Sequential Change Detection on Data Streams

S. Muthukrishnan
Rutgers University
muthu@cs.rutgers.edu

Eric van den Berg
Telcordia, Applied Research
evdb@research.telcordia.com

Yihua Wu
Rutgers University
yihwu@cs.rutgers.edu

Abstract

Model-based declarative queries are becoming an attractive paradigm for interacting with many data stream applications. This has led to the development of techniques to accurately answer the queries using distributional models rather than raw values. The quintessential problem with this is that of detecting when there is a change in the input stream, which makes models stale and inaccurate. We adopt the sound statistical method of sequential hypothesis testing to study this problem on streams, without independence assumption. It yields algorithms that are fast, space-efficient, and oblivious to data's underlying distributions. Our experiments demonstrate the effectiveness of our methods to not only determine the existence of a change, but also the point where the change is initiated, relative to the ground truth we obtain. Our methods work seamlessly without window limitations inherent in prior work, thus have clearly shorter delays compared to alternative window-based solutions.

1 Introduction

One of the quintessential problems of interest in monitoring streams of data in a broad range of applications is that of *change detection* [9, 15]. In IP network management, changes in the traffic patterns might indicate an intrusion, attack or an anomaly that needs attention of the network managers [12, 6]. Likewise in financial streams, a change in the pattern of trades might represent an opportunity or a warning for the analyst.

Change detection between two stored data sets in general brings up fundamental questions such as how to measure change and what is a threshold for change. In the context of data streams, additional questions arise. In particular, what are the "two" data sets that would be basis for detecting a change? Typically, data stream queries specify some window W on which the query is executed. For example, there are tumbling window queries or sliding window queries [14]. An issue is how to determine the W. Although the need of continuous detection for changes gives rise to the sliding window model [13, 7], in change detection problems, fixing a window size can only work well if information on the time-scale of change is available, but this is rarely the case. Indeed, we need scale-free detection algorithm to find changes as quickly as possible. However, this goal may be delayed by the size of the window, since a window smaller than changing rate may miss the detection or capture the change at a later time when gradual changes cumulate over time, and a window larger than changing rate delays the detection until the end of the window. One can tradeoff some performance by searching over multiple sliding windows (of different sizes) in parallel by spending some analysis time and space [13, 7]. But it is infeasible to exhaustively search over all possible window sizes. A different strategy uses a decaying-weighted window [8] to weight the importance of examples according to their age. In this case, the choice of a decay constant should match the unknown rate of change. Still, what is needed is a seamless change detection method, such that whenever a change occurs, it accurately estimates when it happens with short delay.

In this paper, we present such change detection algorithms that are inspired by the well-known *sequential hypothesis testing* [2] in Statistics. It comes with sound basis for setting the threshold for change, for providing guarantees on estimating the change point incrementally and for the minimum detection delay, under the assumptions that (1) distributions before and after a change are both known as *a priori*; (2) observations are generated independently.

Our algorithmic contribution is to design a very space-efficient, yet fast method for change detection in one pass over the data streams, where we have to relax the above two assumptions. The thorough study of our algorithms not only shows their accuracy and robustness to detect changes in one pass, but also demonstrates the accurate change-point estimate, relative to those obtained offline. Our approaches have short detection delay, compared to alternative window-based solutions.

2 Change Detection Problem

$A_1^n = (x_1 \dots x_n)$ is a sequence of input items, with $x_i \in U = \{0..u-1\}$, generated from some underlying distribution at time i. Its sub-sequence $A_1^w = (x_1 \dots x_w)$, $w < n$, is a baseline data set generated from an original

0-7695-3019-2/07 $25.00 © 2007 IEEE
DOI 10.1109/ICDMW.2007.89

distribution P_0. With P_0 known as *a priori*, two situations are possible: either every $x_i \in A_1^n$ is generated from P_0, or there exists an *unknown change point* $w < \lambda \leq n$ such that, $x_i \sim P_0$ for $i < \lambda$ and $x_i \sim P_1$ for $w < \lambda \leq i \leq n$, where P_0 and P_1 are called pre- and post-change distributions. A change in P_0 has occurred if P_1 differs significantly from P_0, which is measured by a distance function $D_\lambda(P_1||P_0)$. Here $D_\lambda(P_1||P_0)$ depends on the change-point λ.

A change detection problem is based on hypotheses: $\mathcal{H}_0$ — there is no change in P_0; and $\mathcal{H}_1$, otherwise. Indeed $\mathcal{H}_1$ is a composite of a set of "parallel" tests $\{H_\lambda | w < \lambda \leq n\}$, where H_λ is the hypothesis that a change occurs at time λ; that is, $\mathcal{H}_1 = \bigcup_{w<\lambda\leq n} H_\lambda$. In one pass over the data streams, at each x_n, a desired change detection algorithm operates in two phases: it first locates an index $w < \lambda \leq n$, which has the highest probability of being a change point (i.e., the largest $D_\lambda(P_1||P_0)$); in the second phase, the hypothesis testing compares the largest $D_\lambda(P_1||P_0)$ against a threshold to tell whether the difference is significant. Once we accept H_λ at time n, we call λ the *change point*, and n the *change detection point*; otherwise, we continue to make an additional observation.

Definition 1. *Let $(x_1, ..., x_n)$ be an input stream, and its sub-stream $(x_1, ..., x_w)$, $w < n$, is generated from a prior distribution P_0. A change detection problem is to decide whether to accept $\mathcal{H}_1$. We accept $\mathcal{H}_1$ at time n when*

$$T^n = D_{\lambda^*}(P_1||P_0) = \max_{w<\lambda\leq n} D_\lambda(P_1||P_0) \geq \tau,$$

where τ is the change detection threshold to be specified in Section 3. A change detection algorithm outputs $\lambda^* = \arg\max_{w<\lambda\leq n} D_\lambda(P_1||P_0)$ as a change-point estimate, where P_0 and P_1 are pre- and post-change distributions.

3 Preliminaries

Sequential change detection algorithms base their detection decision on the classic *Sequential Probability Ratio Test (SPRT)* [2]. When there is a change from P_0 to P_1 at point λ, it is natural that the probability of observing subsequence $A_\lambda^n = (x_\lambda, ..., x_n)$ under P_1 is "significantly" higher than that under P_0, that is, the ratio of the two probabilities is no smaller than a threshold. Given that both P_0 and P_1 are known as *a priori*, and that points in the stream are independently generated, the test statistic for testing the hypothesis H_λ against $\mathcal{H}_0$ is

$$\begin{aligned} T_\lambda^n &= D_\lambda(P_1||P_0) = \log \frac{\Pr(x_\lambda \ldots x_n|P_1)}{\Pr(x_\lambda \ldots x_n|P_0)} \\ &= \textstyle\sum_{i=\lambda}^n \log \frac{P_1[x_i]}{P_0[x_i]} = T_\lambda^{n-1} + \log \frac{P_1[x_n]}{P_0[x_n]}. \end{aligned}$$

Page's (CUSUM) procedure [2] identifies that the statistic $T^n = \max_{w<\lambda\leq n} T_\lambda^n$ obeys the recursion ($T^0 = 0$): $T^n = \max\left(0, T^{n-1} + \log \frac{P_1[x_n]}{P_0[x_n]}\right)$. Therefore, it takes each new item $O(1)$ time to update T^n and to detect a change by testing $T^n \geq \tau$. Accordingly, we update the most likely change point to n in $O(1)$ time when $T^n = 0$. When a change is detected at time n, the change-point estimate is $\lambda^* = \max\{\lambda | T^\lambda = 0, w < \lambda \leq n\}$. In addition, Wald's approximation [2] identifies the threshold $\tau = \log((1-\beta)/\alpha)$, given user-specified false alarm rate $\alpha = \Pr(\mathcal{H}_1|\mathcal{H}_0) = \Pr(T^n \geq \tau|\mathcal{H}_0)$ and miss detection rate $\beta = \Pr(\mathcal{H}_0|\mathcal{H}_1) = \Pr(T^n < \tau|\mathcal{H}_1)$.

Here T^n is an integration of the log likelihood ratios of the most recent $(n - \lambda + 1)$ observations, such that T_λ^n is maximized. Thus, it is inherent that only items after the change point contribute to T^n. And its natural advantage is to detect changes at different scales without instantiating windows of different sizes in parallel.

Count-Min (CM) sketch [5] is a small-space data structure to summarize the input, and has nearly the best time and space performance for a variety of stream analyses.

Fact 1. *With probability at least $1 - \delta$, CM sketch [5] provides ϵ approximations to point and inner product queries in $O(\frac{1}{\epsilon}\log\frac{1}{\delta})$ space. And the per-item update time is $O(\log(1/\delta))$.*

Exponential histograms (EHs) [8] are used to estimate all data counts in a recent window of size W. Datar et al provided tight bounds where $\log^2 n$ bits are sufficient and necessary for $(1+\epsilon)$ approximate estimates at time n. EH maintains buckets over ranges of data points. All data (i_t, c_t) seen in a time range $t \in (t_{i-1}, t_i]$ are aggregated into the ith bucket, and the EH maintains the value t_i and the count $C_i = \sum_{t_{i-1}<t\leq t_i} c_t$ for each bucket. Buckets where $t_i < n - W$ are discarded. When a new data point arrives, it is placed in its own new bucket. Buckets are then merged in a certain way such that there is always a logarithmic $O(\log n)$ number of buckets. At any given point n, the count estimate on the most recent W observations can be obtained from $\sum_i C_i$. Furthermore, the sequence of bucket counts (from the most- to the least-recent) is a non-decreasing sequence of powers of 2, and for some $k = \Theta(1/\epsilon)$ and some $P \leq \log\frac{2n}{k} + 1$, for each possible count $2^p < 2^P$, there are exactly k or $k+1$ buckets having count 2^p, there are at most $k+1$ buckets having count 2^P, and no buckets have counts greater than 2^P.

4 Our Change Detection Algorithms

Section 3 assumes that both P_0 and P_1 are known as *a priori*. In practice, it is fair to assume the availability of P_0, but not the prior knowledge of P_1. So (1) how to derive the post-change distribution P_1 from the stream? How to incrementally maintain it? (2) What if observations in stream are not independent, just as with streaming data? We shall address these issues in Section 4.1. Then we design a fast and small-space algorithm in Section 4.2.

4.1 Offline Algorithm

In the context of our problem, the input is a sequence of points $A_1^n = (x_1 \ldots x_w \ldots x_n)$; each $x_i \in U$. The original distribution P_0 is known as *a priori*, and is constructed

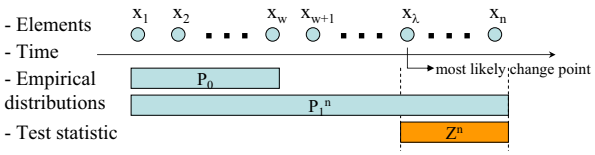

Figure 1: An illustration for our sequential method

from the first w observations $A_1^w = (x_1 \dots x_w)$. It is defined as a vector representing the relative frequency of each distinct element $j \in U$, with smoothing:

$$P_0[j] = \frac{|\{i|x_i=j, 1\le i\le w\}|+\gamma}{w+\gamma\cdot u} = \frac{S_1^w[j]+\gamma}{w+\gamma\cdot u}.$$

Here $S_1^w[j]$ denotes the frequency for item j in A_1^w. γ is usually set to 0.5 for smoothing, which controls the sensitivity to previously unseen items (i.e., $\forall j \in U,\ P_0[j] > 0$); $u = |U|$. We assume that the size w of the baseline data set is large enough to guarantee the goodness of P_0, compared to its true counterpart. [1]

Our P_1 is incrementally updated based on P_0 with new arrivals: at any time n, P_1^n is derived from A_1^n: $\forall j \in U$, $P_1^n[j] = \frac{S_1^n[j]+\gamma}{n+\gamma\cdot u}$. An inherent problem with this setup is that when a change occurs, P_1 constructed based on all observations A_1^n makes use of stale data, thus is reluctant to reflect the change promptly. Our argument is that having a change-point-dependent test statistic greatly alleviates such detection delays. We will study the effectiveness of this setup against alternative solutions in experiments. In practice, we can improve our algorithm by adopting, for example sliding window [8] or decaying window [4] models, to build more up-to-date P_1. We consider such variants orthogonal to our main line of inquiry, and do not consider it explicitly any further.

Theorem 1. *With dependent observations,* $T_\lambda^n \simeq Z_\lambda^n = \sum_{i=\lambda}^n \log \frac{P_1^i[x_i]}{P_0[x_i]}$, *where* $P_1^i[x_i] = \frac{S_1^i[x_i]+\gamma}{i+\gamma\cdot u}$; $S_1^i[x_i] = |\{j|x_j = x_i, 1 \le j \le i\}|$; $\gamma = 0.5$; $u = |U|$; *"$\simeq$" means asymptotically equal.*

Proof. The proof is based on Bayes's rule. ∎

This shows that the incremental maintenance of test statistics is by no means restricted to the independence assumption. Again, test statistic $Z^n = \max_{w<\lambda\le n} Z_\lambda^n$ obeys the recursion ($Z^0 = 0$): $Z^n = \max\left(0, Z^{n-1} + \log \frac{P_1^n[x_n]}{P_0[x_n]}\right)$. Figure 1 illustrates the basic structure of the algorithm: "base" distribution P_0 is derived from $(x_1 \dots x_w)$ as *a priori*; P_1^n is the empirical distribution over $(x_1 \dots x_n)$; test statistic Z^n integrates the change-point estimate and is evaluated only according to points after the change. Formally, at any time n:

[1] There are a variety of methods to capture the underlying distribution of a sequence. For simplicity and popularity reasons, we use empirical distributions after smoothing throughout the paper. However, our change detection methodology, as a whole, fits well with other alternatives. That is, we can isolate definition of a change and mechanism to detect it from the data representation.

- It takes $\Theta(1)$ time to update $S_1^n[x_n] \leftarrow S_1^{n-1}[x_n] + 1$ and $P_1^n[x_n]$.
- It takes $\Theta(1)$ time to update test statistic Z^n according to the recursion. If $Z^n \ge \tau$, report a detected change and the associated most likely change point λ^*. Otherwise, if $Z^n = 0$, update $\lambda^* = n$.

Moreover, the threshold τ remains unchanged, as specified in Section 3, with dependent items.

Theorem 2. *Our offline sequential change detection algorithm takes $O(u)$ space and $O(1)$ per-item processing time to detect a change, where $u = |U|$.*

4.2 Streaming Algorithm

We will use previously known *sketches* as basic components to summarize the input in small space, from which we will estimate test statistic Z^n and most likely change point λ^*. One simple attempt is to create sketches for S_1^w and S_1^n, hence to derive good estimates for P_0 and P_1^n, respectively. For every new item x_n, the estimate for test statistic Z^n is updated through $\hat{Z}^n = \max\left(0, \hat{Z}^{n-1} + \log \frac{\hat{P_1}^n[x_n]}{\hat{P}_0[x_n]}\right)$. However, this method is severely flawed in two ways. First, approximating the ratio of values in small space could perform poorly, although we have accurate estimates for values. Second, even though we obtain accurate estimate per log-ratio, the additive errors accumulated with infinite new items might lead to unbounded errors. Therefore, a more careful design of a streaming algorithm is desired.

High-level ideas. We focus on designing a sketch-based algorithm to estimate test statistic $\hat{Z}^n$ *directly* in polylogarithmic time per item. Although this is slightly weaker, compared to the offline algorithm's $O(1)$ per-item processing time, we have immediate gain in space. Unlike with the offline solution, we do not incrementally update $\hat{Z}^n$, neither does $\hat{\lambda}^*$. Thus, we will derive $\hat{\lambda}^*$ based on multiple $\hat{Z}_\lambda^n$ estimates. Instead of exhausting all $w < \lambda \le n$ (i.e., $O(n)$) points and taking the max, we probe at values of the change point λ that form a geometric progression. For example, we may estimate $\hat{Z}_\lambda^n$ only for $\lambda = n, n-1, n-2, n-4, \dots, n-2^i, \dots$. The justification is that it achieves a dramatic reduction in computation time, since we need only $O(\log n)$ estimates to keep track of. Our method will give good accuracy for λ closer to n, exactly because for such λ's we have many points to interpolate; it may give a larger error for $\lambda \ll n$, but the relative error will probably be small.

Data structures. In order to obtain a good estimate for test statistics in form of log likelihood ratios, we need to pre-process $A_1^w = (x_1 \dots x_w)$ to be aggregated to $S_1^w[i] = |j|x_j = i, 1 \le j \le w|$. And this is feasible due to our prior knowledge on P_0. As for streaming algorithm, let S_{1/P_0} be the stream whose ith entry is $S_{1/P_0}[i] = \frac{1}{P_0[i]} = \frac{w+\gamma\cdot u}{S_1^w[i]+\gamma}$. We keep a CM sketch for this "inverted" stream, denoted

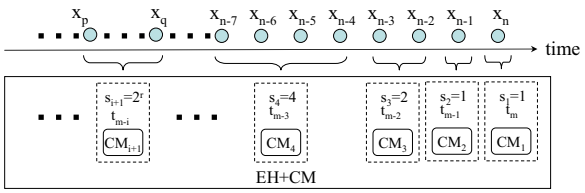

Figure 2: Sketch structure for EH+CM.

CM_{1/P_0}. It can be shown that $\forall i \in U$, $\text{CM}_{1/P_0}[i] = \min_j count[j, h_j(i)]$ is a good estimate for $1/P_0[i]$.

On the other hand, we will use EH+CM to summarize $A_1^n = (x_1 \dots x_n)$ to answer point queries $S_\lambda^n[i]$, for any $1 \le \lambda \le n$, with accuracy guarantees. We refer to this structure by EHCM_{S_1}. EH+CM is an example of *cascaded summary*, or a "sketch within a sketch": each bucket in EHs (see Section 3) contains a CM sketch to summarize data points seen in a time range $t \in (t_{i-1}, t_i]$. Associated with this CM sketch is its size s (i.e., the number of items the CM summarizes) and the timestamp t_i. The data structure is illustrated graphically in Figure 2. Each CM sketch we use takes the same set of hash functions, with $(\epsilon, \delta/\log n)$-guarantees, so they are linearly summable.

Lemma 1. *The space overhead of the EH+CM structure is* $O(\frac{\log n}{\epsilon} \log \frac{\log n}{\delta})$.

Updates. CM_{1/P_0} is constructed during pre-processing. So we only update EHCM_{S_1} for each new item as it arrives:

- Create a bucket with size 1 and the current timestamp, and initialize a CM sketch with the single new item.
- Traverse the list of buckets in order of increasing sizes (right to left). If $s_{q+2} + s_{q+1} \le \epsilon \sum_{i=1}^{q} s_i$, merge the $(q+1)$th and the $(q+2)$th buckets into a single bucket. The timestamp for this new bucket is that of the $(q+1)$th bucket; its CM sketch is the sum of CM_{q+1} and CM_{q+2}; that is, each entry $c[i,j]$ of the sketch satisfies $c'_{q+1}[i,j] = c_{q+1}[i,j] + c_{q+2}[i,j]$. We may need more than one merge.

Lemma 2. *To update the* EHCM_{S_1}*, the per-item processing time is* $O(\frac{\log n}{\epsilon} \log \frac{\log n}{\delta})$ *in the worst case, and its amortized time is* $O(\frac{1}{\epsilon} \log \frac{\log n}{\delta})$.

Proof. According to [5, 8]. ∎

Change detection. Assume that at time n, we have m buckets whose timestamps are after w, $B_1 \dots B_m$, ordered from the most- to the least-recent; each $B_i = (s_i, t_{m-i+1}, \text{CM}_i)$. The change detection algorithm consists of two components. For each new item x_n:

- Estimate $\hat{Z}_\lambda^n$ directly from sketches, for $\lambda = t_1 + 1 \dots t_{m-1} + 1$. The intuition comes from the fact that $T_\lambda^n = \sum_{i=\lambda}^{n} \log \frac{P_1[x_i]}{P_0[x_i]} = \sum_{j \in U} \left(S_\lambda^n[j] \cdot \log \frac{P_1[j]}{P_0[j]} \right)$, where $S_\lambda^n[j] = |\{i | x_i = j, \lambda \le i \le n\}|$. We first derive a CM sketch for P_1^n, denoted $\text{CM}_{P_1^n}$, by linearly combining all the sketches in EHCM_{S_1} and updating every entry in the resulting table: $c[i,j] \leftarrow \frac{c[i,j]+\gamma}{n+\gamma \cdot u}$. Next, we build a sketch $\text{CM}_{\log P_1^n/P_0} = \log(\text{CM}_{P_1^n} \times \text{CM}_{1/P_0})$, where every entry in the table has: $c_{\text{CM}_{\log P_1^n/P_0}}[i,j] = \log(c_{\text{CM}_{P_1^n}}[i,j] \cdot c_{\text{CM}_{1/P_0}}[i,j])$. Then for each $\lambda = t_q + 1$ $(q = 1..m-1)$, we compute $\text{CM}_\lambda^n = \sum_{j=q+1}^{m} \text{CM}_j$. This gives good estimate for S_λ^n. Finally, $\hat{Z}_\lambda^n = \text{median}_i \sum_j (c_{\text{CM}_\lambda^n}[i,j] \cdot c_{\text{CM}_{\log P_1^n/P_0}}[i,j])$.
- Having $\hat{Z}_\lambda^n$ for $\lambda = t_1 + 1 \dots t_{m-1} + 1$, the missing estimates for $\hat{Z}_\lambda^n$ for $w < \lambda \le n$ are approximated by interpolation with a cubic spline. After interpolation, we can use any known method to find the local maxima. If the local maximum is larger than the threshold τ, we report the change, and the maximizer is the change-point estimate.

Lemma 3. *Our streaming change detection procedure takes* $O(\frac{\log n}{\epsilon} \log \frac{\log n}{\delta})$ *per-item processing time to detect a change, and to estimate the change point when it occurs.*

5 Experiments

In this section we evaluate the efficacy of our proposed method SPRT (Section 4.1) and its streaming heuristic (Section 4.2) which we dub EHCM for detecting changes against the alternative approaches.

5.1 Experiment Setup

For control purposes, we evaluate the performance of our methods on synthetic data sets. We can apply SPRT and EHCM to various standard distributions. Due to space limit, we demonstrate results from one ubiquitous distribution family—Zipf [14]—in depth, with parameter values ranging from 0 (uniform) to 3 (highly skewed).

In our experiments, the domain size of a Zipf with parameter value z is 10,000. Ideally, there is no change in distribution if its z-value keeps constant. However with real applications, this rarely happens. For robustness, we generalize the concept of no change in distribution: we generate different amount of noise corruptions from a normal distribution $N(0, \sigma^2)$, where $\sigma^2 \in [0, 1]$, to the underlying Zipf z, denoted (z, σ^2). We typically consider two sets of distributions: $A_z = \{(z, i\sigma_0^2) | i = 0, 1, ..., 10,\ \sigma_0^2 = 0.1\}$; $B = \{(i\sigma_0^2, 0) | i = 0, 1, ..., 30,\ \sigma_0^2 = 0.1\}$. Any switch between distributions in A_z, for a given z, is considered *change-free*, whereas a data set is *change-contained* if the distribution changes from one to another within B.

In each of the following experiments, we generate 10 streams with 100,000 points each; the first and second w =50,000 points of each stream are generated from $P_0 = (z_0, 0)$ and P_1, respectively. We consider two scenarios: in the i-th change-contained stream, $P_1 = (z_0 + \Delta z, 0)$; in the i-th change-free stream, $P_1 = (z_0, \Delta z)$, $\Delta z = 0.1..1$.

5.2 Efficacy of Proposed Methods

Accuracy of detected global changes. With 10 change-contained streams, our change detection algorithms (both offline and streaming) never miss detecting a single change. So the miss detection rate on this data set is 0. We repeat the

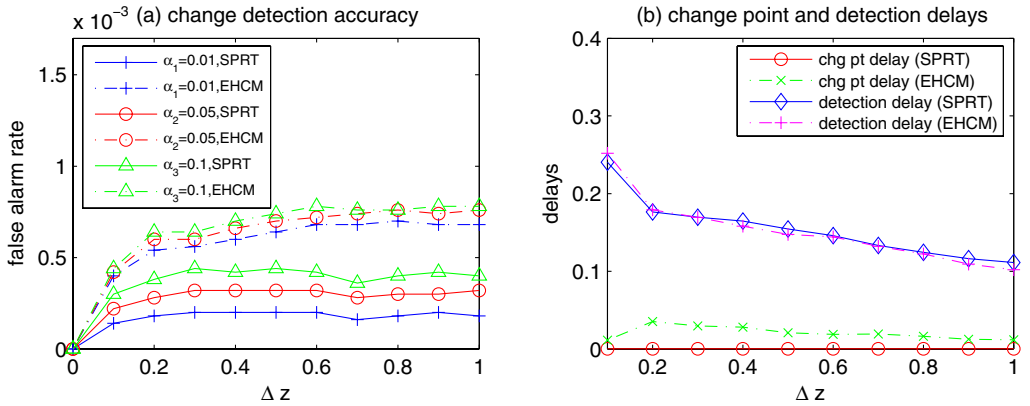

Figure 3: Performance evaluation for change detection algorithms.

experiment on change-free data sets. Ideally, there should be no alarm triggered, therefore corresponding number of alarms, after being normalized by w, indicates the false alarm rate of our algorithms. We plot it in Figure 3(a) as a function of Δz, for $z_0 = 1.2$. We also observe very similar results with other z_0 values. Here user-desired miss detection probability β is fixed to 0.05, and we vary user-desired false alarm rate α to decide threshold. Indeed, the observed false alarm rate increases as α gets larger, but is always bounded by α (in the worst case within 0.05%). It is more interesting to observe that with an increased amount of noise, the false alarm rate from change-free data sets flattens out. This reflects the robustness of our methods to noise. Moreover, the approximation error resulting from the sketch technique by comparing the false alarm rate obtained with and without sketch structure is negligible, in the worst case within an absolute difference of 0.05%.

Change point estimates and adaptive windows. For sequential methods, questions about how accurate the change point estimate is and how effective it is to adapt to various rates of change are answered in Figure 3(b). In this experiment, $P_0 = (z_0, 0)$ and $P_1 = (z_0 + \Delta z, 0)$, $\Delta z = 0.1..1$. We report results when $z_0 = 1.2$. Our change detection methods output change point estimate λ^* and detection time n for each detected change. Knowing w as a true change point, the *(normalized) change point delay* equals $\frac{|n-w|}{w}$; the *(normalized) detection delay* is $\frac{n-\lambda^*}{w}$. It is interesting to observe that all (offline) change-point estimates are close to their true values; whereas faster changes have shorter delays, as we expect. Again, streaming heuristic EHCM behaves equally well with its offline counterpart, across all Δz's.

5.3 Comparison of Methods

We consider three alternative approaches:

- **Fixed-cumulative window model (CWIN).** Given input stream $A_1^n = (x_1, ..., x_n)$, $w < n$, P_0 and P_1 are empirical probability distributions constructed from A_1^w (a fixed window) and A_1^n (a cumulative window). This setup is the same as our SPRT and EHCM. But CWIN computes KL-distance [7] between P_0 and P_1 at every new item x_n, and signal an alarm when the distance is significantly large.
- **Fixed-sliding window model (FSWIN).** This algorithm has a similar flavor to CWIN, but differs in the way that P_1 is built based on the most recent w observations $A_{n-w+1}^n = (x_{n-w+1} \ldots x_n)$, which forms a sliding window of size w.
- **Benchmark Sequential Method (BSM).** This approach assumes that not only P_0 but also P_1 is known as *a priori*, so that it guarantees optimum in terms of detection delay.

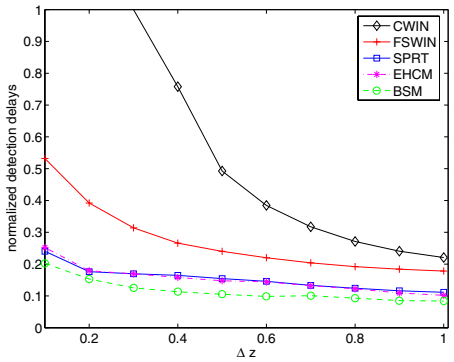

Figure 4: Normalized detection delays of various methods.

Detection delay. We compare them against our SPRT and EHCM. Again in this experiment, $P_0 = (z_0, 0))$, $P_1 = (z_0 + \Delta z, 0)$, $\Delta z = 0.1..1$, and the true change point is at w. Without knowing the true threshold to trigger significant changes with window-based methods (i.e., CWIN and FSWIN), we adopt a conservative threshold — the KL-distance between distributions $(z_0, 0)$ and $(z_0, 1)$. This is smaller than the true threshold, since we consider it change-free by adding random noise $N(0, 1)$ to the original data. So the detection delay using the conservative threshold is shorter than the true delay. We compare it against detection delays from other methods, and infer the comparative results when using the true threshold.

All methods start detecting changes since x_{w+1}. Figure 4 demonstrates the results when $z_0 = 1.2$. As expected, the

detection delay of each method decreases with the increase of Δz. CWIN yields the longest detection delay, and even misses detections in the first three streams when change is small. This is because points prior to the change point decay the amount of change in distribution. Therefore FSWIN performs better, since it forgets all stale data that arrive at least w time steps ago. And we can infer that delays from a change detection algorithm with decaying-weighted window will fall between those from FSWIN and CWIN. However, FSWIN's detection delay is still roughly twice of the delay from equally-well-performed SPRT and EHCM. This indicates the effectiveness of a test statistic when the change-point estimate is integrated, even though the estimate for the associated P_1 may be out-of-date to some degree. Due to the use of a conservative threshold for all window-based solutions, their true gap from sequential methods should get enlarged with the real threshold. There is no surprise that BSM always achieves the minimum delay, but it is interesting to see that detection delays from SPRT/EHCM are mostly close to BSM. This is another strong indicator of prompt response of sequential approaches to changes. Moreover, the threshold τ does not depend on different change scales in above experiments.

6 Related Work

There is a lot of work on change detection in data streams. Most [10, 13, 7, 6] are based on fixed-size windows, where a delay in change detection is inevitable when the time-scale of change is unavailable. Authors in [10, 13, 7] exploited parallel windows of different sizes to adapt to different change scales. But this is not suitable in data stream context, since we can not afford to explore all window sizes. On the other hand, it has been proposed to use windows of varied sizes [9, 15, 1]. This line of research has to guess *a priori* the time-scale of change, and is mostly computationally expensive. Ho [11] recently proposed a martingale framework for change detection on data streams, where a "pseudo-adaptive" window — delimited by two adjacent change detection points — was used. However, no change-point estimate was considered in this work. In contrast to the bulk of this literature, our approaches differ in two key ways. First, when a change is detected, our methods also estimate the most likely change point in time. Second, the change-point estimate is inherent to the evaluation of our test statistic to detect a change, hence, our methods achieve shorter detection delays.

Applying sequential hypothesis testing to change detection is a natural approach. [12] studied sequential change detection method offline, under a simpler scenario: (1) both pre- and post-change distributions are known as *a priori*; (2) the first observation is assumed to be the change point. Our work here is more general, being able to estimate the actual change point anywhere which is the crux of the problem. In addition, our challenge is to adapt the well-known technique to the streaming scenario, even without independence assumption. Another relevant work is [3], where they adopted exponential histograms to detect change of various scales. However, their approach does not work for data with large domain size, which is our crucial focus.

7 Conclusions

There have been plenty of work on techniques for detecting changes in data, but most, if not all, focus on certain "windows" to define and study changes in distributions. This is limiting since window size is a parameter that needs to be determined which is nontrivial, and its size is a lower bound on the detection delay. We have adopted the sound statistical method of sequential hypothesis testing to yield fast and space-efficient change detection algorithms on streams, without explicit window specification. Our methods are oblivious to data's underlying distributions, and work without independence assumption. Our detailed study of these methods reveals their effectiveness in detecting changes.

References

[1] P.L. Bartlett, S. Ben-David, and S.R. Kulkarni. Learning changing concepts by exploiting the structure of change. In *Computational Learing Theory*, pages 131–139, 1996.

[2] M. Basseville and I.V. Nikiforov. *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall, Inc.

[3] A. Bifet and R. Gavalda. Learning from time-changing data with adaptive windowing. In *SDM*, 2007.

[4] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *PODS*, 2003.

[5] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. In *LATIN*, 2004.

[6] G. Cormode and S. Muthukrishnan. What is new: Finding significant differences in network data streams. In *INFOCOM*, pages 1534–1545, 2004.

[7] T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi. An information-theoretic approach to detecting changes in multi-dimensional data streams. In *Interface*, 2006.

[8] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *SODA*, pages 635–644, 2002.

[9] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *Lecture Notes in Computer Science*, 2004.

[10] V. Ganti, J.E. Gehrke, R. Ramakrishnan, and W.-Y. Loh. A framework for measuring changes in data characteristics. *Journal of Computer and System Sciences*, 64:542–578, 2002.

[11] S. Ho. A martingale framework for concept change detection in time-varying data streams. In *ICML*, pages 321–327, 2005.

[12] J. Jung, V. Paxson, A.W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *IEEE symposium on security and privacy*, 2004.

[13] D. Kifer, S. Ben-David, and J. Gehrke. Detecting changes in data streams. In *VLDB*, 2004.

[14] S. Muthukrishnan. Data streams: Algorithms and applications. In *SODA*, 2003.

[15] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.