldots, v_n >$ of values, maintain an approximation minimizing the number of buckets used such that $|\hat{v}_i - v_i| \leq f(x_i)$ for all i, where $\hat{v}_i$ is the approximated value for v_i.* The main conceptual difficulty in this *amnesic approximation problem* arises from the fact that the age of each item changes every time instant. The simple but key observation in our solution is the fact that only those buckets that cross the age group boundaries require special care; the remaining ones behave as in the standard streaming model. This requires only a minor change to the generic minmerge, and we call this variant the AMNESIC-MERGE (AM) algorithm:

1) Allocate a bucket for the next value in the stream.
2) For each age group with two or more buckets in it, if merging any two adjacent buckets produces error less than or equal to the allowable error in the age group, merge the two buckets.
3) Go to Step 1.

A bucket is in an age group if its starting timestamp lies in the age group. This algorithm is nearly identical to the algorithm proposed in [24] with one minor difference: in that algorithm, the merge *precedes* the allocation of the new bucket, while in our algorithm these steps are reversed.

1) Analysis of the Amnesic-Merge (AM): We first point out the problem formulation of the amnesic approximation is quite different from our earlier problem. There, we wished to minimize the error *given a number B of buckets*. In the AM algorithm, we want to minimize the number of buckets *given error bounds*. We show that if the optimal algorithm uses B^* optimal buckets for a given amnesic function, then the AM algorithm uses no more than $3B^*$ buckets, while achieving the same error guarantees for each age group. Thus, AM is a $(1, 3)$ approximation.

Lemma 3: The bucket approximation produced by the AMNESIC-MERGE algorithm satisfies the L_∞ error bounds for each age group, and uses at most 3 times the optimal number of buckets.

Proof: Let us denote the number of buckets in age group i by b_i', where age group i is the set of values with age in the range $(\tau_{i-1}, \tau_i]$. As mentioned earlier, the buckets can be thought of as a linear partition of the age axis. There are at most $m - 1$ buckets which intersect with the partition obtained by the age groups defined by the amnesic function. Every age group i has at least $b_i' - 1$ non-intersecting buckets. Let us denote the number of non-intersecting buckets associated with age group i by b_i''. Then,

$$b_i'' \geq b_i' - 1 \qquad (15)$$

Assume for now that $b_i'' \geq 2$. Since we merge buckets only if they produce error less than the required error for the age group, the buckets in every age group are within the required error. Let us denote the set of values in age group i approximated by non-intersecting buckets by T_i''. Let us denote the maximum error in the non-intersecting buckets formed by AM algorithm in age group i by $e_m(T_i'', b_i'')$, where b_i'' is the number of buckets used. Also, let us denote the maximum error in the buckets produced by the optimal algorithm in approximating set T_i'' using b_i buckets by $e(T_i'', b_i)$. From the AM algorithm we know that if there was even one more merge in the b_i'' non-intersecting buckets in age group i, the error will be more than the allowable error. Hence,

$$e(T_i'', \lfloor (b_i'' - 1)/2 \rfloor) \geq e_m(T_i'', b_i'' - 1) > e_i \qquad (16)$$

The first part of this equation follows from the fact that the buckets obey the Min-Merge property. This equation implies that even if one bucket is shared across every neighboring age group by the optimal algorithm, with less than $(\sum_i \lfloor (b_i' - 1)/2 \rfloor) - m$ buckets optimal algorithm cannot achieve the desired error in every range. Let $B = (\sum_i \lfloor (b_i'' - 1)/2 \rfloor) - m$. We also know that the merge algorithm with at most $\sum_i (b_i')$ buckets achieves the desired error in every range. Let $B' = \sum_i (b_i')$. Using this and equation 15 we can show that $B' \leq 2B + O(m) \leq 3B$. (The $O(m)$ terms also takes care of the case when $b_i' \geq 3$ does not hold, and since $m = o(B)$, the approximation still holds.) ∎

To implement the merging of two buckets and computing the L_∞ error of a bucket, we need to maintain the convex hull of the values in each bucket. In the worst case, the size of the convex hull for a set of n values is $O(n)$, but we can use the techniques presented in [3] to reduce this size to a user specified constant. We omit the details and point the reader

to [3] for details, and summarize our result in the following theorem.

Theorem 5: The AMNESIC-MERGE algorithm achieves a $(1 + \varepsilon, 3)$ approximation for the amnesic approximation problem and uses $O(\varepsilon^{-1/2} B^* \log(1/\varepsilon))$ space and $O(\log B^* + \varepsilon^{-1/2} \log(1/\varepsilon))$ update time for each value, where B^* is the number of buckets in the optimal solution.

Remark: Palpanas et al. [24] also propose the dual version of amnesic approximation problem described above. In the dual version, we are given an input parameter B, and the constraints on the ratios of the tolerable errors in each age group. It is difficult to achieve bounds on this problem as the number of buckets that needs to be assigned to a contiguous piece of data might change in a non-monotonic manner depending on the incoming data. We believe that there is a negative result for this problem and intend to address this issue in a future study.

VI. OUT-OF-ORDER STREAMS

We now discuss an extension of our GENERIC-MINMERGE that holds for approximating out-of-order streams. The *out-of-order* stream model is motivated by the asynchrony in many distributed and networked data processing applications [8], where data is sensed at dispersed locations and then transmitted to a central node for processing. Due to various networking delays and asynchronous communication, data values may not arrive in the strict time-order in which they were recorded. An approximation scheme for the out-of-order stream should be tolerant of a small number of *gaps* in the arrival sequence. A gap is maximal block of timestamps $[t_i, t_j]$ such that items with timestamps $t_i - 1$ and $t_j + 1$ have arrived but none of the items with timestamp t, for $t_i \leq t \leq t_j$, have arrived. We call the maximal block of timestamps between two consecutive gaps a *fragment*. Figure 4 shows a snapshot of an out-or-order stream with gaps and fragments.
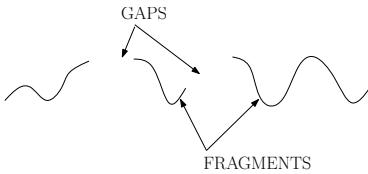

Fig. 4. Snapshot of an out-of-order stream, showing gaps and fragments.

We are interested in PC or PL bucket approximations using L_2 or L_∞ error. A bound on the approximation error means that no bucket can include a gap—the missing value can be arbitrarily far off from the approximation. Thus, it is easy to see that if there are k *gaps* in the data seen so far, then an guaranteed error approximation requires at least $\Omega(min(k, n - k))$ space. In practice, we assume that the number of gaps is small, and show that our approximation uses space only linear in the number of gaps. Our algorithm for the out-of-order streams is called FRAGMENT-MERGE and is described below.

A. The Fragment-Merge (FM) Algorithm

Algorithm 2 gives the pseudocode for the FM algorithm. The value for the constant c is set to 2 for the L_∞ norm, and 4 for the L_2 norm. At a high level, the algorithm works as follows. On the arrival of a new value, a new bucket is allocated. Suppose the timestamp associated with the new value is t_i. Depending on how t_i relates to the gaps and fragments, we can have three cases.

1) A new fragment is formed if the values with timestamps t_{i-1} and t_{i+1} have not been seen so far.
2) The new value is adjacent to exactly one existing fragment—that is, either t_{i-1} or t_{i+1} have been seen, but not both.
3) The new value closes the gap between two fragments—that is, both t_{i-1} and t_{i+1} have been seen.

These three cases are shown in Figure 5. Depending on which category the new value falls in, the FM algorithm performs 0, 1 or 2 merges. The FRAG-BEST-ADJPAIR function returns the two buckets whose merge results in a bucket with least error among all adjacent pairs of buckets in all fragments. We analyze this algorithm in the next section.

Algorithm 2 FRAGMENT-MERGE(S, c, B)

1: $\mathcal{M} = \emptyset$, $k = 0$
2: **for all** v_i **do**
3:     Allocate new bucket u
4:     $\mathcal{M} = \mathcal{M} \bigcup \{u\}$
5:     **if** $|\mathcal{M}| \geq cB + k$ **then**
6:         **if** Exactly one of the neighboring values of v_i has been seen **then**
7:             $\{v_1, v_2\}$ = FRAG-BEST-ADJPAIR(S)
8:             v = MERGE(v_1, v_2)
9:             $\mathcal{M} = \mathcal{M} \setminus \{v_1, v_2\}$
10:            $\mathcal{M} = \mathcal{M} \bigcup \{v\}$
11:         **else if** Two neighboring values of v_i have been seen **then**
12:            $counter = 0$
13:            **while** $counter < 2$ **do**
14:                $\{v_1, v_2\}$ = FRAG-BEST-ADJPAIR(S)
15:                v = MERGE(v_1, v_2)
16:                $\mathcal{M} = \mathcal{M} \setminus \{v_1, v_2\}$
17:                $\mathcal{M} = \mathcal{M} \bigcup \{v\}$
18:                $counter = counter + 1$
19:            **end while**
20:            $k = k - 1$
21:         **else**
22:            $k = k + 1$
23:         **end if**
24:     **end if**
25: **end for**

B. Analysis of Fragment-Merge (FM)

We detail the analysis for the L_∞ norm, and mention briefly the case of the L_2 norm. Because of the out-of-order arrival of items, there are two different notions of time: the timestamp,
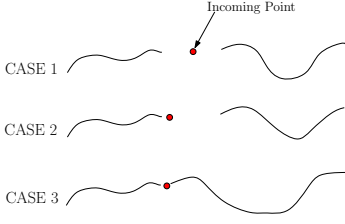
Fig. 5. Three different configurations on the arrival of a new item: creation of a new fragment, extension of an existing fragment, and the merging of two existing fragments.

which represents the time at which data is generated (at the source), and the time at which it is received at the processing node (sink). We use the notation t to denote the source-time and τ to denote the processing or sink-time. Let k_i be the number of fragments at time instant τ_i. The analysis of FM rests on two lemmas: the first one proves that the buckets satisfy the Min-Merge property and bounds the size of approximation; the second does the error analysis.

Lemma 4: At any time τ_i, the FRAGMENT-MERGE algorithm maintains an approximation of size *at most* $2B + k_i$ buckets, where k_i is the number of fragments at time τ_i. After each completed iteration of the algorithm, the buckets obey the Min-Merge property.

Proof: Our proof is by induction, on the length of the time series. For the base case, we have one value, so for any B, we have less than $2B + k_i$ buckets, where $k_i = 1$. We show that the lemma is true for all iterations of the algorithm after the first time instant τ_j where we have $2B + k_j$ buckets. Before the first such time instant the number of buckets is less than $2B + k_i$ for any instant τ_i and the Min-Merge property holds because there have been no merges. We assume that the claim is true for time instant τ_i and show that the claim is true for τ_{i+1}. We separately analyze the three cases mentioned earlier.

- Case 1. *The new value seen creates a new fragment of size 1*: The number of fragments $k_{i+1} = k_i + 1$. For the new value a separate bucket is allocated by the algorithm which has zero error. The number of buckets at this point is $2B + k_i + 1$, which is $2B + k_{i+1}$ and this proves the first part of the claim. Since the Min-Merge property was satisfied at time instant τ_i, it is also satisfied at time instant τ_{i+1} as the new bucket is not merged with any bucket.
- Case 2. *The new value increases the length of exactly one existing fragment by 1*: The number of fragments $k_{i+1} = k_i$. One merge operation is performed, so the number of buckets first increases to $2B + k_i + 1$ (allocation of bucket for new value), and then decreases with the merge to $2B + k_i$, which is the same as $2B + k_{i+1}$. There are two sub cases that can occur:
  - Case a: In the merge operation the new bucket gets merged with adjacent bucket, say v. Let us say that the bucket formed is u. Then, the error of bucket u is no less than the error of bucket v (monotonicity of L_∞ error). Hence, the error in the bucket formed

by further merging of u will be no less than the resultant error if v had been merged. But merging v again would itself result in a bucket with highest error. This is true as Min-Merge property holds at time τ_i. Hence the Min-Merge property holds at time τ_{i+1}.
  - Case b: In the merge operation, the new bucket is not merged. Then let us say buckets u' and v' get merged and the error in the resultant bucket is e. But this implies that any further merge (including merge involving new bucket), will give a bucket with error greater than e, which is the maximum error in the approximation after $(i+1)$th iteration. Hence after this one merge Min-Merge property is satisfied.
- Case 3. *The new value joins two previously disjointed fragments*: This case is similar to Case 2 and its proof is omitted due to lack of space.

This completes the proof. ∎

The second lemma establishes the quality of the approximation formed by FM algorithm with respect to optimal.

Lemma 5: At any time τ_i, the approximation error of the FRAGMENT-MERGE algorithm using $2B + k_i$ buckets is no worse than the optimal approximation error using B buckets, where k_i is the number of fragments at time τ_i.

Proof: Let us denote the number of buckets used by the optimal algorithm for fragment j by b_j, where $1 \leq j \leq k_i$ and $\sum_{j=1}^{k_i} b_j = B$. Let the error associated with the optimal approximation using B buckets be E^*. Let us denote the set of values in fragment j by F_j. Let the error associated with optimal approximation of fragment F_j using b_j buckets be $e(F_j, b_j)$, and hence $E^* = \max_{j=1}^{k_i} e(F_j, b_j)$.

Let us denote the number of buckets used by the FM algorithm for the jth fragment by b'_j, where $1 \leq j \leq k_i$ and $\sum_{j=1}^{k_i} b'_j \leq 2B + k_i$. Let the corresponding error associated with fragment j be $e_m(F_j, b'_j)$. The overall error of the approximation is $E = \max_{j=1}^{k_i} e_m(F_j, b'_j)$. We want to show that $E \leq E^*$. We will show the result by contradiction, and hence let us assume the following,

$$E^* < E \qquad (17)$$

For the above equation to be true, there must exist at least one u, such that $b'_u < 2b_u$. This follows from the fact that every fragment follows the Min-Merge property (Theorem 3). Without loss of generality, let this be the fragment which results in the maximum error i.e.

$$E = e_m(F_u, b'_u) \qquad (18)$$

Then, there must also be at least one fragment v, such $b'_v > 2b_v + 1$ (Pigeon hole principle). Now, consider the last algorithm iteration which resulted in the number of buckets for u to go from $b'_u + 1$ to b'_u. Note that there could be multiple such merges and we consider the last such merge before time instant τ_i, say at time instant τ_j. At time instant τ_j, fragment v has either more than $2b_v$ buckets, or less (equal to) than $2b_v$ buckets. We handle the arguments for these two cases separately.

- Case 1: If v has more than $2b_v$ buckets, say it has b_v'' buckets, and let the fragment itself at that time be F_v' where $F_v' \subseteq F_v$, then

$$e_m(F_v', b_v'' - 1) \le e(F_v, b_v) \le E^* \qquad (19)$$

This follows from the fact that the fragments obey Min-Merge property and the buckets obeying Min-Merge gives a $(1, 2)$ approximation for L_∞ error (Theorem 3). But, FM algorithm merges u before v, so error in fragment v after one merge, is no less than the error in fragment u when it has b_u' buckets,

$$e_m(F_v', b_v'' - 1) \ge e_m(F_u, b_u') \qquad (20)$$

Combining equations 18, 19 and 20, we get $E^* \ge E$, which contradicts our assumption.

- Case 2: At time instant τ_j, fragment v has less than (or equal to) $2b_v$ buckets. Then there must be a new value at some other time instant, say τ_l, where τ_l is after τ_j which results in v having more than $2b_v + 1$ buckets (after completed iteration of FM merge). This value can be either one which is next to one end of a fragment (as in Case 2, Lemma 4) or it can be the value which joins two fragments (as in Case 3, Lemma 4). We leave out these cases in the interest of space.

This concludes the proof. ∎

With the natural assumption that $k_i \le B$, we get that $2B + k_i \le 3B$. The analysis above requires us to maintain the convex hull of the values in each bucket in order to determine the approximation error in the bucket. Using the approximate convex hulls as in [3], we can get an ε approximation. Due to space constraints, we skip those details, and state the main result for this section.

Theorem 6: The FRAGMENT-MERGE algorithm achieves a $(1 + \varepsilon, 3)$ approximation for the piecewise linear approximation of an out-of-order stream under the L_∞ norm, using $O(\varepsilon^{-1/2} B \log(1/\varepsilon))$ space and $O(\log B + \varepsilon^{-1/2} \log(1/\varepsilon))$ update time for each value.

For piecewise constant approximation, the space and time bounds are $O(B)$ and $O(\log B)$ respectively, for a $(1, 3)$ approximation for the L_∞ norm. We can show a similar result for the L_2 approximation.

Theorem 7: The FRAGMENT-MERGE algorithm gives a $(\sqrt{2}, 5)$ approximation for the piecewise constant (or piecewise linear) approximation of an out-of-order stream under the L_2 norm using $O(B)$ space and $O(\log B)$ update time for each value.

VII. SIMULATION RESULTS

In this section, we report on the simulation-based performance evaluation of our three main algorithms, for computing histogram for streams (L_2 approximation), histograms for amnesic approximation (L_∞ approximation), and the out-of-order stream approximation. In particular, we implemented the following algorithms, including some optimal schemes for benchmarking. (All the code is written in Python or C++, and is available at www.cs.ucsb.edu/~foschini/files/ooodelay.zip, along with the instructions to run it. All the benchmarks reported in this paper are run on an Intel Core Duo Processor (@ 2.00GHz) equipped with 2GB RAM.)

- GENERIC-MINMERGE, AMNESIC-MERGE, FRAGMENT-MERGE: We implemented all three main variants of our generic scheme (GM, AM and FM).
- *Dynamic Programming*: We implemented the dynamic programming scheme described in [14] for computing the optimal error histogram approximation. We only use the optimal histogram for error comparison, and so techniques for improving the space bounds are irrelevant in this case [10].
- SPACEAPXWAVEHIST *Algorithm*: We also implemented Guha's algorithm described in [10], which currently has the best theoretical approximation bound[1]. (More details about this can be found in Section VII-A.3. For performance reasons, this algorithm is implemented in C++.)
- *Optimal Amnesic Approximation*: We implemented the optimal (offline) amnesic approximation scheme. This algorithm processes values in order of decreasing (source) timestamps. Initially the algorithm starts with an empty bucket. Values are added to the bucket as long as the error in the bucket is less than the tolerable error for the highest timestamp value in the bucket. If not, a new bucket is started, and so on.

In our experiments, we used a variety of datasets from various applications. From these, 5 datasets are shown in Figure 6.

- STEAMGEN [18]: The set has 9600 measurements of the drum pressure, taken every 3 seconds, from a model of a steam generator at Abbott Power Plant in Champaign IL.
- RANDOM [18]: This is the classical random walk data used in many papers on time series indexing. This set consists of 65536 values.
- DJIA: This set has the value of the Dow Jones Industrial Average index starting from year 1900. We obtained this dataset from StatLib and it consists of 25737 values.
- HUM: This set consists of measurements from humidity sensors monitoring servers in a large data center. It has 69652 values, representing humidity measurements every 30 seconds.
- TEMP: This set of 93861 values consists of temperature readings taken once every 5 minutes from the Davis weather station in Amherst, Massachusetts.

We begin by evaluating the quality of approximations produced by the GM algorithm and demonstrating the scalability of the scheme in Section VII-A. In Section VII-B we compare the performance of AM algorithm with the optimal offline algorithm. And finally, in Section VII-C we demonstrate the quality of approximations produced by the FM algorithm for out of order streams.

---

[1][11] provides similar bounds as [10], the postprocessing time is nearly the same for both the algorithms and this is what we compare with GM and hence we implement only SpaceApxWaveHist.
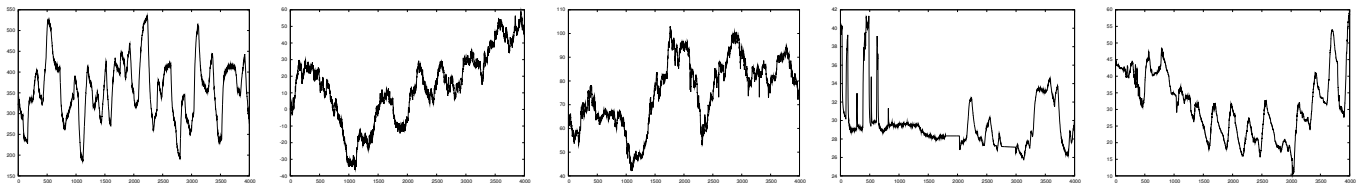
Fig. 6. Left to Right: STEAMGEN, RANDOM, DJIA, HUM and TEMP

A. Minmerge for the L_2 Histograms

We evaluate the quality and scalability of L_2 histogram approximation by our generic minmerge algorithm GM. (A minmerge type algorithm for the L_∞ histogram was already presented in [3].) Curiously, algorithms that use greedy bucket-merges for the L_2 streaming histograms have been used widely before, *but with no worst-case error bounds* [16], [24]. In fact, there is only one minor difference in the details of our GM and the GRAP-R algorithm [16]: we do the merge *after allocating the new bucket* for the new item, while GRAP-R does the merge first. Empirically, this minor change has little effect on the performance of the algorithm, but it does allow us to prove a worst-case error bound for GM.

1) The Approximation Quality: We compare the quality of the approximations produced by the GM algorithm with the dynamic programming scheme. Because dynamic programming has quadratic time complexity, it is extremely slow for large data sets, so we ran our test using $n = 4000$ values. The results are shown in Figure 7, which plots the L_2 approximation error vs. the number of buckets.

Our results were virtually identical across all 5 data sets, so this figure shows a representative sample for the HUM data set. In the experiment, we vary the buckets from 50 to 200. (Since the slowness of the dynamic programming limited us to $4K$ size data sets, there did not seem to be a point in increasing the number of buckets to larger values. In later experiments on larger data sets, we allow larger values of B.)

The total error predictably decreases with an increasing number of buckets. Recall that Theorem 4 gives a $(\sqrt{2}, 4)$ approximation guarantee, meaning that the error of the GM histogram using $4B$ buckets is no worse than $\sqrt{2}$ times the error of the optimal B-bucket histogram. Thus, Figure 7 shows 3 bars for each value of B: the leftmost bar is the optimal error; the middle bar is the error of the GM using $4B$ buckets; and the rightmost bar is the *error of the GM using just B buckets.* The left part shows the results for PC histograms and the right part shows the results for PL histograms. These results show that GM's performance is much better than the worst-case guarantee: with $4B$ buckets, the error is about a third of the optimal error, and even with the same memory as optimal (B buckets), the error of the GM approximation is only slightly worse (never more than 30%).

In fact, for the rest of the empirical evaluations, even for AM and FM algorithms, we always use only B buckets for our algorithms, and not the constant-factor-times B buckets required by our theorems.

2) Scalability of GM algorithm: In this experiment we evaluate the scalability (processing time) of the GM algorithm as a function of the size of the stream. We use a large dataset
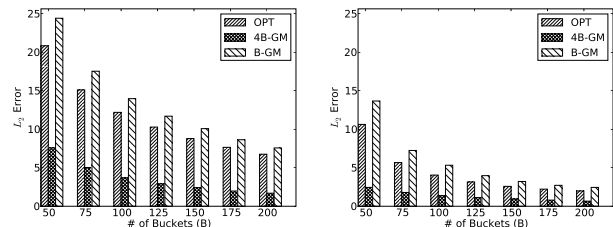

Fig. 7. The approximation quality as a function of the number of buckets, for the HUM data. The results shown are for PC histograms (left) and PL histograms (right). B-GM refers to the B-bucket approximation, while $4B$-GM refers to the $4B$-bucket approximation (as needed for the theorem).
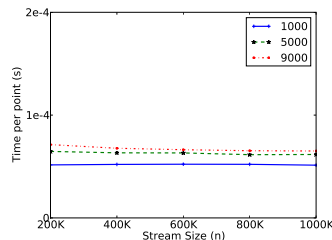

Fig. 8. Per-item processing time vs. the size of the time stream, for $B = 1000, 5000, 9000$ for the PRECIP dataset.

to show the scalability, we call this dataset PRECIP. The archive [27] contains the average monthly and annual gage-corrected precipitation recorded by $26,858$ stations worldwide. Station averages of precipitation are interpolated to a 0.5 degree by 0.5 degree of latitude/longitude grid, where the grid nodes are centered on 0.25 degree, resulting in 259200 geographically distinct measurements. The dataset PRECIP here analyzed contains $1M$ values from the previous archive obtained by concatenating the measurements for all stations for several months after having sorted the archive on (latitude, longitude) of the stations.

We vary the size of the dataset from $200K$ to $1M$ in intervals of $200K$ and plot the time taken per data point by the GM algorithm for different number of buckets: $B = 1000, 5000, 9000$. The results are shown in Figure 8 for PC approximation. As predicted by theory, the GM algorithm scales linearly with the size of the stream and shows a negligible effect of the number of buckets. The results for PL approximations are the same, just that it takes more processing time per data point (roughly a factor of 3 more).

3) Comparison with SPACEAPXWAVEHIST: The current best theoretical bound for the L_2 histogram approximation is due to Guha [10]: his SPACEAPXWAVEHIST algorithm achieves $(1+\varepsilon, 1)$ approximation in $O(n+B^3\varepsilon^{-2}\log^2 n)$ time using $O(B\varepsilon^{-2}\log n\log\varepsilon^{-1})$ space. Our GM algorithm only

guarantees a $(\sqrt{2}, 4)$ approximation, but has the advantage of utter simplicity, as well as generalizability to models such as amnesic and out-of-order streams. A second, and perhaps, more significant advantage of our scheme may be that it maintains a *query-ready* indexable structure at all times. In other words, our time series approximation is directly represented as a histogram that can be queried at any time, with no additional overhead. This is in contrast to schemes such as Guha's [10] whose synopses consist of wavelet coefficients that must be processed to build a B bucket histogram. The overhead of this post-processing requirement for the guarantee can be significant—for instance, the technique in [10] requires $O(B^3 \log^2 n)$ time to build the B bucket histogram before a query can be answered. Thus, the two schemes, ours and Guha's, have different advantages—the latter has the best provable worst-case error guarantee while the former maintains an online query-ready histogram—so it is worth comparing the two in a simulation.

Since a public implementation of SPACEAPXWAVEHIST was no available, we implemented it ourselves. After some optimizations and parameter tuning, the observed performance seems to match with what is reported in [10], so we feel that our comparisons should be fair. Figures 9 shows the main results of our experiments (for HUM dataset): the error in the approximation of the GM algorithm (using the same amount of memory) is close to that achieved by SPACEAPXWAVEHIST (the left figure), however the running time of that algorithm is significantly worse than GM (as predicted by theory), as shown in the right figure. Results for other datasets are virtually the same. The SPACEAPXWAVEHIST scheme uses memory in two places, wavelet synopsis and partial dynamic programming table. *Rather than optimize the memory distribution across these two components, we allocated to each part as much memory as our GM scheme requires, thereby potentially giving* SPACEAPXWAVEHIST *twice the memory*. More details of the experiment are omitted due to lack of space.
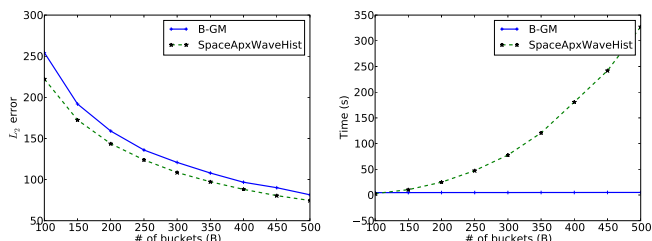

Fig. 9. GM vs. SPACEAPXWAVEHIST. The approximation error (left) of B-bucket GM is comparable to the B-bucket SPACEAPXWAVEHIST, with the latter given twice the working space. The running time of GM is significantly faster, due to the $O(B^3)$ term in SPACEAPXWAVEHIST.

B. The Amnesic Time Series Approximation

In order to set up a natural aging function, we divided the data into 4 groups, with approximation error set to 1%, 2%, 3% and 4%, respectively, of the maximum data range, where the data range is defined as the difference between the maximum and the minimum value. The spread of the groups is exponential. In other words oldest $n/2$ are allowed error 4%,
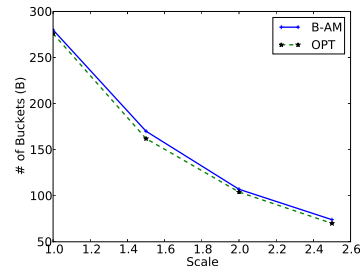

Fig. 10. The number of buckets required by the AM (B-AM) algorithm as compared to optimal for different scaling factors for the DJIA dataset.

the next $n/4$ values are allowed error 3% and so on. Thus, we have a monotone piecewise constant amnesic function with 4 steps. We also used a scaling function to adjust the values of the allowable errors, while keeping their ratio the same. In other words, by using a scaling factor of 2, we get the approximation errors for the 4 age groups to be 2.0%, 4.0%, 6.0% and 8.0%.

The approximation quality for the amnesic histogram is given by the number of buckets used by the AM algorithm, so we compare this to the optimal amnesic approximation. We show the results of the streaming amnesic approximation for one representative data set (DJIA), which contains $25K$ data points. Our results for PC approximations are shown in Figure 10. One can see that the number of buckets used by the AM algorithm closely track the optimal number of buckets and typically within a factor 1.2 of the optimal number of buckets. The results for the other datasets for both PC and PL approximations are similar; the factors go up to 1.3.
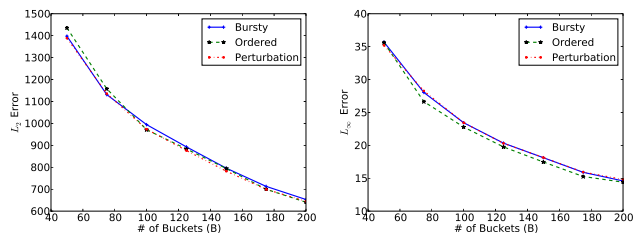
C. Performance of the Fragment-Merge (FM)

Fig. 11. The quality of approximation produced for the Bursty, Perturbation and the Ordered stream models for L_2 (left) and L_∞ (right) error metrics. The FM algorithm adapts very well for out of order arrivals.

Our last set of experiments analyze the performance of the FRAGMENT-MERGE algorithm for approximating out-of-order streams. We first describe our data model for generating a realistic out-of-order stream. There are two aspects to this: how frequently an item appears out of order, and how far from its true position it reappears. Let us first consider the second question. Using the Internet as a guide, we envision two different models for packet delay: a *bursty* model, which may account for buffer overflow at a node causing many data values to be lost and retransmitted, and a *perturbation* model, which may account for the multi-path routing phenomenon in networking, causing packets to arrive slightly out of order. For

the first question, we use a correlated random model to decide which data value to delay.

In more details, we construct an out-of-order stream from an ordered stream statistically in the following manner. The ith item in the stream is reordered (delayed) with probability p_i, given as follows:

$$p_i = \frac{(1-\chi) \cdot \rho + \chi \cdot I_{i-1}}{I_{i-1} + \rho}, \qquad (21)$$

where ρ is the independent probability for an item to be reordered, χ is the correlation probability that accounts for the previous item being reordered, and I_{i-1} is the indicator variable which is either 1 or 0 depending on whether the $(i-1)$th value was reordered or not. In order to simulate this, for the ith value we flip a coin biased with probability p_i. Once an item is taken out, it is reinserted later in the stream with a delay uniformly distributed between 0 and L (lag). Thus, our reordering strategy is completely described by the parameters χ, ρ and L. The bursty model is simulated with small value of ρ, and large values for χ and L. The perturbation model is simulated with large value of ρ, and small values of χ and L. (Due to space constraints, we omit the details.)

We evaluate the error performance of the FM algorithm by comparing it with the approximation obtained by the GM algorithm for the ordered version of the stream. Figure 11 shows the results of this experiment for the DJIA data, where the out-of-order streams are generated both with the bursty and the perturbation models, and the plot shows the error of the approximation vs. the bucket parameter B.

The figure on the left shows the L_2 approximation, while the one on the right shows the L_∞ approximation. In both cases, it is easy to see that the error of the FRAGMENT-MERGE approximation is very close to the error of the GM approximation for the ordered version of the stream. In some cases, the FM error approximation is actually smaller than the GM approximation, but that is explained by the fact the FM approximation uses a maximum of $B + k$ buckets, where k is the number of gaps in the stream, while the ordered stream uses only B buckets. In this set of experiments, the maximum number of gaps for the bursty model is 84, and for the perturbation model it is 23.

We also ran scalability for experiments for both AM and FM algorithms. The results for these are very similar to the GM algorithm, though individual per item processing times are slightly slower for both AM and FM as compared to GM.

VIII. CONCLUSIONS AND FUTURE WORK

We presented a generic framework for online approximation of time-series data that yields a unified set of algorithms for several popular models: data streams, amnesic approximation, and out-of-order stream approximation. Our framework essentially develops a popular greedy method of bucket-merging into a more generic form, for which we can prove space-quality approximation bounds. When specialized to piecewise linear bucket approximations and commonly used error metrics, such as L_2 or L_∞, our framework leads to provable error bounds where none were known before, offers new results, or yields simpler and unified algorithms. The conceptual simplicity of our scheme translates into highly practical implementations, as borne out in our simulation studies: the algorithms produce near-optimal approximations, require very small memory footprints, and run extremely fast.

REFERENCES

[1] M. Abam, M. Berg, P. Hachenberger, and A. Zarei. Streaming algorithms for line simplification. In *SOCG*, 2007.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, 2002.

[3] C. Buragohain, N. Shrivastava, and S. Suri. Space efficient streaming algorithms for the maximum error histogram. In *ICDE*, 2007.

[4] C. Busch and S. Tirthapura. A deterministic algorithm for summarizing asynchronous streams over a sliding window. In *STACS*, 2007.

[5] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *The VLDB Journal*, 10(2-3), 2001.

[6] K. Chan and A. Fu. Efficient time series matching by wavelets. In *ICDE*, 1999.

[7] Y. Chen, G. Dong, J. Han, B. Wah, and J. Wang. Multidimensional regression analysis of time-series data streams. In *ICDE*, 2002.

[8] G. Cormode, F. Korn, and S. Tirthapura. Time-decaying aggregates in out-of-order streams. In *PODS*, 2008.

[9] A. Gilbert, Y. Kotidis, S. Guha, S. Muthukrishnan, et al. Fast small-space algorithms for approximate histogram maintenance. In *STOC*, 2002.

[10] S. Guha. On the space—time of optimal, approximate and streaming algorithms for synopsis construction problems. *The VLDB Journal*, 17(6):1509–1535, 2008.

[11] S. Guha. Tight results for clustering and summarizing data streams. In *Departmental Papers, Department of Computer and Information Science, Univeristy of Pennsylvania*, 2009.

[12] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *STOC*, 2001.

[13] S. Guha, N. Koudas, and K. Shim. Approximation and streaming algorithms for histogram construction problems. *ACM TODS*, 31(1):396–438, 2006.

[14] H. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, et al. Optimal histograms with quality guarantees. In *VLDB*, 1998.

[15] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD*, 2001.

[16] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *In ICDM*, 2001.

[17] E. Keogh and M. Pazzani. A simple dimensionality reduction technique for fast similarity search in large time series databases. In *PADKK*, 2000.

[18] E. Keogh, X. Xi, L. Wei, and C. Ratanamahatana. The ucr time series classification/clustering homepage. 2006.

[19] F. Korn, H. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *SIGMOD*, 1997.

[20] J. Li, K. Tufte, V. Shkapenyuk, V. Papadimos, T. Johnson, and D. Maier. Out-of-order processing: a new architecture for high-performance stream systems. *Proc. VLDB Endow.*, 1(1), 2008.

[21] M. Li, M. Liu, L. Ding, E. Rundensteiner, and M. Mani. Event stream processing with out-of-order data arrival. In *ICDCS Workshop*, 2007.

[22] S. Muthukrishnan and S. Strauss. Approximate histogram and wavelet summaries of streaming data. In *DIMACS TR 52*, 2003.

[23] S. Nath. Energy efficient sensor data logging with amnesic flash storage. In *IPSN*, 2009.

[24] T. Palpanas, M. Vlachos, E. Keogh, D. Gunopulos, and W. Truppel. Online amnesic approximation of streaming time series. In *ICDE*, 2004.

[25] D. Rafiei and A. Mendelzon. Similarity-based queries for time series data. In *SIGMOD*, 1997.

[26] P. Tucker, D. Maier, T. Sheard, and L. Fegaras. Exploiting punctuation semantics in continuous data streams. *IEEE Trans. on Knowl. and Data Eng.*, 15(3), 2003.

[27] C. Willmott and K. Matsuura. Global air temperature and precipitation: Regridded monthly and annual climatologies. In *http://climate.geog.udel.edu/*.