ASAP: Automatic Smoothing for Attention Prioritization in Streaming Time Series Visualization

Kexin Rong, Peter Bailis
Stanford InfoLab

ABSTRACT

Time series visualization of streaming telemetry (i.e., charting of key metrics such as server load over time) is increasingly prevalent in recent application deployments. Existing systems simply plot the raw data streams as they arrive, potentially obscuring large-scale deviations due to local variance and noise. We propose an alternative: to better prioritize attention in time series exploration and monitoring visualizations, smooth the time series as much as possible to remove noise while still retaining large-scale structure. We develop a new technique for automatically smoothing streaming time series that adaptively optimizes this trade-off between noise reduction (i.e., variance) and outlier retention (i.e., kurtosis). We introduce metrics to quantitatively assess the quality of the choice of smoothing parameter and provide an efficient streaming analytics operator, ASAP, that optimizes these metrics by combining techniques from stream processing, user interface design, and signal processing via a novel autocorrelation-based pruning strategy and pixel-aware preaggregation. We demonstrate that ASAP is able to improve users' accuracy in identifying significant deviations in time series by up to 38.4% while reducing response times by up to 44.3%. Moreover, ASAP delivers these results several orders of magnitude faster than alternative optimization strategies.

1. INTRODUCTION

Data volumes continue to rise, fueled in large part by an increasing number of automated sources, including sensors, processes, and devices. For example, each of LinkedIn, Twitter, and Facebook reports that their production infrastructure generates over 12M events per second [16,49,62]. As a result, the past several years have seen an explosion in the development of data platforms for managing, storing, and querying large-scale data streams of time-stamped data—i.e., time series—from on-premises databases including InfluxDB [6], Ganglia [3], Graphite [5], OpenTSDB [9], Prometheus [10], and Facebook Gorilla [49], to cloud services including DataDog [2], New Relic [8], AWS CloudWatch [1], Google Stackdriver [4], and Microsoft Azure Monitor [7]. These database engines provide application writers, site operations, and "DevOps" engineers a means of performing monitoring, health checks, alerting, and analysis of unusual events such as failures [19, 30].

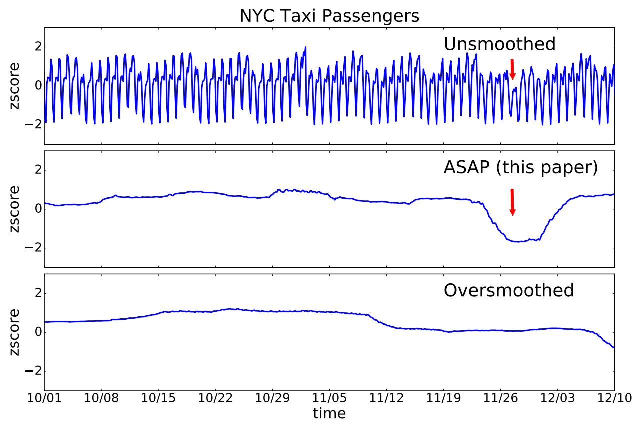


Figure 1: Normalized number of NYC taxi passengers over 10 weeks.[1] From top to bottom, the three plots show the hourly average (unsmoothed), the weekly average (smoothed) and the monthly average (oversmoothed) of the same time series. The arrows point to the week of Thanksgiving (11/27), when the number of passengers dips. This phenomenon is most prominent in the smoothed plot produced by ASAP, the subject of this paper.

Today, these engines have automated and optimized common tasks in the storage and processing of large-scale time series; however, they are surprisingly less well optimized for the visualization of time series. That is, in conversations with engineers using time series data and databases in cloud services, social networking, industrial manufacturing, electrical utilities, and mobile applications, we learned that many production time series visualizations (i.e., "dashboards") simply display raw data streams as they arrive. Engineers reported this display of raw data can be a poor match for production scenarios involving data exploration and debugging. That is, as data arrives in increasing volumes, even small-scale variations in data values can obscure overall trends and behavior.

For example, an electrical utility employs two staff to perform 24-hour monitoring of generators via time series visualization. It is critical that these staff quickly identify any systematic shifts, even those that are "sub-threshold" with respect to a critical alarm. Unfortunately, these sub-threshold events are easily obscured due to short-term fluctuations in the raw signal.

The resulting challenge in time series visualization at scale is presenting the *appropriate* plot that prioritizes the users' attention by highlighting significant deviations. To illustrate this challenge

[1]Here and later in this paper, the plots show z-scores [38] instead of raw values. When comparing across plots, z-scores provide a means of normalizing the visual field across plots while highlighting large-scale trends.

using non-proprietary data, consider the time series depicted in Figure 1. The top plot shows raw data: an hourly average of the number of NYC taxi passengers over 75 days in 2014 [39]. Daily fluctuations dominate the visual field, obscuring a significant long-term deviation: the number of taxi passengers experienced a sustained dip during the week of Thanksgiving. Ideally, we would smooth the local fluctuations to highlight this deviation (Figure 1, middle). However, if we smooth too aggressively, we may hide this trend entirely (Figure 1, bottom).

Our answer to this challenge is to smooth time series visualizations as much as possible while preserving large-scale deviations. This leads to two key questions. First, how can we quantitatively assess the quality of a given visualization in prioritizing users' attention towards significant deviations? Second, how can we use such a quantitative metric to produce a high-quality plot quickly and at large scale? In this paper, we address both questions through a new time series visualization operator, called ASAP (Automatic Smoothing for Attention Prioritization), which quantifiably improves end-user accuracy and speed in identifying significant deviations in time series, and is highly optimized to execute at scale.[2]

To address the first question of quantitative metrics for prioritizing attention, we combine two statistics. First, we measure the smoothness of a time series visualization using the variance of first differences [20], or the difference between consecutive points in the series. By applying moving averages of increasing sizes, we can increasingly smooth the plot. However, as we have illustrated in Figure 1, it is possible to oversmooth and obscure the trend entirely. Second, to prevent oversmoothing, we introduce a constraint based on preserving the kurtosis [24]—a measure of the "outlyingness" of a distribution—of the raw time series, thus preserving its structure. This kurtosis measure can also be used to determine when *not* to smooth (e.g., when the plot contains a few well-defined outlying regions). ASAP combines these metrics to capture both our goals of smoothness and preservation of systematic deviation. We demonstrate the utility of this novel combination via two user studies (total 270 users): compared to displaying raw data, smoothing visualizations in accordance with these metrics improves users' ability to identify temporal deviations in time series by up to 38.4% and decreases the response time by up to 44.3%.

Given these metrics, we develop ASAP, the first streaming analytics operator that automatically smooths time series to highlight large-scale deviations. The ASAP operator ingests a time series data stream and outputs a stream of visualizations that maximizes smoothness while preserving kurtosis. That is, given a duration of time to visualize (e.g., the past 30 minutes of a time series), ASAP automatically selects and applies smoothing parameters on users' behalf, obviating the need for extensive parameter tuning.

There are three major challenges in enabling this automated, efficient smoothing. First, our target workloads exhibit extreme data volumes—up to millions of events per second, requiring ASAP to be scalable. As we demonstrate, an exhaustive search over even 1M points can require over an hour to render a single plot. Second, to support interactive visualization, ASAP must be able to refresh quickly, updating visualizations promptly as new data arrives. We target sub-second response time. Third, appropriate smoothing parameters may change over time: a high-quality parameter choice for one time period may oversmooth or undersmooth in another. Thus, ASAP must adapt its smoothing parameters in response to changes in the data stream.

To address these challenges, ASAP combines techniques from stream processing, user interface design, and signal processing. First, to scale to large volumes, ASAP pushes constraints regarding the target end-user display into its search procedure. ASAP exploits the fact that its results are designed to be displayed in a fixed number of pixels (e.g., maximum 1334 pixels at a time on the iPhone 7) and uses the target resolution as a natural lower bound—there is often little benefit in searching for parameter settings that would result in a resolution greater than the target display size. ASAP accordingly pre-aggregates data, thus reducing the input space and improving rendering time. Second, ASAP further prunes the search space by performing binary search for aperiodic data, and by searching only for time windows capturing periodicity (i.e., time lag with high autocorrelation) for periodic data; we demonstrate both analytically and empirically that this search strategy leads to smooth aggregated series. Third, to enable high-volume processing while quickly responding to changes in the data stream, ASAP avoids recomputing smoothing parameters upon the arrival of each new data point. Instead, ASAP re-renders plots on human-observable timescales by leveraging techniques from partial view materialization.

In total, ASAP achieves its goals of efficient, parameter-free smoothing by treating visualization not as a post-processing step in the user interface but as a critical consideration in stream processing. As we demonstrate empirically, this co-design yields quality results, quickly and without manual tuning. We have implemented ASAP as a time series explanation operator in the MacroBase fast data engine [18], and as a Javascript library. ASAP demonstrates order-of-magnitude improvements in runtime over alternative strategies while retaining high-quality smoothing for visualizations that users empirically prefer.

In summary, we make the following contributions:

- ASAP, the first stream processing operator for automatically smoothing time series to reduce local variance while highlighting large-scale deviations in visualization.
- Three optimizations for improving ASAP's execution speed that leverage *i.*) end-device resolution in pre-aggregation, *ii.*) autocorrelation to exploit periodicity, *iii.*) and partial materialization for streaming updates.
- A quantitative evaluation demonstrating ASAP's ability to improve user accuracy and response time and deliver order-of-magnitude performance improvements during search compared to alternatives.

The remainder of this paper proceeds as follows. Section 2 provides additional background regarding our target use cases and surveys related work. Section 3 formally introduces ASAP's problem definition and quantitative target metrics. Section 4 presents ASAP's search strategy and optimizations, as well as how to perform streaming execution. Section 5 evaluate ASAP's visualization quality through two user studies, and its performance on a range of synthetic and real-world time series. Finally, Section 6 concludes.

2. ARCHITECTURE AND GOALS

ASAP provides analysts and system operators with an effective and efficient streaming operator for highlighting large-scale deviations in time series visualizations. In this section, we describe ASAP's architecture and goals, and discuss related work.

2.1 ASAP Architecture

ASAP is a dataflow operator for time series. Given an input time series (i.e., set of temporally ordered data points) and target interval for visualization (e.g., the last twelve hours of data), ASAP returns a transformed, smoothed time series (e.g., also of twelve hours, but with a smoothing function applied). In the streaming setting, as new data points arrive, ASAP continuously smooths each fixed-size

[2]Demo and code available at `http://futuredata.stanford.edu/ASAP/`

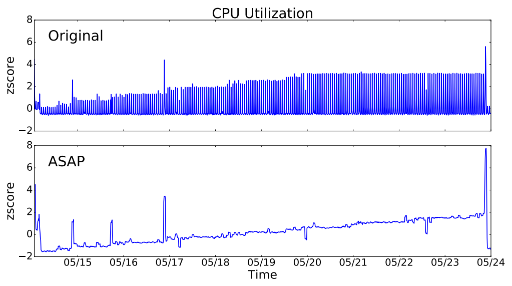


Figure 2: Average server CPU usage across a given cluster over ten days [39], shown in a 10 minute average and an hourly average. The continued increase in CPU usage starting from May 20th is obscured by frequent fluctuations in the original time series.

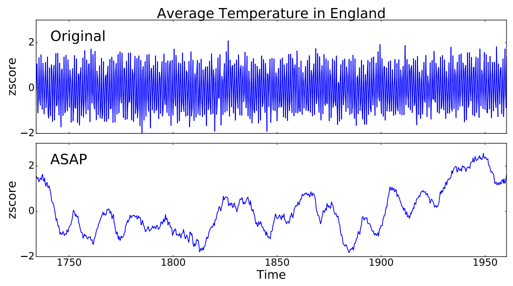


Figure 3: Temperature in England from 1723 to 1970, shown in 2-month average and 23-year average [32]. Fluctuations in the original time series obscure the overall the trend unless smoothed.

time interval, producing a sequence of smoothed time series. Thus, ASAP acts as a transformation on fixed-size sliding windows and over a single time series. When ASAP users change the range of time series to visualize (e.g., zoom-in, zoom-out, scrolling), ASAP re-renders the smoothed output in accordance with the new range. Optionally, for efficiency, ASAP allows users to specify a target display resolution (in pixels) and a desired refresh rate (in seconds).

ASAP can run either server-side or client-side. For resource-constrained clients, or for servers with a large number of visualization consumers, ASAP can execute on the server, sending clients the smoothed stream; this is the execution mode that MacroBase adopts, and MacroBase's ASAP implementation is portable to existing stream processing engines. For lower data volumes and easy integration with web-based front-ends, ASAP can also execute on the client; we provide a JavaScript library for doing so.

ASAP acts as a building block in time series visualization. It can ingest and process raw data from time series databases such as InfluxDB, as well as from visualization clients such as plotting libraries and frontends. For example, when building a monitoring dashboard, a DevOps engineer could employ ASAP and plot the smoothed results in his metrics console, or, alternatively, overlay the smoothed plot on top of the original time series. ASAP can also post-process outputs of time series analyses including motif discovery, anomaly detection, and clustering [36,37,42,64]: given a single time series as output from each of these analyses, ASAP can smooth it before presenting to an end-user.

To further illustrate ASAP's potential uses in prioritizing attention in time series, we introduce two additional use cases:

Datacenter Monitoring. An on-call datacenter operator is paged at 4AM due to an Amazon CloudWatch alarm on a sudden increase in CPU utilization on her Amazon Web Service cloud instances. After reading the alert message, she accesses her cluster telemetry plots that include CPU usage over the past ten days on her smartphone to obtain a basic understanding of the situation. However, the smartphone's display resolution is too small to display all 4000 readings; as a result, the lines are closely stacked together in the plot, making CPU usage appear stable before the alert (Figure 2, top).[3] Unable to obtain useful insights from the plot, the operator must rise from bed and begin checking server logs directly to diagnose the issue. If she were to instead apply ASAP, the continued rise in CPU usage from May 20th would no longer be hidden by noise.

[3]This plot is inspired by an actual use case we encountered in production time series from a large cloud operator; high frequency fluctuations in the plot made it appear that a server was behaving very differently, when in fact, its overall (smoothed) behavior was similar to others in the cluster. We do not include the original data here to preserve data privacy.

Historical Analysis. A researcher interested in climate change examines a data set of monthly temperature in England measured over 200 years. When she initially plots the data to capture the long-term trends, the plots span over five screen heights.[4] Instead of having to scroll to compare temperature in the 1700s with the 1930s, she decides to plot the data herself to fit the entire time series onto one screen. Now, in the re-plotted data (Figure 3, top), seasonal fluctuations each year obscure the overall trend. Instead, if she were to use ASAP instead, she would see a clear trend of rising temperature in the 1900s (Figure 3, bottom).

We provide additional examples of raw time series and their automatically smoothed counterparts in Appendix E.

2.2 Related Work

ASAP is the first streaming analytics operator for automatically smoothing time series to reduce local variance while highlighting large-scale deviations for visualization. Its design draws upon work from several domains including stream processing, data visualization, and signal processing.

Time Series Visualization. Data visualization management systems that automate and recommend visualizations to users have recently become a topic of active interest in the database and human-computer interaction community [63]. Recent systems including SeeDB [48], Voyager [61], and ZenVisage [53] focus on recommending visualizations for large-scale data sets, particularly for exploratory analysis of relational data [45]. In this paper, we focus on the visualization of deviations within time series.

Within the time series literature, which spans dimensionality reduction [35], information retrieval [29], and data mining [12,27,44], visualization plays an important role in analyzing and understanding time series data [26]. There are a number of existing approaches to time series visualization [13]. Perhaps most closely related is M4 [33], which downsamples the original time series while preserving the shape—a perception-aware procedure [25]. Like M4, ASAP aggregates the time series, but ASAP's objective function is to *smooth* rather than *preserve* the original shape. In fact, ASAP can work with M4 to both capture the shape and highlight systematic deviations of the original signal efficiently.

Signal Processing. Noise reduction is a classic and extremely well-studied problem in signal processing. Common reduction techniques include the wavelet transform [23], convolution with smoothing filters [21,51], and non-linear filters [17,57,58]. In this work, we

[4]This is not a theoretical example; in fact, the site from which we obtained this data [32] plots the time series in a six-page PDF. This presentation mode captures fine-grained structure but makes it difficult to determine long-term trends at a glance, as in Figure 3.

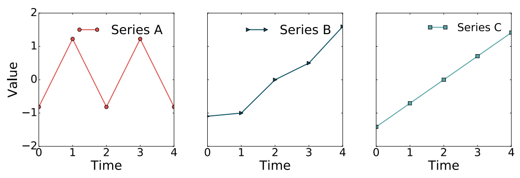


Figure 4: Three time series that appear visually distinct yet all have mean of zero and standard deviation of one. This example illustrates that standard summary statistics such as mean and standard deviation might fail to capture the visual "smoothness" of time series.

study a specific type of linear smoothing filter—moving average—and the problem of its automatic parameter selection. Despite its simplicity, moving average is an effective time domain filter that is optimal at reducing random noise while retaining a sharp step response (i.e., rapid rise in the data) [11].

While there are many studies on parameter selection mechanisms for various smoothing functions [46], the objective of most of the above selection criteria is to preserve the original signal (i.e., minimize variants of root mean squared error between the original and the processed signal). In contrast, ASAP's quality metric is designed to highlight trends and large deviations, leading to a different optimization strategy. In the biomedical field, researchers have explored ideas of selecting a moving average window size that highlights significantly deviating region of DNA sequences [55]. ASAP adopts a similar measure for quality—namely, the existence of non-random deviations in the time series—but is empirically much more efficient than the exhaustive approach described in the study.

Stream Processing. To enable efficient execution, ASAP is architected as a streaming operator and adapts techniques from stream processing systems. As such, ASAP is compatible with and draws inspiration from the wide array of existing systems literature on architectures for combining signal processing and stream processing functionality [28, 34].

Specifically, aggregation over sliding windows has been widely recognized as a core operator over data streams. Sliding window semantics and efficient incremental maintenance techniques have been well-studied in the literature [15, 56]. ASAP adopts the sliding window aggregation model. However, instead of leaving users to select a window manually, in the parlance of machine learning, ASAP performs *hyperparameter tuning* [41] to automatically select a window that quantitatively delivers high-quality smoothed plots. We are unaware of any existing system—in production or in the literature—that performs this hyperparameter selection for smoothing time series plots. Thus, the primary challenge we address in this paper is efficiently and effectively performing this tuning via visualization-specific optimizations that leverage target display resolution, the periodicity of the signal, and on-demand updates informed by the limits of human perception.

3. PROBLEM DEFINITION

In this section, we introduce two key metrics from the statistics literature that ASAP uses to assess the quality of smoothing. We subsequently cast ASAP as an optimization problem.

3.1 Roughness Measure

As we have discussed, noise and/or frequent fluctuations can distract users from identifying large-scale trends in time series visualizations. Therefore, to prioritize user attention, we wish to *smooth as much as possible while preserving systematic deviations* in time series visualizations. We develop metrics to quantitatively measure the degree of smoothing in a given visualization of a time series.

Unfortunately, standard summary statistics such as mean and standard deviation alone may not suffice to capture the salient visual aspects of time series "roughness." For example, consider the three time series in Figure 4: a jagged line (series A), a slightly bent line (series B), and a straight line (series C). These time series appear different, yet all have a mean of zero and standard deviation of one. The reason why series C looks "smoother" than series A and series B is that it has a constant slope. Put another way, the differences between consecutive points in series C have smaller variation than consecutive points in series A and B.

To formalize this intuition, we turn to the statistics literature. We define the roughness (i.e., inverse "smoothness," to be minimized) of a time series as the standard deviation of the differences between consecutive points in the series. The smaller the variation of the differences, the smoother the time series. Formally, given time series $X = \{x_1, x_2, ..., x_N\}$, $x_i \in \mathbb{R}$, we adopt the concept of the *first difference* series from statistics [20] as:

$$\Delta X = \{\Delta x_1, \Delta x_2, ...\} \quad s.t. \quad \Delta x_i = x_{i+1} - x_i, i \in \{1, 2, ..., N-1\}$$

Subsequently, we can define the roughness of time series X as:

$$\text{roughness}(X) = \sigma(\Delta X)$$

where $\sigma(X)$ is the standard deviation of X.

This use of variance of differences is closely related to the concept of a *variogram* [22], a commonly-used measure in spatial statistics (especially geostatistics) that characterizes the spatial continuity (or *surface roughness*) of a given dataset.

By this definition, the roughness of the three time series in Figure 4 are 2.04, 0.4, and 0, respectively. Note that a time series will have roughness value of 0 if and only if the corresponding plot is a straight line (like series C). Specifically, a roughness value of 0 implies the differences between neighboring points are identical and therefore the plot corresponding to the series will have a constant slope, resulting in a straight line.

3.2 Preservation Measure

Per the above observation, if we simply minimize roughness, we will produce plots that approximate straight lines. In some cases, this is desirable; if the overall trend is a straight line, then removing noise may result in a straight line. However, as our examples in Section 2 demonstrate, many meaningful trends are not accurately represented by straight lines. As a result, we need a measure of "trend preservation" that captures how well we are preserving large-scale structure within the time series.

To quantify how well we are preserving large deviations in the original time series, we adopt the metric of distribution *kurtosis* from statistics [24]. Kurtosis measures the "tailedness" of the probability distribution of a real-valued random variable, or how much mass is near the tails of the distribution. More formally, given a random variable X with mean μ and standard deviation σ, kurtosis is defined as the fourth standardized moment:

$$\text{Kurt}[X] = \frac{\mathbb{E}[(X-\mu)^4]}{\mathbb{E}[(X-\mu)^2]^2}$$

Higher kurtosis means that more of the variance is contributed by rare and extreme deviations, instead of more frequent and modestly sized deviations [60]. For reference, the kurtosis for univariate normal distribution is 3. Distributions with kurtosis less than 3, such as the uniform distribution, produce fewer and less extreme outliers compared to normal distributions. Distributions with kurtosis larger than 3, such as the Laplace distribution, have heavier tails compared to normal distributions. Figure 5 illustrates two time series sampled from the normal and Laplace distribution discussed above. Despite having the same mean and variance, the kurtosis measure captures

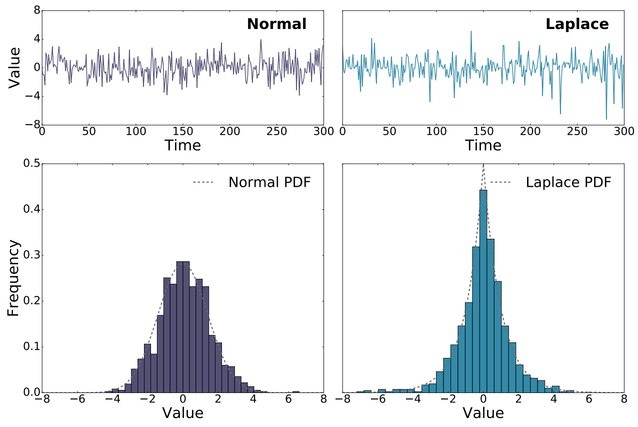
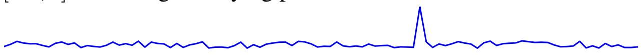


Figure 5: The plot shows two time series and histograms of the underlying distribution (Normal distribution on the left, Laplace distribution on the right) that the corresponding time series was sampled from. Despite having the same mean (0) and variance (2), one series includes a few large deviations, while the other includes a large number of moderate deviations. The difference in tendency to produce outliers is captured by kurtosis: normal distribution has kurtosis of 3, while the Laplace distribution has kurtosis of 6.

the two series' difference in tendency to produce outliers.

To prevent oversmoothing large-scale deviations in the original time series, we compare kurtosis of the time series before and after applying the smoothing function. Informally, if the kurtosis of the smoothed series no less than that of the original series, then the proportion of values that significantly deviate in the smoothed series is no less than the proportion in the original series. In other words, preserving kurtosis is equivalent to preserving the tails of the distribution, or preserving parts of the distribution that are far from mean.[5] If smoothing is effective, then the smoothing will "concentrate" the values around regions of large deviation (i.e., significant shifts from the mean) and therefore highlight these deviations.

If the original time series only contains a few extreme outliers, the smoothing is likely to only average out the deviations, which we also account for in our parameter selection procedure. For example, consider the following time series with all but one point in the range $[-1,1]$ and a single outlying point that has a value of 10:



This outlier may be the most important piece of information that users would like to highlight in the time series, so applying a simple moving average only decreases the extent of this deviation (i.e., the kurtosis of the smoothed time series decreases). In this case, the kurtosis preservation constraint ensures we leave the original time series unsmoothed.

3.3 Smoothing Function

Given our roughness measure and preservation measure, we wish to smooth our time series as much as possible (i.e., minimizing roughness) while preserving deviation (i.e., preserving kurtosis). To perform the actual smoothing, we need a smoothing function to transform the time series.

In this paper, we focus on simple moving average (SMA) as a smoothing function. Three primary reasons motivate this choice. First, SMA is well studied in the stream processing literature, with several existing techniques for efficient execution and incremental maintenance [40]. We adopt these techniques, while using roughness and preservation metrics as a means of automatically tuning its parameters for visual effect. Second, SMA is also well studied in the signal processing community. Statistically, the moving average is optimal for recovering the underlying trend of the time series when the fluctuations about the trend are normally distributed [11]. This is perhaps surprising given SMA's relatively light computational footprint and conceptual simplicity compared to alternatives. Third, we experimented with several alternatives including the exponentially weighted moving average, short-time Fourier transform [54], and the Savitzky-Golay filter [51]; SMA had fewer parameters to tune and proved more effective at smoothing per our target metrics.

Given input $w \in \mathbb{N}$, SMA averages every sequential set of w points in the original time series X to produce one point in the smoothed series Y. We can express SMA as:

$$\text{SMA}(X,w) = \{y_1, \ldots y_{N-w}\} \quad s.t. \quad y_i = \frac{1}{w}\sum_{j=0}^{w-1} x_{i+j}$$

In the sliding window model, the window specification also includes a *slide size* parameter. That is, while the window size determines the number of elements in each window, the slide size determines the distance between two neighboring windows. In time series visualization, slide size determines the sampling frequency of the original time series and, therefore, the number of distinct, discrete data points in the plot. As a result, increasing slide size removes detail from users, which, when coupled with smoothing, conflicts with our goal of providing users with more (useful) context and information. In addition, a sampled series may be a biased representation of the original signal.[6] Thus, in this work, we focus on tuning window size for a fixed slide size.

3.4 Smoothing Problem Statement

Given our roughness and preservation measures and smoothing function, we present the smoothing problem statement as follows:

PROBLEM. *Given time series $X = \{x_1, x_2, \ldots, x_N\}$, let $Y = \{y_1, y_2, \ldots, y_{N-W}\}$ be the smoothed series of X obtained by applying a simple moving average with window size w (i.e. $y_i = \frac{1}{w}\sum_{j=0}^{w-1} x_{i+j}$). Find window size $\hat{w}$ where:*

$$\hat{w} = \arg\min_{w} \sigma(\Delta(Y)) \quad s.t. \quad \text{Kurt}[Y] \geq \text{Kurt}[X]$$

That is, we wish to reduce roughness in a given time series as much as possible by applying a sliding window average function to the data while preserving kurtosis.

4. ASAP

In this section, we describe ASAP's core search strategy and optimizations for solving the problem of window length selection. We first focus on smoothing a single, fixed-length time series, starting with a walkthrough of a strawman solution (Section 4.1). We then analyze the problem dynamics under a simple, IID distribution (Section 4.2) and, using the insights from this analysis, we develop a pruning optimization based on autocorrelation (Section 4.3). We further introduce a pixel-aware optimization that greatly reduces the search space via preaggregation (Section 4.4). Finally, we discuss the streaming setting (Section 4.5).

[5]Thus, in this paper, we use kurtosis preservation as a constraint on window size. However, our results generalize to any constraint that is monotonic in window size under sliding-window moving average.

[6]As we have discussed, related work has considered the problem of non-uniform sampling for minimizing reconstruction display error in time series plots [33]. Here, we are interested in minimizing roughness instead.

4.1 Strawman Solution

As a strawman solution, we could exhaustively search all possible window lengths and return the window length that gives the smallest roughness measure while satisfying the kurtosis constraint. For each candidate window length, we will need to smooth the series and evaluate the variance and kurtosis. Each of these operations requires linear time ($O(N)$) to evaluate. However, there are also many candidates to evaluate. For a time series of size N, we may need to evaluate up to N possible window lengths for a total running time of $O(N^2)$. In the regime where N is even modestly large, this computation can be prohibitively expensive.

We could improve the runtime of this exhaustive search by performing grid search via a sequence of larger step sizes or by performing a binary search. However, as we will demonstrate momentarily, roughness is not guaranteed to be monotonic in window length and therefore the above search strategies may deliver poor quality results. Thus, in the remainder of this section, we describe an alternative search strategy that is able to retain the quality of exhaustive search while achieving meaningful speedups by optimizing for the desired pixel density and quickly pruning unpromising candidates.

4.2 Basic IID Analysis

To leverage properties of the roughness metric to speed ASAP's search, we first consider how window length affects the roughness and kurtosis of the smoothed series.

To begin to understand this relationship, assume that the original time series $X : \{x_1, x_2, ..., x_N\}$ consists of samples drawn identically independently distributed (IID) from some distribution with mean μ and standard deviation σ. After applying a moving average of window length w, we obtain the smoothed series:

$$Y = \text{SMA}(X, w), \quad y_i = \frac{1}{w}\sum_{j=0}^{w-1} x_{i+j}, i \in \{1, 2, ..., N-w\}$$

We can write the forward difference series as $\Delta Y = \{\Delta y_1, \Delta y_2, ...\}$,

$$\Delta y_i = y_{i+1} - y_i = \frac{1}{w}\sum_{j=0}^{w-1}(x_{i+j+1} - x_{i+j}) = \frac{1}{w}(x_{i+w} - x_i)$$

For convenience, we denote the first $N-w$ points of X as $X_f = \{x_1, x_2, ..., x_{N-w}\}$ and the last $N-w$ points of X as $X_l = \{x_{w+1}, x_{w+2}, ..., x_N\}$. Then $\Delta Y = \frac{1}{w}(X_l - X_f)$, and roughness of the smoothed series Y can be written as:

$$\begin{aligned} \text{roughness}(Y) &= \sigma(\Delta Y) = \frac{1}{w}\sigma(X_l - X_f) \\ &= \frac{1}{w}\sqrt{\text{var}(X_f) + \text{var}(X_l) - 2\text{cov}(X_l, X_f)} \end{aligned} \tag{1}$$

Since each x_i is drawn IID from the same distribution, we have that:

$$\text{var}(X_f) = \text{var}(X_l) = \sigma^2, \quad \text{cov}(X_f, X_l) = 0$$

Substituting in Equation 1 we obtain:

$$\text{roughness}(Y) = \frac{\sqrt{2}\sigma}{w} \tag{2}$$

This analysis demonstrates that for IID data, roughness linearly decreases with increased window length. Furthermore, the kurtosis of random variable S, defined as the sum of independent random variables $R_1, R_2, ..., R_n$, is [47]

$$\text{Kurt}[S] - 3 = \frac{1}{(\sum_{j=1}^{n}\sigma_j^2)^2}\sum_{i=1}^{n}\sigma_i^4(\text{Kurt}[R_i] - 3)$$

where σ_i is the standard deviation of random variable R_i. In our case, Y is the sum of w IID random variables X. Therefore, the above equation simplifies to

$$\text{Kurt}[Y] - 3 = \frac{\text{Kurt}[X] - 3}{w} \tag{3}$$

Thus, for distributions with initial kurtosis less than 3, kurtosis monotonically increases with window length and for those with initial kurtosis larger than 3, kurtosis monotonically decreases.

In summary, these results indicate that for IID data, we can simply search for the largest window length that satisfies kurtosis constraint via binary search. Specifically, given a range of candidate window lengths, ASAP applies SMA with window length that is in the middle of the range. If the resulting smoothed series violates the kurtosis constraint, ASAP searches the smaller half of the range; ASAP searches the large half otherwise. This binary search routine is justified because the roughness of the smoothed series monotonically decreases with window length (Equation 2), and the kurtosis of the smoothed series monotonically decreases with window length or achieves its minimum at window length equals one (Equation 3).

However, many time series do not exhibit IID behavior and instead exhibit temporal correlations, which breaks the IID assumption. This complicates the problem of window search; we present a solution in the next subsection.

4.3 Optimization: Autocorrelation Pruning

We have just shown that, for IID data, binary search is accurate, yet many time series are not IID; instead, they are often periodic or exhibit other temporal correlations. For example, many servers and automated processes have regular workloads and exhibit periodic behavior across hourly, daily, or longer intervals.

To measure temporal correlations within a time series, we adopt the concept of *autocorrelation*, or the similarity of a signal with itself as a function of the time lag between two points [52]. Formally, given a process X whose mean μ and variance σ^2 are time independent (i.e., is a *weakly stationary* process), denote X_t as the value produced by a given run of the process at time t. The lag τ autocorrelation function (ACF) on X is defined as

$$\text{ACF}(X, \tau) = \frac{\text{cov}(X_t, X_{t+\tau})}{\sigma^2} = \frac{\mathbb{E}[(X_t - \mu)(X_{t+\tau} - \mu)]}{\sigma^2}$$

The value of the autocorrelation function ranges from [-1, 1], with 1 indicating perfect correlation, 0 indicating the lack of correlation and -1 indicating anti-correlation.

4.3.1 Autocorrelation and Roughness

As hinted above, we can take advantage of the periodicity in the original time series to prune the search space. Specifically, given the original time series $X : \{x_1, x_2, ..., x_N\}$, and the smoothed series $Y : \{y_1, y_2, ..., y_{N-w}\}$ obtained by applying a moving average of window length w, we show that:

$$\text{roughness}(Y) = \frac{\sqrt{2}\sigma}{w}\sqrt{1 - \frac{N}{N-w}\text{ACF}(X, w)} \tag{4}$$

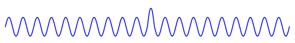
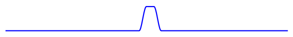
given X is a weakly stationary process. We provide a full derivation of Equation 4 in Appendix B.1. Intuitively, this equation implies that both window length and autocorrelation have an important role to play in minimizing roughness. Consider a time series recording the number of taxi trips taken over 30-minute intervals. Due to the regularity of commuting routine, this time series exhibits autocorrelation across week-long periods (e.g., a typical Monday is likely to be much more similar to another Monday than a typical Saturday). Furthermore, a rolling weekly average of the number of trips should, in expectation, have a smaller variance than rolling 6-day averages: for example, if people are more likely to take taxis during weekdays than during weekends, then the average from Monday to Saturday should be larger than the average from Tuesday to Sunday. Therefore, window lengths that align with periods of high autocorrelation also make the resulting series smoother.

We experimentally validate this analytic relationship on real world data (Appendix B.1) and use this relationship to aggressively prune the space of windows to search (Section 4.3.3).

4.3.2 *Autocorrelation and Kurtosis*

In addition to investigating the impact of temporal correlation on roughness, we also wish to understand how it affects our kurtosis constraint. We start with an example that illustrates the case when choosing window lengths that align with periods of the time series leads to higher kurtosis.

Suppose our time series (sparkline on the left) is a sine wave with 640 data points. Each complete sine wave is 32 data points long, and in the region from 320th to 336th data point, the peak of the sine wave is taller than usual. In this case, applying a moving average with window lengths that are not multiples of the period will result in a similarly shaped time series with smaller amplitude. In contrast, when applying a window that are multiples of the period, the smoothed series (sparkline on the right) is zero everywhere except around the region where the peak is higher. The smoothed series in the latter case has higher kurtosis because it only contains one large deviation from the mean.

This example illustrates that applying moving average with window lengths aligning with the period of the time series can "remove" periodic fluctuations from the visualization, and therefore highlight deviations from period to period. The kurtosis of the smoothed series is also larger at these window lengths. In ASAP, this means that if applying a candidate window that aligns with the period of the time series does not result in a smoothed series satisfying the kurtosis constraint, it is less likely that a candidate window off the periodicity boundary would do.

4.3.3 *Pruning Strategies*

Following the above observations, ASAP adopts the following two pruning strategies. The corresponding pseudocode for the search is listed in Algorithm 1.

Autocorrelation peaks. To quickly filter out suboptimal window lengths, we search for windows that correspond to periods of high autocorrelation. Specifically, we only check autocorrelation *peaks*, which are local maximums in the autocorrelation function and correspond to periods in the time series. For periodic datasets, these peaks are usually much higher than neighboring points, meaning that the corresponding roughness of the smoothed time series is much lower. This is justified by Equation 4 — all else equal, roughness decreases with the increase of autocorrelation.

Naïvely computing autocorrelation via brute force requires $O(n^2)$ time; this approach is unlikely to deliver speedups over the naïve exhaustive search for finding window length. However, we can improve the runtime of autocorrelation, to $O(n\log(n))$ time, using two Fast Fourier Transforms (FFT) [50]. In addition to providing asymptotic speedups, this approach also allows us to make use of optimized FFT routines designed for signal processing, in the form of mature software libraries and increasingly common hardware implementations (e.g., DSP accelerators).

Large to small. Since roughness decreases with window length (Equation 4, roughness is proportional to $\frac{1}{w}$), ASAP searches from larger to smaller window lengths. When two windows $w_1, w_2 (w_1 < w_2)$ have identical autocorrelation, the larger window will always have lower roughness under SMA. However, when they have different autocorrelations a_1, a_2, the smaller window w_1 will only provide lower roughness if $w_1 > w_2\sqrt{\frac{1-a_1}{1-a_2}}$. Moreover, since we are only considering autocorrelation peaks as candidate windows, a_1 is no larger than the largest autocorrelation peak in the time series, which we refer to as *maxACF*. Therefore, the smallest window w_1 that is able to produce smoother series than w_2 must satisfy

$$w_1 > w_2\sqrt{\frac{1-a_1}{1-a_2}} > w_2\sqrt{\frac{1-maxACF}{1-a_2}} \qquad (5)$$

If we find a feasible smoothing window relatively early in the search, we can use Equation 5 to prune out smaller windows that will not produce a smoother series (UPDATELB in Algorithm 1). Similarly, once we have a feasible window, we can also prune search candidates whose roughness estimate (via Equation 4) is larger than the current best (ISROUGHER in Algorithm 1). The lower bound pruning cuts the search space from below, and the roughness estimate cuts the search space from above, eliminating search candidates larger than the lower bound.

Algorithm 1 Search for periodic data

```
Variables:
$X$: time series; $candidates$: array of candidate window lengths
$opt$: set of states < roughness, LB, window, largestFeasible >
$acf[w]$: autocorrelation for $w$
$maxACF$: maximum autocorrelation peak

function UPDATELB(LB, w)                                   ▷ Update lower bound
    return MAX(LB, $w\sqrt{\frac{1-maxACF}{1-acf[w]}}$)
function ISROUGHER(opt, w)                                 ▷ Compare roughness
    return $\frac{\sqrt{1-acf[w]}}{w} > \frac{\sqrt{1-acf[opt.window]}}{opt.window}$
function SEARCHPERIODIC(X, candidates, opt)
    N = candidates.length
    for i ∈ {N, N-1, ..., 1} do                            ▷ Large to small
        w = candidates[i]
        if w < opt.LB then                                 ▷ Lower bound pruning
            break
        if ISROUGHER(opt, w) then                          ▷ Roughness pruning
            continue
        Y = SMA(X, w)
        if ROUGHNESS(Y) < opt.roughness and
            KURT(Y) ≥ KURT(X) then                         ▷ Kurtosis constraint
            opt.window = w
            opt.roughness = ROUGHNESS(Y)
            opt.LB = UPDATELB(opt.LB, w)
            opt.largestFeasible = MAX(opt.largestFeasible, i)
    return opt
```

Algorithm 2 Batch ASAP

```
function FINDWINDOW(X, opt)
    candidates = GETACFPEAKS(X)
    opt = SEARCHPERIODIC(X, candidates, opt)
    head = MAX(opt.LB, candidates[opt.largestFeasible] + 1)
    tail = MIN(maxWindow, candidates[opt.largestFeasible + 1])
    opt = BINARYSEARCH(X, head, tail, opt)
    return opt.window
```

Device	Resolution	Reduction on 1M pts
38mm Apple Watch	272 x 340	3676x
Samsung Galaxy S7	1440 x 2560	694x
13” MacBook Pro	2304 x 1440	434x
Dell 34 Curved Monitor	3440 x 1440	291x
27” iMac Retina	5120 x 2880	195x

Table 1: Popular devices and data reduction achieved using pixel-aware preaggregation for 1M points.

4.4 Optimization: Pixel-Aware Preaggregation

In addition to leveraging statistical properties of the data, we can also leverage perceptual properties of the target devices. That is, ASAP's smoothed time series are designed to be displayed on devices such as computer monitors, smartphones, and tablet screens for human consumption. Each of these target media has a limited resolution; as Table 1 illustrates, even high-end displays such as the 2016 Apple iMac 5K are limited in horizontal resolution to 5120 pixels, while displays such as the 2016 Apple Watch contain as few as 272 pixels. These pixel densities place restrictions on the amount of information that can be displayed in a plot.

ASAP is able to leverage these limited pixel densities to improve search time. Specifically, ASAP avoids searching for window lengths that would result in more points than pixels supported by the target device. For example, a datacenter server may report CPU utilization metrics every second (604,800 points per week). If an operator wants to view a plot of weekly CPU usage on her 2016 Retina MacBook Pro, she will only be able to see a maximum of 2304 distinct pixels as supported by the display resolution. If ASAP smooths using a window smaller than 262 seconds (i.e., $\frac{604,800}{2304}$), the resulting plot will contain more points than pixels on the operator's screen (i.e., to display all information in the original time series, the slide size must be no larger than window length). As a result, this *point-to-pixel* ratio places a lower bound on the window length that ASAP should search. In addition, the point-to-pixel ratio is also a useful proxy for the granularity of information content contained in a given pixel. While one could search for window lengths that correspond to sub-pixel boundaries, in practice, we have found that searching for windows that are integer multiples of the point-to-pixel ratio suffices to capture the majority of useful information in a plot. We provide an analysis in Appendix B.2, and empirically demonstrate these phenomena in Section 5.2.2.

Combined, these observations yield a powerful optimization for ASAP's search strategy. Given a target display resolution (or desired number of points for a plot), ASAP pushes this information into its search strategy by only searching windows that are integer multiples of point-to-pixel ratio. To implement this efficiently, ASAP preaggregates the data points according to groups of size corresponding to the point-to-pixel ratio, then proceeds to search over these preaggregated points. With this preaggregation, ASAP's performance is not dependent on the number of data points in the original time series but instead depends on the target resolution of the end device. As a result, in Section 5, we evaluate ASAP's performance over different target resolutions and demonstrate scalability to millions of incoming data points per second.

4.5 Streaming ASAP

ASAP is designed to process streams of time series and update plots as new data arrives. In this section, we describe how ASAP efficiently operates over data streams by combining techniques from traditional stream processing with constraints on human perception.

Basic Operations. As new data points arrive, ASAP must update its smoothing parameters to accommodate changes in the trends, such as periodicity. As in Section 4.4, in the streaming setting, we can preaggregate data as it arrives according to the point-to-pixel ratio. However, as data transits the duration of time ASAP is configured to smooth (e.g., the last 30 minutes of readings), ASAP must remove outdated points from the window. To manage this intermediate state, ASAP adapts standard techniques from streaming processing that sub-aggregate input streams for performance gain. That is, sliding window aggregates such as SMA can be computed more efficiently by sub-aggregating the incoming data into disjoint segments (i.e., *panes*) that are sizes of greatest common divisor of window and slide size [40]. We can perform similar pixel-aware preaggregations for data streams using panes.

ASAP maintains a linked list of all subaggregations in the window and, when prompted, re-executes the search routine from the previous section. Instead of recomputing the smoothing window from scratch, ASAP records the result of the previous rendering request and uses it as a “seed” for the new search. Specifically, since streams often exhibit similar behavior over time, the previous smoothing parameter could possibly apply to the current request. In this case, we start the new search with a known feasible window length, which enables the roughness estimation pruning procedure (ISROUGHER in Algorithm 1) to potentially rule out candidates automatically.

Optimization: On-demand updates. A naïve strategy for updating ASAP's output is to update the plot upon arrival of each point. This is inefficient. For example, consider a data stream with volume of one million points per second. Refreshing the plot for every data point requires updating the plot every 0.001 milliseconds. However, since humans can only perceive changes on the order of 60 events per second [31], this update rate is unnecessary. With pixel-aware preaggregation, we would refresh for each aggregated data point instead, the rate of which may still be higher than necessary. To visualize 10 minutes of data on a 27-inch iMac for example, pixel-aware preaggregation provides us aggregates data points that are 12ms apart (83Hz). As a result, we designed ASAP to only refresh at (configurable) timescales that are perceptible to humans. In our example above, a 1Hz update speed results in a 83$\times$ reduction in number of calls to the ASAP search routine; this reduction means we will either use less processing power and/or be able to process data at higher volumes. In Section 5.2.2, we empirically investigate the relationship between total runtime and refresh rate.

Putting it all together. Algorithm 3 shows the full streaming ASAP algorithm. We aggregate the incoming data points according to the point-to-pixel ratio, and maintain a linked list of the aggregates. After collecting a refresh interval time worth of aggregates, ASAP updates data points in the current visualization, and recalculates the autocorrelation (UPDATEACF). ASAP then checks whether the window length from the last rendering request is still feasible (CHECKLASTWINDOW). If so, ASAP uses this previous window length to quickly improve the lower bound for the new search.

5. EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate the quality and efficiency of ASAP's results via two user studies and a series of performance benchmarks. Our goal is to demonstrate that:

- ASAP's visualizations improve both accuracy and response time (Section 5.1).
- ASAP identifies high quality window sizes quickly (Section 5.2.1).
- ASAP's optimizations—autocorrelation, pixel-aware aggregation and on-demand update—provide complementary speedups up to seven order-of-magnitude over baseline (Section 5.2.2).

Algorithm 3 Stream ASAP

```
Variables:
X: preaggregated time series; interval: refresh interval

function CHECKLASTWINDOW(X, opt)
    Y = SMA(X, opt.window)
    if KURT(Y) $\geq$ KURT(X) then
        update roughness and LB for opt
    else
        re-initialize opt
    return opt
function UPDATEWINDOW(X, interval)
    while True do
        collect new data points until interval
        subaggregate new data points, and update X
        UPDATEACF(X)
        opt = CHECKLASTWINDOW(X, opt)
        FINDWINDOW(X, opt)
```

5.1 User Studies

We first empirically evaluate the effectiveness of ASAP's visualizations via two user studies. We demonstrate that ASAP visualizations lead to faster and more accurate identifications of anomalies.

Visualization Techniques for Comparison. In each study, we compare ASAP's visualizations to a series of alternative visualizations: *i*) the original data (i.e., similar to M4 [33]), *ii*) visualizations generated by piecewise aggregate approximation (PAA) [35] (PAA100 reduces the number of points in the time series to 100; PAA800 reduces the number to 800), and *iii*) an "oversmoothed" plot generated by applying an SMA on the original time series with a window size of a quarter of the number of points.

Datasets. We select five publicly-available datasets—Taxi, Power, Sine, EEG, Temp (described in Table 2)—because each has known ground truth anomalies. We use this ground truth as a means of assessing visualization quality; specifically, we use this ground truth to determine whether users are able to identify anomalous behaviors in the visualization and to assess their preferences.

We present all time series plots and the accompanying text descriptions for the user study as well as a case study of ASAP smoothing real world datasets in the Appendix.

5.1.1 User Study: Anomaly Identification

To assess how different smoothing algorithms affect users' ability to identify anomalies in time series visualization, we ran a large-scale user study on Amazon Mechanical Turk, in accordance with Stanford University IRB guidelines.

In this study, we presented users with textual descriptions of each dataset and anomaly and asked them to select which of five equally-sized regions in a given time series visualization the described anomaly occurred. We presented a single, randomly chosen visualization for each dataset, and users performed this anomaly identification on five separate datasets. For each question, we recorded user's accuracy as well as page submission time.

Figure 6 depicts the accuracy and response time from 250 users. For reference, 143 self-reported as intermediate or expert users of Excel, 107 self-reported as intermediate or expert users of databases, and 101 self-reported seeing time series at least once per month. When shown ASAP's visualizations, users were more likely to correctly identify the anomalous region and to do so more quickly. Specifically, users were 23.9% more likely to select the correct anomalous region when presented with ASAP's visualizations in-

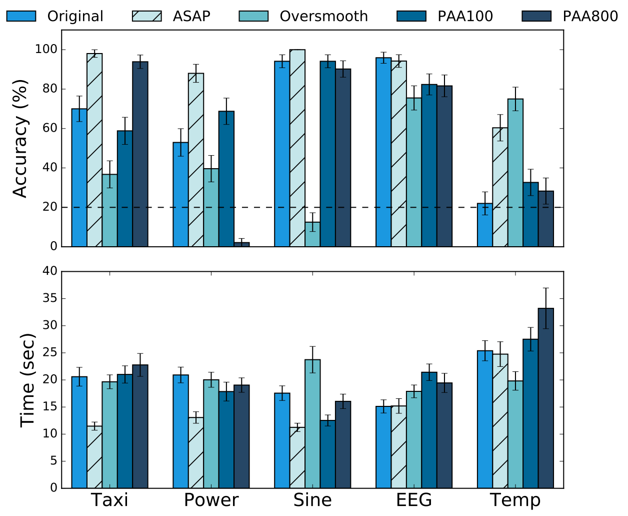


Figure 6: Accuracy of identifying anomalous region and page submission time for each dataset and visualization, with error bars indicating standard error of samples. On average, ASAP improves accuracy by 48.3% while reducing response time by 26.3% faster compare to other visualizations.

stead of the original time series, and they did so 31.5% more quickly. Compared to all other methods, on average, users were 54.6% (max: 84.1%) more likely to select the correct region with ASAP, with a 27.0% (max: 31.5%) decrease in response time. ASAP led to most accurate results for all datasets except for dataset Temp, in which the oversmooth strategy was able to better highlight (by 14.6%) a large increasing temperature trend over several decades, corresponding to the rise of global warming. However, for this dataset, ASAP results in 38.4% more accurate identification than the raw data. Overall, ASAP consistently produces high-quality plots, while the quality of alternative visualization methods varies widely across datasets.

5.1.2 User Study: Visual Preferences

In addition to the above user study, which was based on a large crowdsourced sample, we performed a targeted user study with 20 graduate students in Computer Science (also in accordance with Stanford IRB guidelines). We retained the same datasets from the previous study, presented users with textual descriptions of each dataset and anomaly, and asked them to select the *visualization* that best illustrates the anomaly. That is, due to smaller sample size, in contrast with the previous study, we presented a set of four visualizations (anonymized and randomly permuted) and asked users to select the visualization that best highlights the described anomaly. We presented with four anonymized options: original, ASAP, PAA100, and oversmooth.

Figure 7 presents results from this study. Across all five datasets, users preferred ASAP's visualizations as a means of visualizing anomalies in 66% of the trials (random: 25%). Specifically, for datasets Taxi (Figure D.2, Appendix), EEG (Figure D.3, Appendix), and Power (Figure D.5, Appendix), over 75% of users preferred ASAP's presentation of the time series. For these datasets, smoothing helps remove the high-frequency fluctuations in the original dataset and therefore highlights the known anomalies. For dataset Sine (Figure D.4, Appendix), a simulated noisy sine wave with a small region where the period is halved, 60% users chose ASAP, followed by 30% choosing PAA100. In follow-up interviews, some users expressed uncertainty about this particular plot: while the ASAP plot clearly highlights the anomaly, the PAA100 plot more

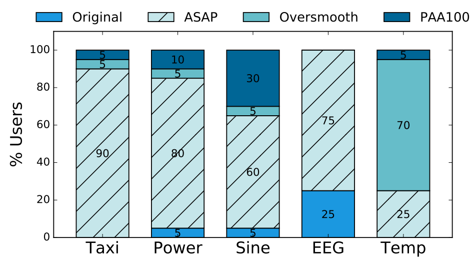


Figure 7: User's preferred visualization for each dataset.

closely resembles the description of the original signal. As in the anomaly identification study, in the Temp dataset, 70% of users chose the oversmoothed plot, and 25% chose ASAP. No user preferred the original temperature plot, further confirming that proper smoothing is necessary.

In summary, these results illustrate the utility of ASAP's target metrics in producing high-quality time series visualizations that highlight anomalous behavior.

5.2 Performance Analysis

While the above user studies illustrate the utility of ASAP's visualizations, it is critical that ASAP is able to render them quickly and over changing streams. To assess ASAP's end-to-end performance as well as the impact of each of its optimizations, we performed a series of performance benchmarks.

Implementation and Experimental Setup. We implemented an ASAP prototype as an explanation operator for processing output data streams in the MacroBase streaming analytics engine [18]. We report results from evaluating the prototype on a server with four Intel Xeon E5-4657L 2.4GHz CPUs containing 12 cores per CPU and 1TB of RAM (although we use considerably less RAM in processing). We exclude data loading time from our results but report all other computation time. We report results from the average of three or more trials per experiment. We use a set of 11 of datasets of varying sizes collected from a variety of application domains; Table 2 provides detailed descriptions of each dataset, and we include the original and smoothed plots from the experiments in Appendix E.

5.2.1 End-to-End Performance

To demonstrate ASAP's ability to find high-quality window sizes quickly, we evaluate ASAP's window quality and search time compared to alternative search strategies. We compare to exhaustive search, grid search of varying step size (2, 10), and binary search.

First, as Table 2 illustrates, with a target resolution of 1200 pixels, ASAP is able to find the same smoothing parameter as the exhaustive search for all datasets. For one dataset (Twitter_AAPL), both exhaustive search and ASAP leave the visualization unsmoothed; this time series (Figure E.1, Appendix) is smooth except for a few unusual peaks, so further smoothing would have averaged out the peaks. Overall, exhaustive search checks an average of 113.64 candidates per dataset, while ASAP produces the same results by checking only 8.64 candidates.

Second, we evaluate differences in wall-clock speed and achieved smoothness. All algorithms run on preaggregated data, so the throughput difference is only caused by the difference in search strategies; we further investigate the impact of pixel-aware preaggregation in Section 5.2.2. Figure 8 shows that ASAP is able to achieve up to 60× faster search time than exhaustive search, with near-identical roughness ratio. ASAP's runtime performance scales comparably to binary search, although it lags by up to 50% due

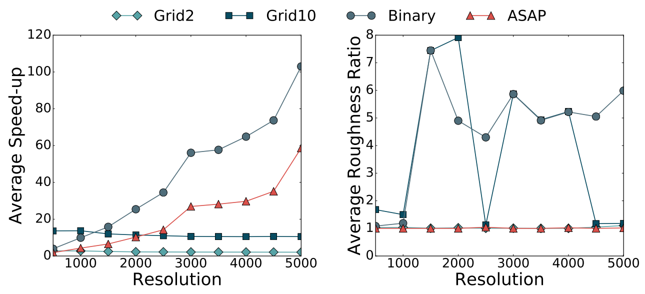


Figure 8: Throughput and quality of ASAP, grid search with different step sizes, and binary search running on time series preaggregated according to varying target resolutions. Here we report an average from 7 datasets with more than 5000 points in Table 2, and use throughput and roughness of exhaustive search as the baseline for both plots. ASAP exhibits similar speed up trends with binary search, while retaining quality close to the exhaustive search. While the autocorrelation calculation in ASAP causes up to 50% lag in performance comparing to binary search, binary search is up to 7.5× rougher than ASAP.

to its autocorrelation calculation; however, while ASAP produces high-quality smoothed visualizations, binary search is up to 7.5× rougher than ASAP. Grid search with step size two delivers similar-quality results as ASAP but fails to scale, while grid search with step size ten delivers the worst overall results. In summary, end-to-end, ASAP provides significant speedups over exhaustive search while retaining its quality of visualization.

5.2.2 Impact of Optimizations

In this section, we further evaluate the contribution of each of ASAP's optimizations—autocorrelation pruning, pixel-aware preaggregation, on-demand update—individually and combined.

Pixel-aware preaggregation. We first perform a microbenchmark on the impact of pixel-aware preaggregation (Section 4.4) on both throughput and quality. Figure 9 shows the throughput and quality of ASAP and exhaustive search with and without pixel-aware preaggregation under varying target resolutions. With pixel-aware preaggregation, ASAP achieves roughness within 20% of exhaustive search over the raw series and sometimes outperforms exhaustive search because the initial pixel-aware preaggregation results in lower initial kurtosis. The preaggregation strategy enables a five and a 2.5 order-of-magnitude speedups over exhaustive search (Exhaustive) and ASAP on raw data (ASAPno-agg), respectively. In summary, pixel-aware preaggregation has a modest impact on result quality and massive impact on computational efficiency (i.e., sub-second versus hours to process 1M points). Should users desire exact result quality, they can still choose to disable pixel-aware preaggregation while retaining speedups from other optimizations. We provide an analysis and additional throughput results in Appendix B.2.

On-demand update. To investigate the impact of the update interval in the streaming setting (Section 4.5), we vary ASAP's refresh rate and report throughput under each setting. The log-log plot (Figure 10) shows a linear relationship between the refresh interval and throughput. This is expected because updating the plot twice as often means that it would take twice as long to process the same number of points. For fast-moving streams, this strategy can save substantial computational resources.

Factor Analysis. In addition to analyzing the impact of individual optimizations, we also investigate how ASAP's three main optimizations combine. Figure 11 depicts a factor analysis, where we enable

Dataset	Description	# points	Duration	Exhaustive Search	ASAP
gas_sensor [43]	Recording of a chemical sensor exposed to a gas mixture	4,208,261	12 hours	window size: 26 # candidates: 115	window size: 26 # candidates: 7
EEG [36]	Excerpt of electrocardiogram	45,000	180 sec	window size: 22 # candidates: 119	window size: 22 # candidates: 21
Power [36]	Power consumption for a Dutch research facility in 1997	35,040	35040 sec	window size: 16 # candidates: 115	window size: 16 # candidates: 23
traffic_data [14]	Vehicle traffic observed between two points for 4 months	32,075	4 months	window size: 84 # candidates: 120	window size: 84 # candidates: 6
machine_temp [39]	Temperature of an internal component of an industrial machine	22,695	70 days	window size: 44 # candidates: 125	window size: 44 # candidates: 7
Twitter_AAPL [39]	A collection of Twitter mentions of Apple	15,902	2 months	window size: 1 # candidates: 120	window size: 1 # candidates: 7
ramp_traffic [43]	Car count on a freeway ramp in Los Angeles	8,640	1 month	window size: 96 # candidates: 117	window size: 96 # candidates: 5
sim_daily [39]	Simulated two week data with one abnormal day	4,033	2 weeks	window size: 72 # candidates: 100	window size: 72 # candidates: 5
Taxi [39]	Number of NYC taxi passengers in 30 min bucket	3,600	75 days	window size: 112 # candidates: 120	window size: 112 # candidates: 4
Temp [32]	Monthly temperature in England from 1723 to 1970	2,976	248 years	window size: 112 # candidates: 120	window size: 112 # candidates: 4
Sine [37]	Noisy sine wave with an anomaly that is half the usual period	800	800 sec	window size: 64 # candidates: 79	window size: 64 # candidates: 6

Table 2: Dataset descriptions and batch results from ASAP and exhaustive search. All results are generated for target resolution 1200 pixels. For all datasets, ASAP is able find the same smoothing parameter as exhaustive search while searching over 13× fewer candidates on average.

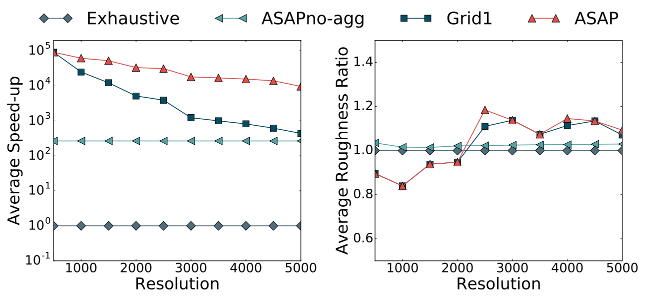


Figure 9: Throughput and quality of ASAP and exhaustive search on preaggregated time series over the baseline (exhaustive search over the original time series), under varying target resolutions, taken over three runs on six datasets. ASAP on aggregated time series is up to 4 orders of magnitude faster, while retaining roughness within 1.2 times of the baseline.

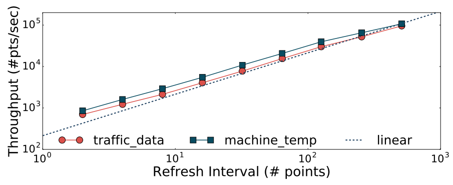


Figure 10: Throughput of streaming ASAP on two datasets, with varying refresh interval (measured in number of points) for target resolution 2000 pixels in log-log scale. The plot shows a linear relation between throughput and refresh interval, as expected.

each optimization in turn cumulatively. Pixel-aware aggregation provides between two and four orders of magnitude improvement depending on the target resolution. Autocorrelation gives an additional two orders of magnitude. Finally, on demand update with a daily refresh interval (updating for every 288 points in the original series versus updating for each preaggregated point) provides an-

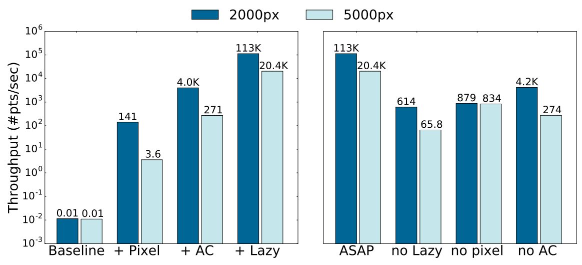


Figure 11: Factor analysis on machine_temp dataset under two display settings. Adding optimizations one at a time shows that each optimization contributes positively to final throughput, and together, the three optimizations enable seven orders of magnitude speedup over the baseline. In addition, removing each optimization decreases the throughput by two to three orders of magnitude

other two order-of-magnitude speedups. These results demonstrate that ASAP's optimizations are additive, and that end-to-end, optimized streaming ASAP is approximately seven orders of magnitude faster than the baseline.

To illustrate that no one ASAP optimization is responsible for all speedups, we also report results from a lesion study, where we remove each optimization from ASAP while keeping the others enabled (Figure 11, right). Removing on-demand update, pixel-aware aggregation, and autocorrelation-enabled pruning each decreases the throughput by approximately two to three orders of magnitude, in line with results from the previous experiment. Without pixel-aware preaggregation, the algorithm makes no distinction between higher and lower resolution setting, so the results for both resolutions are near-identical similar throughput for both resolutions. In contrast, removing the other two optimizations degrades the performance for the higher resolution setting more. Thus, each of ASAP's optimizations is necessary to achieve maximum performance.

6. CONCLUSIONS

In this paper, we introduced ASAP, a new streaming operator that automatically smooths time series to reduce noise and prioritizes user's attention to systematic deviations in visualizations. We demonstrated that ASAP's target metrics—roughness and kurtosis—produce visualizations that enable users to better and faster identify deviations in time series. We also demonstrated that three optimizations—autocorrelation-based search, pixel-aware preaggregation and on-demand update—provide multiple order-of-magnitude speedups over alternatives without compromising quality. Looking forward, we are interested in adapting the metrics to additional time series visualization tasks, such as using the kurtosis measure to query "interesting" or unusual time windows in a given time series.

7. ACKNOWLEDGMENTS

We thank the many members of the Stanford InfoLab for providing feedback on this work. This research was supported in part by Toyota Research Institute, Intel, the Army High Performance Computing Research Center, RWE AG, Visa, Keysight Technologies, Facebook, VMWare, and Philips Lighting. As ASAP and MacroBase are both open source and publicly available, there is no correspondence—either direct or implied—between the use cases described in this work and the above institutions that supported this research.

8. REFERENCES

[1] Amazon CloudWatch. `https://aws.amazon.com/cloudwatch/`.

[2] Datadog. `https://www.datadoghq.com/`.

[3] Ganglia Monitoring System. `http://ganglia.info/`.

[4] Google Stackdriver. `https://cloud.google.com/stackdriver/`.

[5] Graphite. `https://graphiteapp.org/`.

[6] InfluxDB. `https://docs.influxdata.com/influxdb/`.

[7] Microsoft Azure Monitor. `https://docs.microsoft.com/azure/monitoring-and-diagnostics`.

[8] New Relic. `https://newrelic.com/`.

[9] OpenTSDB. `http://opentsdb.net/`.

[10] Prometheus. `https://prometheus.io/`.

[11] CHAPTER 15 - moving average filters. In S. W. Smith, editor, *Digital Signal Processing*. 2003.

[12] R. Agrawal, K.-I. Lin, et al. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *VLDB*, 1995.

[13] W. Aigner, S. Miksch, H. Schumann, and C. Tominski. *Visualization of time-oriented data*. Springer, 2011.

[14] M. I. Ali, F. Gao, and A. Mileo. Citybench: A configurable benchmark to evaluate rsp engines using smart city datasets. In *ISWC*, 2015.

[15] A. Arasu and J. Widom. Resource sharing in continuous sliding-window aggregates. In *VLDB*, 2004.

[16] A. Asta. Observability at Twitter: technical overview, part i, 2016. `https://blog.twitter.com/2016/observability-at-twitter-technical-overview-part-i`.

[17] V. Aurich and J. Weule. *Non-Linear Gaussian Filters Performing Edge Preserving Diffusion*. Springer, 1995.

[18] P. Bailis, E. Gan, S. Madden, D. Narayanan, K. Rong, and S. Suri. MacroBase: Prioritizing attention in fast data. In *SIGMOD*, 2017.

[19] B. Beyer, C. Jones, et al., editors. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly, 2016.

[20] C. Chatfield. *The Analysis of Time Series: An Introduction, Sixth Edition*. 2016.

[21] J. Chen, J. Benesty, et al. New insights into the noise reduction wiener filter. *TASLP*, 2006.

[22] N. Cressie. *Statistics for spatial data*. 1993.

[23] I. Daubechies. The wavelet transform, time-frequency localization and signal analysis. *IEEE Transactions on Information Theory*, 1990.

[24] L. T. DeCarlo. On the meaning and use of kurtosis. *Psychological methods*, 2(3):292, 1997.

[25] A. N. Eugene Wu. Towards perception-aware interactive data visualization systems. In *DSIA*, 2015.

[26] M. C. Ferreira de Oliveira and H. Levkowitz. From visual data exploration to visual data mining: A survey. *TVCG*, 2003.

[27] T.-c. Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011.

[28] L. Girod, K. Jamieson, et al. Wavescope: a signal-oriented data stream management system. In *SenSys*, 2006.

[29] H. Hochheiser and B. Shneiderman. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. *Information Visualization*, 2004.

[30] M. Httermann. *DevOps for developers*. Apress, 2012.

[31] V. Hulusić, G. Czanner, et al. Investigation of the beat rate effect on frame rate for animated content. In *SCCG*, 2009.

[32] R. Hyndman. Time series data library. Accessedccessed 12-Sep-2016.

[33] U. Jugel, Z. Jerzak, and other. M4: A visualization-oriented time series data aggregation. In *VLDB*, 2014.

[34] Y. Katsis, Y. Freund, and Y. Papakonstantinou. Combining databases and signal processing in plato. In *CIDR*, 2015.

[35] E. Keogh, K. Chakrabarti, et al. Dimensionality reduction for fast similarity search in large time series databases. *KAIS*, 2001.

[36] E. Keogh, J. Lin, and A. Fu. HOT SAX: Efficiently finding the most unusual time series subsequence. In *ICDM*, 2005.

[37] E. Keogh, S. Lonardi, and B. Y.-c. Chiu. Finding surprising patterns in a time series database in linear time and space. In *KDD*, 2002.

[38] E. Kreyszig. *Advanced Engineering Mathematics*. Wiley, NY, fourth edition, 1979.

[39] A. Lavin and S. Ahmad. Evaluating Real-time Anomaly Detection Algorithms - the Numenta Anomaly Benchmark. *arXiv 1510.03336*, 2015.

[40] J. Li et al. No pane, no gain: Efficient evaluation of sliding-window aggregates over data streams. *SIGMOD Rec.*, 2005.

[41] L. Li, K. Jamieson, et al. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv:1603.06560*, 2016.

[42] T. W. Liao. Clustering of time series data: a survey. *Pattern Recognition*, 2005.

[43] M. Lichman. UCI machine learning repository, 2013. Accessed 19-Aug-2016.

[44] J. Lin, E. Keogh, S. Lonardi, J. P. Lankford, and D. M. Nystrom. Visually mining and monitoring massive time series. In *KDD*, 2004.

[45] J. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *TVCG*, 2007.

[46] J. S. Marron. Automatic smoothing parameter selection: A survey. *Empirical Economics*, 13(3):187–208, 1988.

[47] M. Nikulin. Excess coefficient. Accessed 01-Oct-2016.

[48] A. Parameswaran, N. Polyzotis, and H. Garcia-Molina. SeeDB: Visualizing database queries efficiently. In *VLDB*, 2013.

[49] T. Pelkonen et al. Gorilla: A fast, scalable, in-memory time series database. In *VLDB*, 2015.

[50] W. H. Press, S. A. Teukolsky, et al. *Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing*. 1992.

[51] A. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 1964.

[52] R. H. Shumway and D. S. Stoffer. *Time Series Analysis and Its Applications*. Springer, 2005.

[53] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. Parameswaran. zenvisage: Effortless visual data exploration. In *VLDB*, 2017.

[54] J. O. Smith. *Spectral Audio Signal Processing*. 2011.

[55] F. Tajima. Determination of window size for analyzing dna sequences. *Journal of Molecular Evolution*, 1991.

[56] K. Tangwongsan, M. Hirzel, S. Schneider, and K.-L. Wu. General incremental sliding-window aggregation. In *VLDB*, 2015.

[57] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *ICCV*, 1998.

[58] Z. Wang and D. Zhang. Progressive switching median filter for the removal of impulse noise from highly corrupted images. *IEEETCASAD*, 1999.

[59] S. Weart. The carbon dioxide greenhouse effect.

[60] P. H. Westfall. Kurtosis as Peakedness, 1905–2014. RIP. *The American Statistician*, 2014.

[61] K. Wongsuphasawat, D. Moritz, et al. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *TVCG*, 2016.

[62] A. Woodie. Kafka tops 1 trillion messages per day at LinkedIn. *Datanami*, September 2015. http://www.datanami.com/2015/09/02/kafka-tops-1-trillion-messages-per-day-at-linkedin/.

[63] E. Wu, L. Battle, and S. R. Madden. The case for data visualization management systems: vision paper. In *VLDB*, 2014.

[64] Y. Zhu and D. Shasha. Efficient elastic burst detection in data streams. In *KDD*, 2003.

APPENDIX

A. CASE STUDY

In this section, we walk through a concrete example of ASAP smoothing a real world dataset. The Temp dataset contains 2976 data points, which correspond to monthly average temperatures in Central England measured from 1723 to 1970 (Figure 3, page 3). In this example, the target resolution is set to 1500 pixels.

First, ASAP aggregates the original time series using a 2 month (248 years / 1500) window via pixel-aware preaggregation. The resulting time series, shown in the top plot in Figure 3, is dominated by yearly fluctuations. Not surprisingly, the kurtosis for this time series is around 1.68, close to that of the uniform distribution.

Next, we compute the autocorrelation for this aggregated series. The autocorrelation plot peaks roughly at multiples of 6, which corresponds to the yearly period in the original time series. From here, ASAP starts the periodic search from the largest autocorrelation peak (window size = 146) within the maximum window limit (150). The window size of 146 produces a smoothed series with a kurtosis of 2.82, which satisfies our constraint. ASAP records the roughness of this smoothed series, and updates the lower bound on window size to 99 via Equation 5. At the next candidate window (window size = 140), this lower bound is improved to 104 since the corresponding autocorrelation peak is higher. From here, ASAP searches 5 more peaks until exhausting the search space. Finally, ASAP performs binary searches from the largest autocorrelation peak (146) to the maximum window size. In total, ASAP evaluates 10 candidate window sizes, while exhaustive search needs to check all 150 candidates before coming to the same conclusion that the minimum roughness is achieved at window size 140. The resulting time series is shown in the bottom plot of Figure 3. The aggregated series shows a clear rising trend after the 1880s, which corresponds to the start of global warming [59].

B. ADDITIONAL EVALUATIONS

B.1 Roughness Estimate

We first provide a full derivation for Equation 4. Given the original time series $X:\{x_1,x_2,...,x_N\}$ (a weakly stationary process), and the smoothed series $Y:\{y_1,y_2,...,y_{N-w}\}$ obtained by applying a moving average of window size w, we want to show that:

$$\text{roughness}(Y)=\frac{\sqrt{2}\sigma}{w}\sqrt{1-\frac{N}{N-w}\text{ACF}(X,w)}$$

Note that in Equation 1, when the IID assumption does not hold, $\text{cov}(X_f,X_l)\neq 0$. The covariance of discrete two random variables X,Y each with a set of N equal-probability values is defined as:

$$\text{cov}(X,Y)=\frac{1}{N}\sum_{i=1}^{N}(x_i-\mathbb{E}(X))(y_i-\mathbb{E}(Y))$$

And for a discrete process, given N equi-spaced observations of the process $x_1,x_2,...,x_N$, an estimate of the autocorrelation function at lag k can be obtained by:

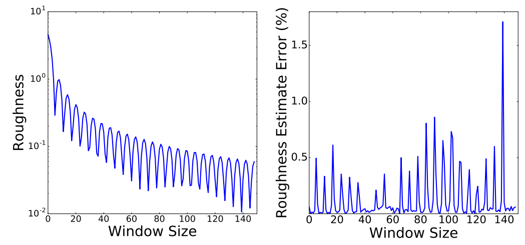


Figure B.1: True roughness and percent error of roughness estimation (Equation 4) over window sizes for dataset Temp. Here, estimate errors are with 1.2% of the true value across all window sizes.

$$\text{ACF}(X,w)=\frac{\sum_{i=1}^{N-w}(x_i-\bar{x})(x_{i+w}-\bar{x})}{\sum_{i=1}^{N}(x_i-\bar{x})^2}$$

Therefore, we can rewrite the autocorrelation function as:

$$\text{ACF}(X,w)=\frac{(N-w)\text{cov}(X_f,X_l)}{N\sigma^2},\text{ or cov}(X_f,X_l)=\frac{N\sigma^2}{N-w}\text{ACF}(X,w)$$

Substituting $\text{cov}(X_f,X_l)$ into (1), we obtain:

$$\begin{aligned}\text{roughness}(Y)&=\frac{1}{w}\sqrt{\sigma^2+\sigma^2-2\frac{N\sigma^2}{N-w}\text{ACF}(X,w)}\\&=\frac{\sqrt{2}\sigma}{w}\sqrt{1-\frac{N}{N-w}\text{ACF}(X,w)}\end{aligned}$$

In addition, we empirically evaluate the accuracy of our roughness estimation (Equation 4) on the Temp dataset, and report the relative error in percent (Figure B.1). For this time series, the roughness of the aggregated series drops sharply at window sizes around multiples of 6, which correspond to the autocorrelation peaks. Furthermore, estimated roughness (via Equation 4) is within 1.2% of the true value across all window sizes.

B.2 Pixel-aware Preaggregation

We first provide an analysis for the pixel-aware preaggregation strategy. Given a time series of length N sampled from uniform distribution and a target resolution of t pixels, we have a point-to-pixel ratio of $p_a=\frac{N}{t}$. Let w_{opt} be the window size that minimizes the roughness on the original time series. Note that searching on preaggregated data is equivalent to only selecting window sizes that are multiples of p_a. Since roughness decreases and kurtosis increases with window size, the optimal window size over the preaggregated data is $w_a=\lfloor\frac{w_{opt}}{p_a}\rfloor$, or $w_ap_a\leq w_{opt}<(w_a+1)p_a$. Therefore, $\frac{w_{opt}}{w_ap_a}<\frac{(w_a+1)p_a}{w_ap_a}=\frac{w_a+1}{w_a}$. Recall roughness scales proportionally with $\frac{1}{w}$ (Equation 2), so preaggregation incurs a penalty of no more than $\frac{w_a+1}{w_a}$ in roughness. Intuitively, as optimal window size increases, quality of preaggregation increases and in the limit, recovers the same solution as the search over the original data. We have a similar analysis for periodic data (roughness varies with $\frac{1}{w}\sqrt{1-ACF(w)}$). Let the autocorrelation corresponding to w_{opt} be ACF_{opt}, and let the maximum change in autocorrelation along a window of size p_a be ACF_Δ. Specifically, while w_{opt} may be able to pick an autocorrelation peak (i.e., a window size with high autocorrelation), searching on preaggregated data may only come within p_a of the peak. By examining the maximum rise of the autocorrelation function over a period of length p_a, we can bound the impact of roughness as above by $\frac{w_a+1}{w_a}\sqrt{\frac{1-ACF_{opt}+ACF_\Delta}{1-ACF_{opt}}}$. This implies that the

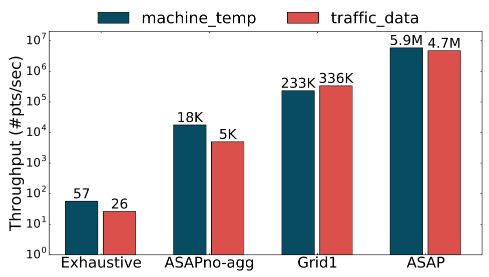


Figure B.2: Throughput of exhaustive search and ASAP on two datasets (machine_temp, traffic_data), without and with pixel-aware preaggregation for a target resolution of 1200 pixels. ASAP on preaggreaged data is up to 5 order of magnitude faster than exhaustive search on raw data (Exhaustive).

Name	Description
Exhaustive	Exhaustive search on raw time series
ASAPno-agg	ASAP on raw time series
Grid1	Exhaustive search on preaggregated data
Grid2	Exhaustive search with step size 2 on preaggregated data
Grid10	Exhaustive search with step size 10 on preaggregated data
Binary	Binary search on preaggregated data
ASAP	ASAP on preaggregated data

Table 3: Algorithms used in Section 5.2

impact of preaggregation on periodic data depends on the sharpness of the autocorrelation function, which is in turn dataset-dependent. Our empirical results confirm that both of these effects are limited on real-world datasets.

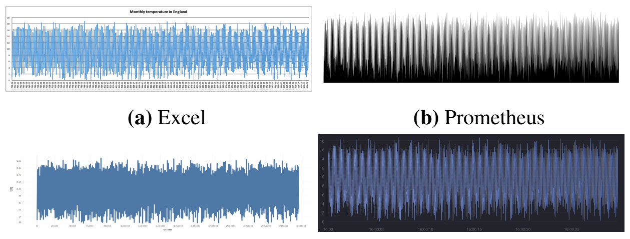
We also provide additional performance evaluations for pixel-aware preaggregation. Figure B.2 shows the throughput of running exhaustive search and ASAP on two similar sized datasets (machine_temp and traffic_data), before and after applying the pixel-aware preaggregation. With a target resolution of 1200 pixels, ASAP on aggregated series is up to 5 order of magnitude faster compared to an exhaustive search on the original time series.

C. SAMPLE VISUALIZATIONS

In this section, we present a sample of visualizations generated by commonly used monitoring systems, applications and plotting libraries for the Temp dataset.

D. USER STUDY MATERIALS

In this section, we present all time series plots and the accompanied text descriptions for the anomaly identification study (Section 5.1.1). The visual preference study (Section 5.1.2) uses a similar set of plots and text descriptions.


(**a**) Excel (**b**) Prometheus

(**c**) Tableau (**d**) Grafana

Figure C.1: Sample visualizations for the Temp dataset using various existing time series visualization tools; none automatically smoothes out the noise

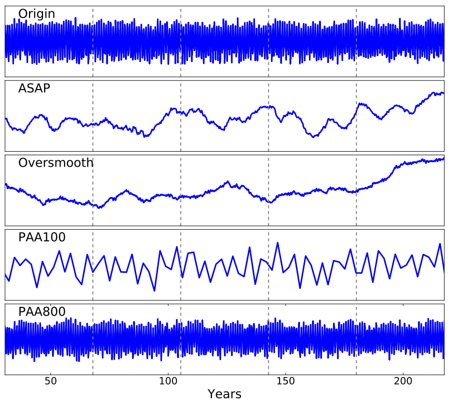


Figure D.1: User study plot for dataset Temp. We gave users the following description: "The following plot depicts the temperature recorded in England in a 250-year period. The 1880s marked the end of a protracted period of cooling called the Little Ice Age, and the overall temperature started to increase afterwards. Which region of the following plot do you think this warming trend happened?"

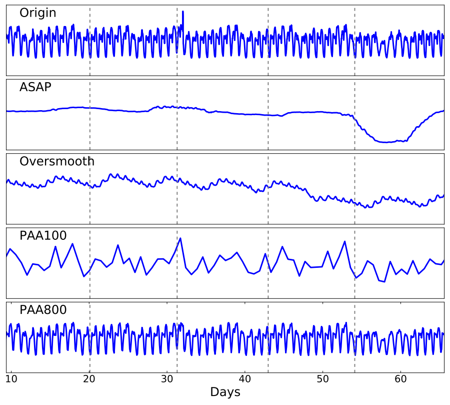


Figure D.2: User study plot for dataset Taxi. We gave users the following description: "The following plot depicts the volume of taxicab trips in New York City in a 2 month period in 2014. The volume of taxicab trips dropped sustainedly during the week of November 24th (due to Thanksgiving). Which region of the following plot do you think this sustained drop in volume happened?"

E. RAW AND SMOOTHED PLOTS

In this section, we present the remaining plots of raw and (ASAP-)smoothed time series for datasets in Table 2.

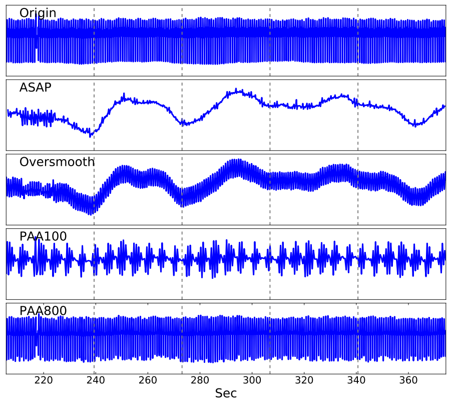


Figure D.3: User study plot for dataset EEG. We gave users the following description: “The following plot depicts 22,500 readings measuring a patient’s brainwaves (EEG activity). The EEG segment shown below contains an abnormal pattern (corresponding to a premature ventricular contraction). Which region of the following plot do you think this abnormal pattern occurred?”

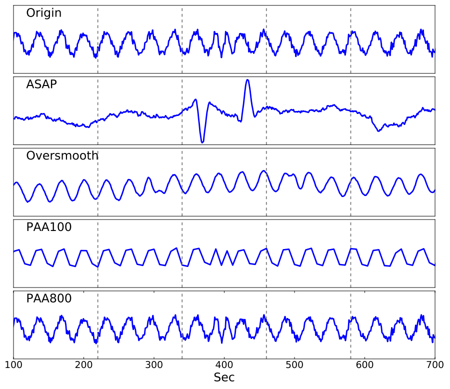


Figure D.4: User study plot for dataset Sine. We gave users the following description: “The following plot depicts 800 readings from a time varying signal. At some point, the signal experienced an unusual deviation from its regular behavior. Which region of the following plot do you think this deviation happened?”

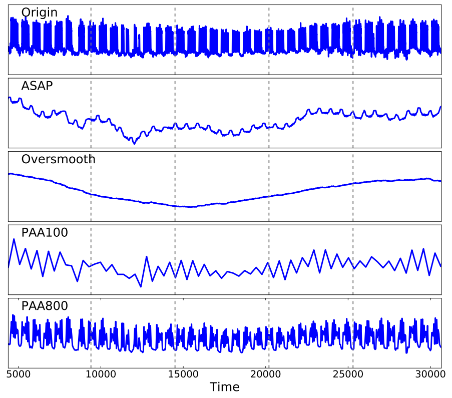


Figure D.5: User study plot for dataset Power. We gave users the following description: “The following plot depicts one year of power demand at a Dutch research facility. The power demand temporarily dips during the Ascension Thursday holiday. Which region of the following plot do you think this dip in power demand occurred?”

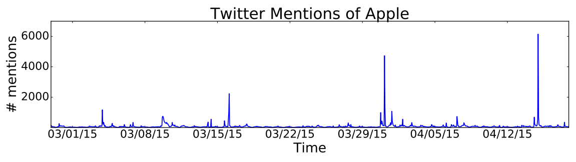
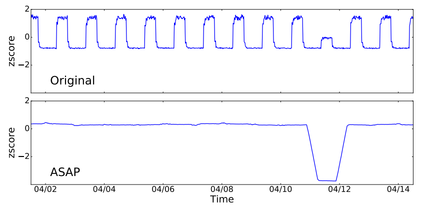
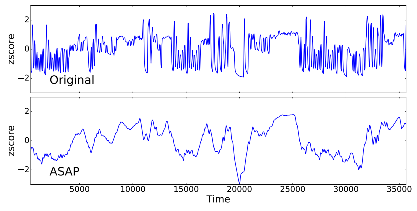


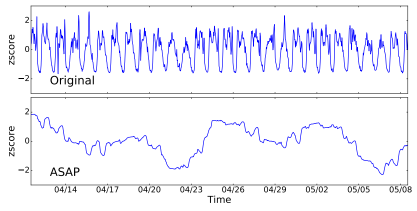
Figure E.1: Time series plot for the Twitter_AAPL data set. This plot is left unsmoothed by both exhaustive search and ASAP due to its high initial kurtosis.



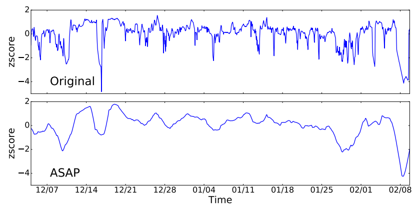
(a) sim_daily



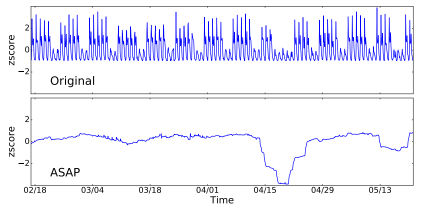
(b) gas_sensor



(c) ramp_traffic



(d) machine_temp



(e) traffic_data

Figure E.2: Original and ASAP-smoothed plots