Streaming Pattern Discovery in Multiple Time-Series

Spiros Papadimitriou Jimeng Sun Christos Faloutsos*

Computer Science Department,
Carnegie Mellon University,
Pittsburgh, PA, USA.
{spapadim,jimeng,christos}@cs.cmu.edu

Abstract

In this paper, we introduce SPIRIT (Streaming Pattern dIscoveRy in multIple Time-series). Given n numerical data streams, all of whose values we observe at each time tick t, SPIRIT can incrementally find correlations and hidden variables, which summarise the key trends in the entire stream collection. It can do this quickly, with no buffering of stream values and without comparing pairs of streams. Moreover, it is any-time, single pass, and it dynamically detects changes. The discovered trends can also be used to immediately spot potential anomalies, to do efficient forecasting and, more generally, to dramatically simplify further data processing. Our experimental evaluation and case studies show that SPIRIT can incrementally capture correlations and discover trends, efficiently and effectively.

1 Introduction

Data streams have received considerable attention in various communities (theory, databases, data mining, networking, systems), due to several important applications, such as network analysis [11], sensor network monitoring [38], moving object tracking [2], financial data analysis [41], and scientific data processing [42]. All these applications have in common that: (i) massive amounts of data arrive at high rates, which makes traditional database systems prohibitively slow, and (ii) users, or higher-level applications, require immediate responses and cannot afford any post-processing (e.g., in network intrusion detection). Data stream systems have been prototyped [1, 29, 9] and deployed in practice [11].

In addition to providing SQL-like support for data stream management systems (DSMS), it is crucial to detect patterns and correlations that may exist in co-evolving data streams. Streams often are inherently correlated (e.g., temperatures in the same building, traffic in the same network, prices in the same market, etc.) and it is possible to reduce hundreds of numerical streams into just a handful of *hidden variables* that compactly describe the key trends and dramatically reduce the complexity of further data processing. We propose an approach to do this incrementally.

1.1 Problem motivation

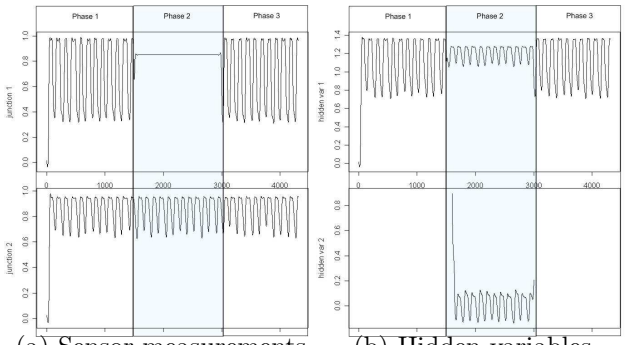
We consider the problem of capturing correlations and finding hidden variables corresponding to trends on collections of semi-infinite, time series data streams, where the data consist of tuples with n numbers, one for each time tick t.

We describe a motivating scenario, to illustrate the problem we want to solve. Consider a large number of sensors measuring chlorine concentration in a drinkable water distribution network (see Figure 1, showing 15 days worth of data). Every five minutes, each sensor sends its measurement to a central node, which monitors and analyses the streams in real time.

The patterns in chlorine concentration levels normally arise from water demand. If water is not refreshed in the pipes, existing chlorine reacts with pipe walls and micro-organisms and its concentration drops. However, if fresh water flows in at a particular location due to demand, chlorine concentration rises again. The rise depends primarily on how much chlorine is

*This material is based upon work supported by the National Science Foundation under Grants No. IIS-0083148, IIS-0209107, IIS-0205224, INT-0318547, SENSOR-0329549, EF-0331657, IIS-0326322, NASA Grant AIST-QRS-04-3031, CNS-0433540. This work is supported in part by the Pennsylvania Infrastructure Technology Alliance (PITA), a partnership of Carnegie Mellon, Lehigh University and the Commonwealth of Pennsylvania's Department of Community and Economic Development (DCED). Additional funding was provided by donations from Intel, and by a gift from Northrop-Grumman Corporation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties.

Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005



(a) Sensor measurements (b) Hidden variables

Figure 1: *Illustration of problem. Sensors measure chlorine in drinking water and show a daily, near sinusoidal periodicity during phases 1 and 3. During phase 2, some of the sensors are "stuck" due to a major leak. The extra hidden variable introduced during phase 2 captures the presence of a new trend. SPIRIT can also tell us which sensors participate in the new, "abnormal" trend (e.g., close to a construction site). In phase 3, everything returns to normal.*

originally mixed at the reservoirs (and also, to a small extent, on the distance to the closest reservoir—as the distance increases, the peak concentration drops slightly, due to reactions along the way). Thus, since demand typically follows a periodic pattern, chlorine concentration reflects that (see Figure 1(a), bottom): it is high when demand is high and vice versa.

Assume that at some point in time, there is a major leak at some pipe in the network. Since fresh water flows in constantly (possibly mixed with debris from the leak), chlorine concentration at the nodes near the leak will be close to peak at all times.

Figure 1(a) shows measurements collected from two nodes, one away from the leak (bottom) and one close to the leak (top). At any time, a human operator would like to know how many trends (or *hidden variables*) are in the data and ask queries about them. Each hidden variable essentially corresponds to a group of correlated streams.

In this simple example, SPIRIT discovers the correct number of hidden variables. Under normal operation, only one hidden variable is needed, which corresponds to the periodic pattern (Figure 1(b), top). Both observed variables follow this hidden variable (multiplied by a constant factor, which is the *participation weight* of each observed variable into the particular hidden variable). Mathematically, the hidden variables are the *principal components* of the observed variables and the participation weights are the entries of the *principal direction* vectors[1].

However, during the leak, a second trend is detected and a new hidden variable is introduced (Figure 1(b), bottom). As soon as the leak is fixed, the number of hidden variables returns to one. If we examine the hidden variables, the interpretation is straightforward: The first one still reflects the periodic demand pattern in the sections of the network under normal operation. All nodes in this section of the network have a participation weight of ≈ 1 to the "periodic trend" hidden variable and ≈ 0 to the new one. The second hidden variable represents the additive effect of the catastrophic event, which is to cancel out the normal pattern. The nodes close to the leak have participation weights ≈ 0.5 to both hidden variables.

Summarising, SPIRIT can tell us that (Figure 1):

- Under normal operation (phases 1 and 3), there is one trend. The corresponding hidden variable follows a periodic pattern and all nodes participate in this trend. All is well.
- During the leak (phase 2), there is a *second* trend, trying to cancel the normal trend. The nodes with non-zero participation to the corresponding hidden variable can be immediately identified (e.g., they are close to a construction site). An abnormal event may have occurred in the vicinity of those nodes, which should be investigated.

Matters are further complicated when there are hundreds or thousands of nodes and more than one demand pattern. However, as we show later, SPIRIT is still able to extract the key trends from the stream collection, follow trend drifts and immediately detect outliers and abnormal events.

Besides providing a concise summary of key trends/correlations among streams, SPIRIT can successfully deal with missing values and its discovered hidden variables can be used to do very efficient, resource-economic forecasting.

Of course, there are several other applications and domains to which SPIRIT can be applied. For example, (i) given more than 50,000 securities trading in US, on a second-by-second basis, detect patterns and correlations [41], (ii) given traffic measurements [39], find routers that tend to go down together.

1.2 Contributions

The problem of pattern discovery in a large number of co-evolving streams has attracted much attention in many domains. We introduce *SPIRIT (Streaming Pattern dIscoveRy in multIple Time-series)*, a comprehensive approach to discover correlations that effectively and efficiently summarise large collections of streams. SPIRIT satisfies the following requirements:

- It is *streaming*, i.e., it is incremental, scalable, *any-time*. It requires very memory and processing time per time tick. In fact, both are independent of the stream length t.
- It scales *linearly* with the number of streams n, not quadratically. This may seem counter-intuitive, because the naïve method to spot correlations across n streams examines all $O(n^2)$ pairs.

[1] More precisely, this is true under certain assumptions, which will be explained later.

- It is *adaptive*, and fully *automatic*. It dynamically detects changes (both gradual, as well as sudden) in the input streams, and automatically determines the number k of hidden variables.

The correlations and hidden variables we discover have multiple uses. They provide a succinct summary to the user, they can help to do fast forecasting and detect outliers, and they facilitate interpolations and handling of missing values, as we discuss later.

The rest of the paper is organised as follows: Section 2 discusses related work, on data streams and stream mining. Section 3 overviews some of the background and explains the intuition behind our approach. Section 4 describes our method and Section 5 shows how its output can be interpreted and immediately utilised, both by humans, as well as for further data analysis. Section 6 discusses experimental case studies that demonstrate the effectiveness of our approach. In Section 7 we elaborate on the efficiency and accuracy of SPIRIT. Finally, in Section 8 we conclude.

2 Related work

There is a large body of work on streams, which we loosely classify in two groups.

Data stream management systems (DSMS). We include this very broad category for completeness. DSMS include Aurora [1], Stream [29], Telegraph [9] and Gigascope [11]. The common hypothesis is that (i) massive data streams come into the system at a very fast rate, and (ii) near real-time monitoring and analysis of incoming data streams is required. The new challenges have made researchers re-think many parts of traditional DBMS design in the streaming context, especially on query processing using correlated attributes [14], scheduling [6, 8], load shedding [35, 12] and memory requirements [5].

In addition to system-building efforts, a number of approximation techniques have been studied in the context of streams, such as sampling [7], sketches [16, 10, 19], exponential histograms [13], and wavelets [22]. The main goal of these methods is to estimate a global aggregate (e.g. sum, count, average) over a window of size w on the recent data. The methods usually have resource requirements that are sublinear with respect to w. Most focus on a single stream.

The emphasis in this line of work is to support traditional SQL queries on streams. None of them try to find patterns, nor to do forecasting.

Data mining on streams. Researchers have started to redesign traditional data mining algorithms for data streams. Much of the work has focused on finding interesting patterns in a single stream, but multiple streams have also attracted significant interest. Ganti et al. [20] propose a generic framework for stream mining. Guha et al. [23] propose a one-pass k-median clustering algorithm. Domingos and Hulten [17] construct a decision tree online, by passing over the data only once. Recently, [25, 36] address the problem of finding patterns over concept drifting streams. Papadimitriou et al. [32] proposed a method to find patterns in a single stream, using wavelets. More recently, Palpanas et al. [31] consider approximation of time-series with *amnesic* functions. They propose novel techniques suitable for streaming, and applicable to a wide range of user-specified approximating functions.

Keogh et al. [27] propose parameter-free methods for classic data mining tasks (i.e., clustering, anomaly detection, classification), based on compression. Lin et al. [28] perform clustering on different levels of wavelet coefficients of multiple time series. Both approaches require having all the data in advance. Recently, Ali et al. [4] propose a framework for *Phenomena Detection and Tracking (PDT)* in sensor networks. They define a phenomenon on descrete-valued streams and develop query execution techniques based on multi-way hash join with PDT-specific optimizations.

CluStream [3] is a flexible clustering framework with online and offline components. The online component extends micro-cluster information [40] by incorporating exponentially-sized sliding windows while coalescing micro-cluster summaries. Actual clusters are found by the offline component. StatStream [41] uses the DFT to summarise streams within a finite window and then compute the highest pairwise correlations among all pairs of streams, at each timestamp. Very recently, BRAID [33] addresses the problem of discovering lag correlations among multiple streams. The focus is on time and space efficient methods for finding the earliest and highest peak in the cross-correlation functions beteeen all pairs of streams. Neither CluStream, StatStream or BRAID explicitly focus on discovering hidden variables.

Guha et al. [21] improve on discovering correlations, by first doing dimensionality reduction with random projections, and then periodically computing the SVD. However, the method incurs high overhead because of the SVD re-computation and it can not easily handle missing values. MUSCLES [39] is exactly designed to do forecasting (thus it could handle missing values). However, it can not find hidden variables and it scales poorly for a large number of streams n, since it requires at least quadratic space and time, or expensive reorganisation (*selective MUSCLES*).

Finally, a number of the above methods usually require choosing a sliding window size, which typically translates to buffer space requirements. Our approach does not require any sliding windows and does not need to buffer *any* of the stream data.

In conclusion, none of the above methods simultaneously satisfy the requirements in the introduction: "any-time" streaming operation, scalability on the number of streams, adaptivity, and full automation.

3 Principal component analysis (PCA)

Here we give a brief overview of PCA [26] and explain the intuition behind our approach. We use standard matrix algebra notation: vectors are lower-case bold, matrices are upper-case bold, and scalars are in plain font. The transpose of matrix $\mathbf{X}$ is denoted by $\mathbf{X}^T$. In the following, $\mathbf{x}_t \equiv [x_{t,1}\, x_{t,2} \cdots x_{t,n}]^T \in \mathbb{R}^n$ is the column-vector[2] of stream values at time t. The stream data can be viewed as a continuously growing $t \times n$ matrix $\mathbf{X}_t := [\mathbf{x}_1\, \mathbf{x}_2 \cdots \mathbf{x}_t]^T \in \mathbb{R}^{t \times n}$, where one new row is added at each time tick t. In the chlorine example, $\mathbf{x}_t$ is the measurements column-vector at t over all the sensors, where n is the number of chlorine sensors and t is the measurement timestamp.

Typically, in collections of n-dimensional points $\mathbf{x}_t \equiv [x_{t,1} \ldots, x_{t,n}]^T$, $t = 1, 2, \ldots$, there exist correlations between the n dimensions (which correspond to streams in our setting). These can be captured by principal components analysis (PCA). Consider for example the setting in Figure 2. There is a visible linear correlation. Thus, if we represent every point with its projection on the direction of $\mathbf{w}_1$, the error of this approximation is very small. In fact, the first principal direction $\mathbf{w}_1$, is the *optimal* in the following sense.

Definition 3.1 (First principal component). *Given a collection of n-dimensional vectors $\mathbf{x}_\tau \in \mathbb{R}^n$, $\tau = 1, 2, \ldots, t$, the* first principal direction $\mathbf{w}_1 \in \mathbb{R}^n$ *is the vector that minimizes the sum of squared residuals, i.e.,*

$$\mathbf{w}_1 := \arg\min_{\|w\|=1} \sum_{\tau=1}^{t} \|\mathbf{x}_\tau - (\mathbf{w}\mathbf{w}^T)\mathbf{x}_\tau\|^2.$$

The projection of $\mathbf{x}_\tau$ on $\mathbf{w}_1$ is the first principal component (PC) $y_{\tau,1} := \mathbf{w}_1^T \mathbf{x}_\tau$, $\tau = 1, \ldots, t$.

Note that, since $\|\mathbf{w}_1\| = 1$, we have $(\mathbf{w}_1\mathbf{w}_1^T)\mathbf{x}_\tau = (\mathbf{w}_1^T\mathbf{x}_\tau)\mathbf{w}_1 = y_{\tau,1}\mathbf{w}_1 =: \tilde{\mathbf{x}}_\tau$, where $\tilde{\mathbf{x}}_\tau$ is the projection of $y_{\tau,1}$ back into the original n-D space. That is, $\tilde{\mathbf{x}}_\tau$ is the *reconstruction* of the original measurements from the first PC $y_{\tau,1}$. More generally, PCA will produce k vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k$ such that, if we represent each n-D data point $\mathbf{x}_t := [x_{t,1} \cdots x_{t,n}]$ with its k-D projection $\mathbf{y}_t := [\mathbf{w}_1^T\mathbf{x}_t \cdots \mathbf{w}_k^T\mathbf{x}_t]^T$, then this representation minimises the squared error $\sum_\tau \|\mathbf{x}_t - \tilde{\mathbf{x}}_t\|^2$. Furthermore, the principal directions are orthogonal, so the principal components $y_{\tau,i}, 1 \le i \le k$ are *by construction uncorrelated*, i.e., if $\mathbf{y}^{(i)} := [y_{1,i}, \ldots, y_{t,i}, \ldots]^T$ is the stream of the i-th principal component, then $\left(\mathbf{y}^{(i)}\right)^T \mathbf{y}^{(j)} = 0$ if $i \neq j$.

Observation 3.1 (Dimensionality reduction). *If we represent each n-dimensional point $\mathbf{x}_\tau \in \mathbb{R}^n$ using all n principal components, then the error $\|\mathbf{x}_\tau - \tilde{\mathbf{x}}_\tau\| = 0$. However, in typical datasets, we can achieve a very small error using only k principal components, where $k \ll n$.*

[2]We adhere to the common convention of using column vectors and writing them out in transposed form.

Symbol	Description
$\mathbf{x}, \ldots$	Column vectors (lowercase boldface).
$\mathbf{A}, \ldots$	Matrices (uppercase boldface).
$\mathbf{x}_t$	The n stream values $\mathbf{x}_t := [x_{t,1} \cdots x_{t,n}]^T$ at time t.
n	Number of streams.
$\mathbf{w}_i$	The i-th participation weight vector (i.e., principal direction).
k	Number of hidden variables.
$\mathbf{y}_t$	Vector of hidden variables (i.e., principal components) for $\mathbf{x}_t$, i.e., $\mathbf{y}_t \equiv [y_{t,1} \cdots y_{t,k}]^T := [\mathbf{w}_1^T\mathbf{x}_t \cdots \mathbf{w}_k^T\mathbf{x}_t]^T$.
$\tilde{\mathbf{x}}_t$	Reconstruction of $\mathbf{x}_t$ from the k hidden variable values, i.e., $\tilde{\mathbf{x}}_t := y_{t,1}\mathbf{w}_1 + \cdots + y_{t,k}\mathbf{w}_k$.
E_t	Total energy up to time t.
$\tilde{E}_{t,i}$	Total energy captured by the i-th hidden variable, up to time t.
f_E, F_E	Lower and upper bounds on the fraction of energy we wish to maintain via SPIRIT's approximation.

Table 1: *Description of notation.*

In the context of the chlorine example, each point in Figure 2 would correspond to the 2-D projection of $\mathbf{x}_\tau$ (where $1 \le \tau \le t$) onto the first two principal directions, $\mathbf{w}_1$ and $\mathbf{w}_2$, which are the most important according to the distribution of $\{\mathbf{x}_\tau \mid 1 \le \tau \le t\}$. The principal components $y_{\tau,1}$ and $y_{\tau,2}$ are the coordinates of these projections in the orthogonal coordinate system defined by $\mathbf{w}_1$ and $\mathbf{w}_2$.

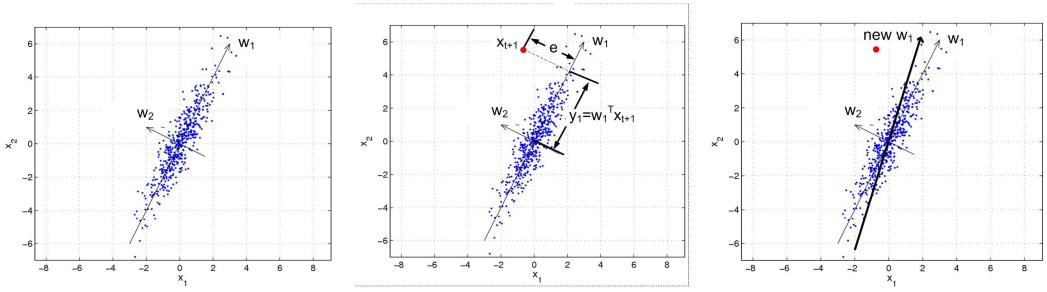
However, batch methods for estimating the principal components require time that depends on the duration t, which grows to infinity. In fact, the principal directions are the eigenvectors of $\mathbf{X}_t^T\mathbf{X}_t$, which are best computed through the singular value decomposition (SVD) of $\mathbf{X}_t$. Space requirements also depend on t. Clearly, in a stream setting, it is impossible to perform this computation at every step, aside from the fact that we don't have the space to store all past values. In short, we want a method that does not need to store *any* past values.

4 Tracking correlations and hidden variables: SPIRIT

In this section we present our framework for discovering patterns in multiple streams. In the next section, we show how these can be used to perform effective, low-cost forecasting. We use auto-regression for its simplicity, but our framework allows any forecasting algorithm to take advantage of the compact representation of the stream collection.

Problem definition. Given a collection of n co-evolving, semi-infinite streams, producing a value $x_{t,j}$, for every stream $1 \le j \le n$ and for every time-tick $t = 1, 2, \ldots$, SPIRIT does the following:

- Adapts the number k of *hidden variables* necessary to explain/summarise the main trends in the collection.
- Adapts the *participation weights* $w_{i,j}$ of the j-th stream on the i-th hidden variable ($1 \le j \le n$ and



(a) Original $\mathbf{w}_1$ (b) Update process (c) Resulting $\mathbf{w}_1$

Figure 2: *Illustration of updating* $\mathbf{w}_1$ *when a new point* $\mathbf{x}_{t+1}$ *arrives.*

$1 \leq i \leq k$), so as to produce an accurate summary of the stream collection.

- Monitors the hidden variables $y_{t,i}$, for $1 \leq i \leq k$.
- Keeps updating all the above efficiently.

More precisely, SPIRIT operates on the column-vectors of observed stream values $\mathbf{x}_t \equiv [x_{t,1}, \ldots, x_{t,n}]^T$ and continually updates the participation weights $w_{i,j}$. The *participation weight vector* $\mathbf{w}_i$ for the i-th principal direction is $\mathbf{w}_i := [w_{i,1} \cdots w_{i,n}]^T$. The hidden variables $\mathbf{y}_t \equiv [y_{t,1}, \ldots, y_{t,k}]^T$ are the projections of $\mathbf{x}_t$ onto each $\mathbf{w}_i$, over time (see Table 1), i.e.,

$$y_{t,i} := w_{i,1}x_{t,1} + w_{i,2}x_{t,2} + \cdots + w_{i,n}x_{t,n},$$

SPIRIT also adapts the number k of hidden variables necessary to capture most of the information. The adaptation is performed so that the approximation achieves a desired mean-square error. In particular, let $\tilde{\mathbf{x}}_t = [\tilde{x}_{t,1} \cdots \tilde{x}_{t,n}]^T$ be the *reconstruction* of $\mathbf{x}_t$, based on the weights and hidden variables, defined by

$$\tilde{x}_{t,j} := w_{1,j}y_{t,1} + w_{2,j}y_{t,2} + \cdots + w_{k,j}y_{t,k},$$

or more succinctly, $\tilde{\mathbf{x}}_t = \sum_{i=1}^{k} y_{i,t}\mathbf{w}_i$.

In the chlorine example, $\mathbf{x}_t$ is the n-dimensional column-vector of the original sensor measurements and $\mathbf{y}_t$ is the hidden variable column-vector, both at time t. The dimension of $\mathbf{y}_t$ is 1 before/after the leak ($t < 1500$ or $t > 3000$) and 2 during the leak ($1500 \leq t \leq 3000$), as shown in Figure 1.

Definition 4.1 (SPIRIT tracking). *SPIRIT updates the participation weights* $w_{i,j}$ *so as to guarantee that the reconstruction error* $\|\tilde{\mathbf{x}}_t - \mathbf{x}_t\|^2$ over time *is predictably small.*

This informal definition describes what SPIRIT does. The precise criteria regarding the reconstruction error will be explained later. If we assume that the $\mathbf{x}_t$ are drawn according to some distribution that does not change over time (i.e., under *stationarity* assumptions), then the weight vectors $\mathbf{w}_i$ converge to the principal directions. However, even if there are non-stationarities in the data (i.e., gradual drift), in practice we can deal with these very effectively, as we explain later.

An additional complication is that we often have missing values, for several reasons: either failure of the system, or delayed arrival of some measurements. For example, the sensor network may get overloaded and fail to report some of the chlorine measurements in time or some sensor may temporarily black-out. At the very least, we want to continue processing the rest of the measurements.

4.1 Tracking the hidden variables

The first step is, for a given k, to incrementally update the k participation weight vectors $\mathbf{w}_i$, $1 \leq i \leq k$, so as to summarise the original streams with only a few numbers (the hidden variables). In Section 4.2, we describe the complete method, which also adapts k.

For the moment, assume that the number of hidden variables k is given. Furthermore, our goal is to minimize the average reconstruction error $\sum_t \|\tilde{\mathbf{x}}_t - \mathbf{x}_t\|^2$. In this case, the desired weight vectors $\mathbf{w}_i, 1 \leq i \leq k$ are the principal directions and it turns out that we can estimate them incrementally.

We use an algorithm based on adaptive filtering techniques [37, 24], which have been tried and tested in practice, performing well in a variety of settings and applications (e.g., image compression and signal tracking for antenna arrays). We experimented with several alternatives [30, 15] and found this particular method to have the best properties for our setting: it is very efficient in terms of computational and memory requirements, while converging quickly, with no special parameters to tune. The main idea behind the algorithm is to read in the new values $\mathbf{x}_{t+1} \equiv [x_{(t+1),1}, \ldots, x_{(t+1),n}]^T$ from the n streams at time $t+1$, and perform three steps:

1. Compute the hidden variables $y'_{t+1,i}, 1 \leq i \leq k$, based on the *current* weights $\mathbf{w}_i, 1 \leq i \leq k$, by projecting $\mathbf{x}_{t+1}$ onto these.
2. Estimate the reconstruction error ($\mathbf{e}_i$ below) and the energy, based on the $y'_{t+1,i}$ values.
3. Update the estimates of $\mathbf{w}_i, 1 \leq i \leq k$ and output the *actual* hidden variables $y_{t+1,i}$ for time $t+1$.

To illustrate this, Figure 2(b) shows the $\mathbf{e}_1$ and y_1 when the new data $\mathbf{x}_{t+1}$ enter the system. Intuitively, the goal is to adaptively update $\mathbf{w}_i$ so that it quickly converges to the "truth." In particular, we want to update $\mathbf{w}_i$ more when $\mathbf{e}_i$ is large. However, the magnitude of the update should also take into account the past data currently "captured" by $\mathbf{w}_i$. For this reason, the update is inversely proportional to the current *energy* $E_{t,i}$ of the i-th hidden variable, which is

$E_{t,i} := \frac{1}{t}\sum_{\tau=1}^{t} y_{\tau,i}^2$. Figure 2(c) shows $\mathbf{w}_1$ after the update for $\mathbf{x}_{t+1}$.

Algorithm TRACKW

0. Initialise the k hidden variables $\mathbf{w}_i$ to unit vectors $\mathbf{w}_1 = [10\cdots0]^T$, $\mathbf{w}_2 = [010\cdots0]^T$, etc. Initialise d_i $(i = 1, ...k)$ to a small positive value. Then:

1. As each point $\mathbf{x}_{t+1}$ arrives, initialise $\acute{\mathbf{x}}_1 := \mathbf{x}_{t+1}$.

2. For $1 \le i \le k$, we perform the following assignments and updates, in order:

$$y_i := \mathbf{w}_i^T \acute{\mathbf{x}}_i \qquad (y_{t+1,i} = \text{projection onto } \mathbf{w}_i)$$
$$d_i \leftarrow \lambda d_i + y_i^2 \qquad (\text{energy} \propto i\text{-th eigenval. of } \mathbf{X}_t^T\mathbf{X}_t)$$
$$\mathbf{e}_i := \acute{\mathbf{x}}_i - y_i\mathbf{w}_i \qquad (\text{error, } \mathbf{e}_i \perp \mathbf{w}_i)$$
$$\mathbf{w}_i \leftarrow \mathbf{w}_i + \frac{1}{d_i} y_i \mathbf{e}_i \qquad (\text{update PC estimate})$$
$$\acute{\mathbf{x}}_{i+1} := \acute{\mathbf{x}}_i - y_i\mathbf{w}_i \qquad (\text{repeat with remainder of } \mathbf{x}_t).$$

The *forgetting factor* λ will be discussed in Section 4.3 (for now, assume $\lambda = 1$). For each i, $d_i = tE_{t,i}$ and $\acute{\mathbf{x}}_i$ is the component of $\mathbf{x}_{t+1}$ in the orthogonal complement of the space spanned by the updated estimates $\mathbf{w}_{i'}, 1 \le i' < i$ of the participation weights. The vectors $\mathbf{w}_i, 1 \le i \le k$ are in order of importance (more precisely, in order of decreasing eigenvalue or energy). It can be shown that, under stationarity assumptions, these $\mathbf{w}_i$ in these equations converge to the true principal directions.

Complexity. We only need to keep the k weight vectors $\mathbf{w}_i$ $(1 \le i \le k)$, each n-dimensional. Thus the total cost is $O(nk)$, both in terms of time and of space. The update cost does not depend on t. This is a tremendous gain, compared to the usual PCA computation cost of $O(tn^2)$.

4.2 Detecting the number of hidden variables

In practice, we do not know the number k of hidden variables. We propose to estimate k on the fly, so that we maintain a high percentage f_E of the *energy* E_t. Energy thresholding is a common method to determine how many principal components are needed [26]. Formally, the energy E_t (at time t) of the sequence of $\mathbf{x}_t$ is defined as

$$E_t := \frac{1}{t}\sum_{\tau=1}^{t} \|\mathbf{x}_\tau\|^2 = \frac{1}{t}\sum_{\tau=1}^{t}\sum_{i=1}^{n} x_{\tau,i}^2.$$

Similarly, the energy $\tilde{E}_t$ of the reconstruction $\tilde{\mathbf{x}}$ is defined as

$$\tilde{E}_t := \frac{1}{t}\sum_{\tau=1}^{t} \|\tilde{\mathbf{x}}_\tau\|^2.$$

Lemma 4.1. *Assuming the $\mathbf{w}_i, 1 \le i \le k$ are orthonormal, we have*

$$\tilde{E}_t = \frac{1}{t}\sum_{\tau=1}^{t} \|\mathbf{y}_\tau\|^2 = \frac{t-1}{t}\tilde{E}_{t-1} + \frac{1}{t}\|\mathbf{y}_t\|.$$

Proof. If the $\mathbf{w}_i, 1 \le i \le k$ are orthonormal, then it follows easily that $\|\tilde{\mathbf{x}}_\tau\|^2 = \|y_{\tau,1}\mathbf{w}_1 + \cdots + y_{\tau,k}\mathbf{w}_k\|^2 = y_{\tau,1}^2\|\mathbf{w}_1\|^2 + \cdots + y_{\tau,k}^2\|\mathbf{w}_k\|^2 = y_{\tau,1}^2 + \cdots + y_{\tau,k}^2 = \|\mathbf{y}_\tau\|^2$ (Pythagorean theorem and normality). The result follows by summing over τ. □

It can be shown that algorithm TRACKW maintains orthonormality without the need for any extra steps (otherwise, a simple re-orthonormalization step at the end would suffice).

From the user's perspective, we have a low-energy and a high-energy threshold, f_E and F_E, respectively. We keep enough hidden variables k, so the retained energy is within the range $[f_E \cdot E_t, F_E \cdot E_t]$. Whenever we get outside these bounds, we increase or decrease k. In more detail, the steps are:

1. Estimate the full energy E_{t+1}, incrementally, from the sum of squares of $x_{\tau,i}$.
2. Estimate the energy $\tilde{E}_{(k)}$ of the k hidden variables.
3. Possibly, adjust k. We introduce a new hidden variable (update $k \leftarrow k+1$) if the current hidden variables maintain too little energy, i.e., $\tilde{E}_{(k)} < f_E E$. We drop a hidden variable (update $k \leftarrow k-1$), if the maintained energy is too high, i.e., $\tilde{E}_{(k)} > F_E E$.

The energy thresholds f_E and F_E are chosen according to recommendations in the literature [26, 18]. We use a lower energy threshold $f_E = 0.95$ and an upper energy threshold $F_E = 0.98$. Thus, the reconstruction $\tilde{\mathbf{x}}_t$ retains between 95% and 98% of the energy of $\mathbf{x}_t$.

Algorithm SPIRIT

0. Initialise $k \leftarrow 1$ and the total energy estimates of $\mathbf{x}_t$ and $\tilde{\mathbf{x}}_t$ per time tick to $E \leftarrow 0$ and $\tilde{E}_1 \leftarrow 0$. Then,

1. As each new point arrives, update $\mathbf{w}_i$, for $1 \le i \le k$ (step 1, TRACKW).

2. Update the estimates (for $1 \le i \le k$)

$$E \leftarrow \frac{(t-1)E + \|\mathbf{x}_t\|^2}{t} \quad \text{and} \quad \tilde{E}_i \leftarrow \frac{(t-1)\tilde{E}_i + y_{t,i}^2}{t}.$$

3. Let the estimate of retained energy be

$$\tilde{E}_{(k)} := \sum_{i=1}^{k} \tilde{E}_i.$$

If $\tilde{E}_{(k)} < f_E E$, then we start estimating $\mathbf{w}_{k+1}$ (initialising as in step 0 of TRACKW), initialise $\tilde{E}_{k+1} \leftarrow 0$ and increase $k \leftarrow k+1$. If $\tilde{E}_{(k)} > F_E E$, then we discard $\mathbf{w}_k$ and $\tilde{E}_k$ and decrease $k \leftarrow k-1$.

The following lemma proves that the above algorithm guarantees the relative reconstruction error is within the specified interval $[f_E, F_E]$.

Lemma 4.2. *The relative squared error of the reconstruction satisfies*

$$1 - F_E \le \frac{\sum_{\tau=1}^{t} \|\tilde{\mathbf{x}}_\tau - \mathbf{x}_\tau\|^2}{\sum_t \|\mathbf{x}_\tau\|^2} \le 1 - f_E.$$

Proof. From the orthogonality of $\mathbf{x}_\tau$ and the complement $\tilde{\mathbf{x}}_\tau - \mathbf{x}_\tau$ we have $\|\tilde{\mathbf{x}}_\tau - \mathbf{x}_\tau\|^2 = \|\mathbf{x}_\tau\|^2 - \|\tilde{\mathbf{x}}_\tau\|^2 = \|\mathbf{x}_\tau\|^2 - \|\mathbf{y}_\tau\|^2$ (by Lemma 4.1). The result follows by summing over τ and from the definitions of E and $\tilde{E}$. □

Finally, in Section 7.2 we demonstrate that the incremental weight estimates are extremely close to the principal directions computed with offline PCA.

4.3 Exponential forgetting

We can adapt to more recent behaviour by using an exponential forgetting factor $0 < \lambda < 1$. This allows us to follow trend drifts over time. We use the same λ for the estimation of both $\mathbf{w}_i$ as well as the AR models (see Section 5.1). However, we also have to properly keep track of the energy, discounting it with the same rate, i.e., the update at each step is:

$$E \leftarrow \frac{\lambda(t-1)E + \|\mathbf{x}_t\|^2}{t} \quad \text{and} \quad \tilde{E}_i \leftarrow \frac{\lambda(t-1)\tilde{E}_i + y_{t,i}^2}{t}.$$

Typical choices are $0.96 \leq \lambda \leq 0.98$ [24]. As long as the values of $\mathbf{x}_t$ do not vary wildly, the exact value of λ is not crucial. We use $\lambda = 0.96$ throughout. A value of $\lambda = 1$ makes sense when we know that the sequence is stationary (rarely true in practice, as most sequences gradually drift). Note that the value of λ does not affect the computation cost of our method. In this sense, an exponential forgetting factor is more appealing than a sliding window, as the latter has explicit buffering requirements.

5 Putting SPIRIT to work

We show how we can exploit the correlations and hidden variables discovered by SPIRIT to do (a) forecasting, (b) missing value estimation, (c) summarisation of the large number of streams into a small, manageable number of hidden variables, and (d) outlier detection.

5.1 Forecasting and missing values

The hidden variables $\mathbf{y}_t$ give us a much more compact representation of the "raw" variables $\mathbf{x}_t$, with guarantees of high reconstruction accuracy (in terms of relative squared error, which is less than $1-f_E$). When our streams exhibit correlations, as we often expect to be the case, the number k of the hidden variables is much smaller than the number n of streams. Therefore, we can apply *any* forecasting algorithm to the vector of hidden variables $\mathbf{y}_t$, instead of the raw data vector $\mathbf{x}_t$. This reduces the time and space complexity by orders of magnitude, because typical forecasting methods are quadratic or worse on the number of variables.

In particular, we fit the forecasting model on the $\mathbf{y}_t$ instead of $\mathbf{x}_t$. The model provides an estimate $\hat{\mathbf{y}}_{t+1} = f(\mathbf{y}_t)$ and we can use this to get an estimate for

$$\hat{\mathbf{x}}_{t+1} := \hat{y}_{t+1,1}\mathbf{w}_1[t] + \cdots + \hat{y}_{t+1,1}\mathbf{w}_k[t],$$

using the weight estimates $\mathbf{w}_i[t]$ from the previous time tick t. We chose auto-regression for its intuitiveness and simplicity, but any online method can be used.

Correlations. Since the principal directions are orthogonal ($\mathbf{w}_i \perp \mathbf{w}_j, i \neq j$), the components of $\mathbf{y}_t$ are *by construction uncorrelated*—the correlations have already been captured by the $\mathbf{w}_i, 1 \leq i \leq k$. We can take advantage of this de-correlation reduce forecasting complexity. In particular for auto-regression, we found that one AR model per hidden variable provides results comparable to multivariate AR.

Auto-regression. Space complexity for multivariate AR (e.g., MUSCLES [39]) is $O(n^3\ell^2)$, where ℓ is the auto-regression window length. For AR per stream (ignoring correlations), it is $O(n\ell^2)$. However, for SPIRIT, we need $O(kn)$ space for the $\mathbf{w}_i$ and, with one AR model per y_i, the total space complexity is $O(kn + k\ell^2)$. As published, MUSCLES requires space that grows cubically with respect to the number of streams n. We believe it can be made to work with quadratic space, but this is still prohibitive. Both AR per stream and SPIRIT require space that grows linearly with respect to n, but in SPIRIT k is typically very small ($k \ll n$) and, in practice, SPIRIT requires less memory and time per update than AR per stream. More importantly, a single, independent AR model per stream cannot capture *any* correlations, whereas SPIRIT indirectly exploits the correlations present *within* a time tick.

Missing values. When we have a forecasting model, we can use the forecast based on $\mathbf{x}_{t-1}$ to estimate missing values in $\mathbf{x}_t$. We then use these estimated missing values to update the weight estimates, as well as the forecasting models. Forecast-based estimation of missing values is the most time-efficient choice and gives very good results.

5.2 Interpretation

At *any* given time t, SPIRIT readily provides two key pieces of information (aside from the forecasts, etc.):

- The number of hidden variables k.
- The weights $w_{i,j}$, $1 \leq i \leq k$, $1 \leq j \leq n$. Intuitively, the magnitude $|w_{i,j}|$ of each weight tells us how much the i-th hidden variable contributes to the reconstruction of the j-th stream.

In the chlorine example during phase 1 (see Figure 1), the dataset has only one hidden variable, because one sinusoidal-like pattern can reconstruct both streams (albeit with different weights for each). Thus, SPIRIT correctly identifies correlated streams. When the correlation was broken, SPIRIT introduces enough hidden variables to capture that. Finally, it also spots that, in phase 3, normal operation is reestablished and thus disposes of the unnecessary hidden variable. In Section 6 we show additional examples of how we can intuitively interpret this information.

6 Experimental case studies

In this section we present case studies on real and realistic datasets to demonstrate the effectiveness of our approach in discovering the underlying correlations among streams. In particular, we show that:

- We capture the appropriate number of hidden variables. As the streams evolve, we capture these

Dataset	n	k	Description
`Chlorine`	166	2	Chlorine concentrations from EPANET.
`Critter`	8	1–2	Temperature sensor measurements.
`Motes`	54	2–4	Light sensor measurements.

Table 2: *Description of datasets.*

changes in real-time [34] and adapt the number of hidden variables k and the weights $\mathbf{w}_i$.

- We capture the essential behaviour with very few hidden variables and small reconstruction error.
- We successfully deal with missing values.
- We can use the discovered correlations to perform good forecasting, with *much* fewer resources.
- We can easily spot outliers.
- Processing time per stream is constant.

Section 7 elaborates on performance and accuracy.

6.1 Chlorine concentrations

Description. The `Chlorine` dataset was generated by EPANET 2.0[3] that accurately simulates the hydraulic and chemical phenomena within drinking water distribution systems. Given a network as the input, EPANET tracks the flow of water in each pipe, the pressure at each node, the height of water in each tank, and the concentration of a chemical species throughout the network, during a simulation period comprised of multiple timestamps. We monitor the chlorine concentration level at all the 166 junctions in the network shown in Figure 3(a), for 4310 timestamps during 15 days (one time tick every five minutes). The data was generated by using the input network with the demand patterns, pressures, flows specified at each node.

Data characteristics. The two key features are:

- A clear global periodic pattern (daily cycle, dominating residential demand pattern). Chlorine concentrations reflect this, with few exceptions.
- A slight time shift across different junctions, which is due to the time it takes for fresh water to flow down the pipes from the reservoirs.

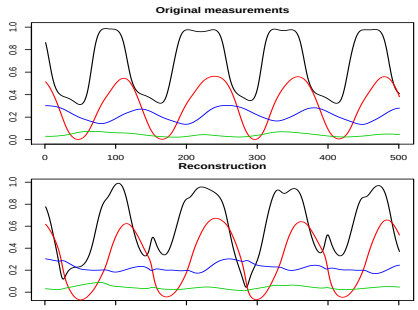
Thus, most streams exhibit the same sinusoidal-like pattern, except with gradual phase shifts as we go further away from the reservoir.

Results of SPIRIT. SPIRIT can successfully summarise the data using just two numbers (hidden variables) per time tick, as opposed to the original 166 numbers. Figure 3(a) shows the reconstruction for four of the sensors (out of 166). Only two hidden variables give very good reconstruction.

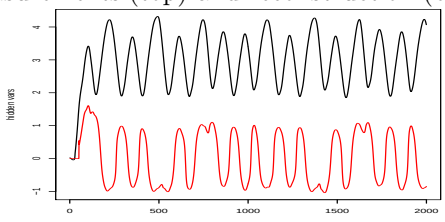
Interpretation. The two hidden variables (Figure 3(b)) reflect the two key dataset characteristics:

- The first hidden variable captures the global, periodic pattern.

[3] `http://www.epa.gov/ORD/NRMRL/wswrd/epanet.html`



(a) Measurements (top) and reconstruction (bottom).



(b) Hidden variables.

Figure 3: *`Chlorine` dataset: (a) Actual measurements and SPIRIT's reconstruction at four junctions (500 consecutive timestamps; the patterns repeat after that). (b) SPIRIT's hidden variables.*

- The second one also follows a very similar periodic pattern, but with a slight "phase shift." It turns out that the two hidden variables together are sufficient to express (via a linear combination) any other time series with an arbitrary "phase shift."

6.2 Light measurements

Description. The `Motes` dataset consists of light intensity measurements collected using Berkeley Mote sensors, at several different locations in a lab (see Figure 4), over a period of a month.

Data characteristics. The main characteristics are:

- A clear global periodic pattern (daily cycle).
- Occasional big spikes from some sensors (outliers).

Results of SPIRIT. SPIRIT detects four hidden variables (see Figure 5). Two of these are intermittent and correspond to outliers, or changes in the correlated trends. We show the reconstructions for some of the observed variables in Figure 4(b).

Interpretation. In summary, the first two hidden variables (see Figure 5) correspond to the global trend and the last two, which are intermittently present, correspond to outliers. In particular:

- The first hidden variable captures the global periodic pattern.
- The interpretation of the second one is again similar to the `Chlorine` dataset. The first two hidden variables together are sufficient to express arbitrary phase shifts.
- The third and fourth hidden variables indicate some of the potential outliers in the data. For example, there is a big spike in the 4th hidden

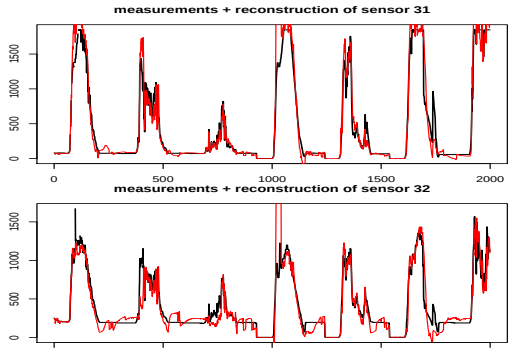


Figure 4: *`Mote` dataset: Measurements (bold) and reconstruction (thin) on node 31 and 32.*

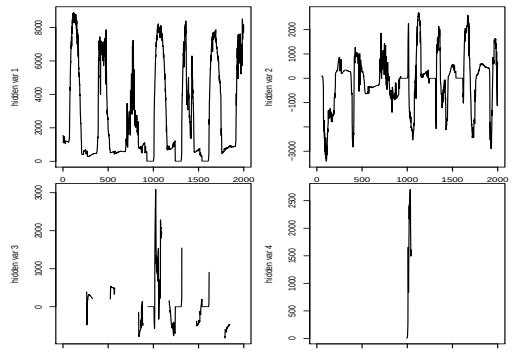


Figure 5: *`Mote` dataset, hidden variables: The third and fourth hidden variables are intermittent and indicate "anomalous behaviour." Note that the axes limits are different in each plot.*

variable at time $t = 1033$, as shown in Figure 5. Examining the participation weights $\mathbf{w}_4$ at that timestamp, we can find the corresponding sensors "responsible" for this anomaly, i.e., those sensors whose participation weights have very high magnitude. Among these, the most prominent are sensors 31 and 32. Looking at the actual measurements from these sensors, we see that before time $t = 1033$ they are almost 0. Then, very large increases occur around $t = 1033$, which bring an additional hidden variable into the system.

6.3 Room temperatures

Description. The `Critter` dataset consists of 8 streams (see Figure 9). Each stream comes from a small sensor[4] (aka. Critter) that connects to the joystick port and measures temperature. The sensors were placed in 5 neighbouring rooms. Each time tick represents the average temperature during one minute.

Furthermore, to demonstrate how the correlations capture information about missing values, we repeated the experiment after blanking 1.5% of the values (five blocks of *consecutive* timestamps; see Figure 7).

Data characteristics. Overall, the dataset does not seem to exhibit a clear trend. Upon closer examination, all sensors fluctuate slightly around a constant temperature (which ranges from 22–27^{o}C, or 72–81^{o}F, depending on the sensor). Approximately half of the sensors exhibit a more similar "fluctuation pattern."

[4] `http://www.ices.cmu.edu/sensornets/`

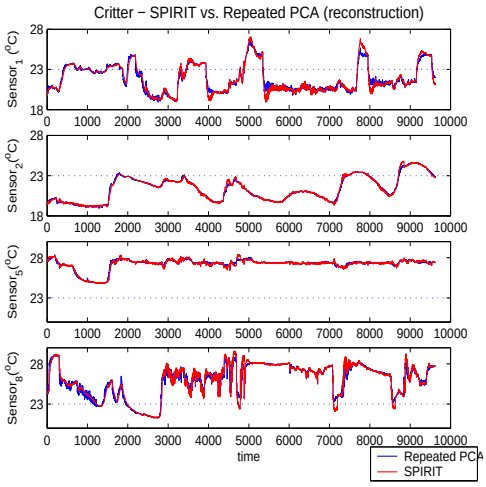


Figure 6: *Reconstructions $\tilde{\mathbf{x}}_t$ for `Critter`. Repeated PCA requires (i) storing the entire data and (ii) performing PCA at each time tick (quadratic time, at best—for example, wall clock times here are 1.5 minutes versus 7 seconds).*

Results of SPIRIT. SPIRIT discovers one hidden variable, which is sufficient to capture the general behaviour. However, if we utilise prior knowledge (such as, e.g., that the pre-set temperature was 23^{o}C), we can ask SPIRIT to detect trends with respect to that. In that case, SPIRIT comes up with two hidden variables, which we explain later.

SPIRIT is also able to deal successfully with missing values in the streams. Figure 7 shows the results on the blanked version (1.5% of the total values in five blocks of *consecutive* timestamps, starting at a different position for each stream) of `Critter`. The correlations captured by SPIRIT's hidden variable often provide useful information about the missing values. In particular, on sensor 8 (second row, Figure 7), the correlations picked by the *single* hidden variable successfully capture the missing values in that region (consisting of 270 ticks). On sensor 7, (first row, Figure 7; 300 blanked values), the upward trend in the blanked region is also picked up by the correlations. Even though the trend is slightly mis-estimated, as soon as the values are observed again, SPIRIT very quickly gets back to near-perfect tracking.

Interpretation. If we examine the participation weights in $\mathbf{w}_1$, the largest ones correspond primarily to streams 5 and 6, and then to stream 8. If we examine the data, sensors 5 and 6 consistently have the highest temperatures, while sensor 8 also has a similar temperature most of the time.

However, if the sensors are calibrated based on the fact that these are building temperature measure-

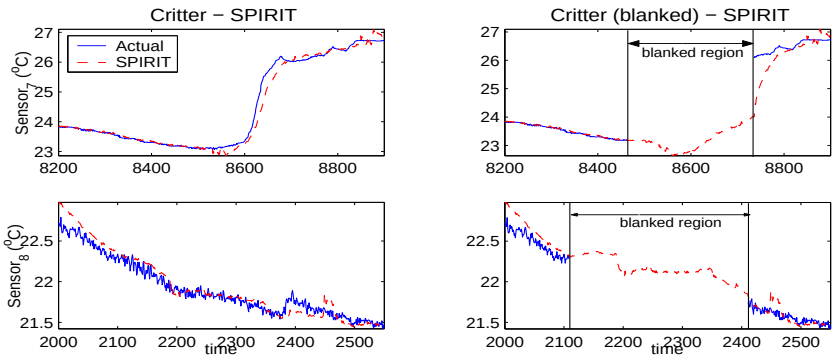


Figure 7: *Detail of the forecasts on* `Critter` *with blanked values. The second row shows that the correlations picked by the* single *hidden variable successfully capture the missing values in that region (consisting of 270* consecutive *ticks). In the first row (300 consecutive blanked values), the upward trend in the blanked region is also picked up by the correlations to other streams. Even though the trend is slightly mis-estimated, as soon as the values are observed again SPIRIT quickly gets back to near-perfect tracking.*

ments, where we have set the thermostat to 23^{o}C (73^{o}F), then SPIRIT discovers two hidden variables (see Figure 9). More specifically, if we reasonably assume that we have the prior knowledge of what the temperature *should be* (note that this has nothing to do with the average temperature in the observed data) and want to discover what happens around that temperature, we can subtract it from each observation and SPIRIT will discover patterns and anomalies based on this information. Actually, this is what a human operator would be interested in discovering: "Does the system work as I expect it to?" (based on my knowledge of how it should behave) and "If not, what is wrong?" So, in this case, we indeed discover this information.

- The interpretation of the first hidden variable is similar to that of the original signal: sensors 5 and 6 (and, to a lesser extent, 8) deviate from that temperature the most, for most of the time. Maybe the thermostats are broken or set wrong?
- For $\mathbf{w}_2$, the largest weights correspond to sensors 1 and 3, then to 2 and 4. If we examine the data, we notice that these streams follow a similar, fluctuating trend (close to the pre-set temperature), the first two varying more violently. The second hidden variable is added at time $t = 2016$. If we examine the plots, we see that, at the beginning, most streams exhibit a slow dip and then ascent (e.g., see 2, 4 and 5 and, to a lesser extent, 3, 7 and 8). However, a number of them start fluctuating more quickly and violently when the second hidden variable is added.

7 Performance and accuracy

In this section we discuss performance issues. First, we show that SPIRIT requires very limited space and time. Next, we elaborate on the accuracy of SPIRIT's incremental estimates.

7.1 Time and space requirements

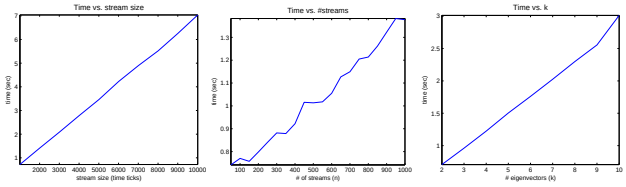
Figure 8 shows that SPIRIT scales linearly with respect to number of streams n and number of hidden variables k. AR per stream and MUSCLES are essentially off the charts from the very beginning. Furthermore, SPIRIT scales linearly with stream size (i.e., requires constant processing time per tuple).

The plots were generated using a synthetic dataset that allows us to precisely control each variable. The datasets were generated as follows:

- Pick the number k of trends and generate sine waves with different frequencies, say $y_{t,i} = \sin(2\pi i/kt), 1 \leq i \leq k$. Thus, all trends are pairwise linearly independent.
- Generate each of the n streams as random linear combinations of these k trend signals.

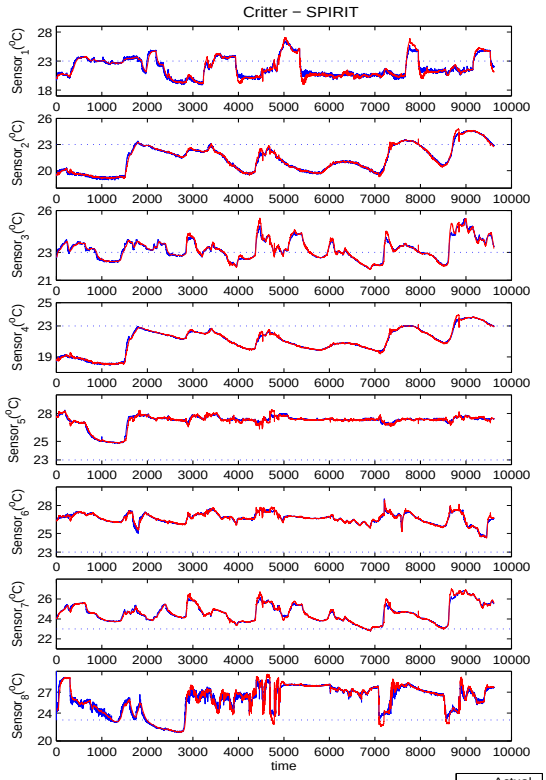
This allows us to vary k, n and the length of the streams at will. For each experiment shown, one of these parameters is varied and the other two are held fixed. The numbers in Figure 8 are wall-clock times of our Matlab implementation. Both AR-per-stream as well as MUSCLES (also in Matlab) are several orders of magnitude slower and thus omitted from the charts.

It is worth mentioning that we have also implemented the SPIRIT algorithms in a real system [34], which can obtain measurements from sensor devices and display hidden variables and trends in real-time.

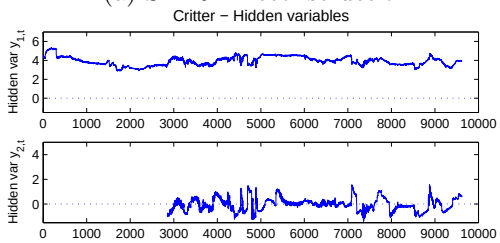


(a) stream size t (b) streams n (c) hidden variables k

Figure 8: *Wall-clock times (including time to update forecasting models). Times for AR and MUSCLES are not shown, since they are off the charts from the start (13.2 seconds in (a) and 209 in (b)). The starting values are: (a) 1000 time ticks, (b) 50 streams, and (c) 2 hidden variables (the other two held constant for each graph). It is clear that SPIRIT scales linearly.*



(a) SPIRIT reconstruction



(b) Hidden variables

Figure 9: *Actual* `Critter` *data and SPIRIT output (a), for each of the temperature sensors. The experiment shows that with only two hidden variable, SPIRIT can track the overall behaviour of the entire stream collection. (b) shows the hidden variables.*

7.2 Accuracy

In terms of accuracy, everything boils down to the quality of the summary provided by the hidden variables. To this end, we show the reconstruction $\tilde{\mathbf{x}}_t$ of $\mathbf{x}_t$, from the hidden variables $\mathbf{y}_t$ in Figure 6. One line uses the true principal directions, the other the SPIRIT estimates (i.e., weight vectors). SPIRIT comes very close to repeated PCA.

We should note that this is an unfair comparison for SPIRIT, since repeated PCA requires (i) storing *all* stream values, and (ii) performing a very expensive SVD computation for *each* time tick. However, the tracking is still very good. This is always the case, provided the corresponding eigenvalue is large enough and fairly well-separated from the others. If the eigenvalue is small, then the corresponding hidden variable is of no importance and we do not track it anyway.

Dataset	Chlorine	Critter	Motes
MSE rate (SPIRIT)	0.0359	0.0827	0.0669
MSE rate (repeated PCA)	0.0401	0.0822	0.0448

Table 3: *Reconstruction accuracy (mean squared error rate).*

Reconstruction error. Table 3 shows the reconstruction error, $\sum \|\tilde{\mathbf{x}}_t - \mathbf{x}_t\|^2 / \sum \|\mathbf{x}_t\|^2$, achieved by SPIRIT. In every experiment, we set the energy thresholds to $[f_E, F_E] = [0.95, 0.98]$. Also, as pointed out before, we set $\lambda = 0.96$ as a reasonable default value to deal with non-stationarities that may be present in the data, according to recommendations in the literature [24]. Since we want a metric of overall quality, the MSE rate weighs each observation equally and does not take into account the forgetting factor λ.

Still, the MSE rate is very close to the bounds we set. In Table 3 we also show the MSE rate achieved by repeated PCA. As pointed out before, this is already an unfair comparison. In this case, we set the number of principal components k to the maximum that SPIRIT uses at any point in time. This choice favours repeated PCA even further. Despite this, the reconstruction errors of SPIRIT are close to the ideal, while using orders of magnitude less time and space.

8 Conclusion

We focus on finding patterns, correlations and hidden variables, in a large number of streams. Our proposed method has the following desirable characteristics:

- It discovers underlying correlations among multiple streams, incrementally and in real-time [34] and provides a very compact representation of the stream collection, via a few *hidden variables*.
- It automatically estimates the number k of hidden variables to track, and it can automatically adapt, if k changes (e.g., an air-conditioner switching on, in a temperature sensor scenario).
- It scales up extremely well, both on database size (i.e., number of time ticks t), and on the number n of streams. Therefore it is suitable for a large number of sensors / data sources.
- Its computation demands are low: it only needs $O(nk)$ floating point operations—no matrix inversions nor SVD (both infeasible in online, any-time settings). Its space demands are similarly limited.
- It can naturally hook up with any forecasting method, and thus easily do prediction, as well as handle missing values.

We showed that the output of SPIRIT has a natural interpretation. We evaluated our method on several datasets, where indeed it discovered the hidden variables. Moreover, SPIRIT-based forecasting was several times faster than other methods.

Acknowledgments. We wish to thank Michael Bigrigg for providing the temperature sensor data.

References

[1] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.

[2] C. C. Aggarwal. A framework for diagnosing changes in evolving data streams. In *SIGMOD*, 2003.

[3] C. C. Aggarwal, J. Han, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB*, 2003.

[4] M. H. Ali, M. F. Mokbel, W. Aref, and I. Kamel. Detection and tracking of discrete phenomena in sensor network databases. In *SSDBM*, 2005.

[5] A. Arasu, B. Babcock, S. Babu, J. McAlister, and J. Widom. Characterizing memory requirements for queries over continuous data streams. In *PODS*, 2002.

[6] B. Babcock, S. Babu, M. Datar, and R. Motwani. Chain : Operator scheduling for memory minimization in data stream systems. In *SIGMOD*, 2003.

[7] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *SODA*, 2002.

[8] D. Carney, U. Cetintemel, A. Rasin, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Operator scheduling in a data stream manager. In *VLDB*, 2003.

[9] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[10] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). In *VLDB*, 2002.

[11] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: a stream database for network applications. In *SIGMOD*, 2003.

[12] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *SIGMOD*, 2003.

[13] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *SODA*, 2002.

[14] A. Deshpande, C. Guestrin, S. Madden, and W. Hong. Exploiting correlated attributes in acqusitional query processing. In *ICDE*, 2005.

[15] K. I. Diamantaras and S.-Y. Kung. *Principal Component Neural Networks: Theory and Applications*. John Wiley, 1996.

[16] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In *SIGMOD*, 2002.

[17] P. Domingos and G. Hulten. Mining high-speed data streams. In *KDD*, 2000.

[18] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1990.

[19] S. Ganguly, M. Garofalakis, and R. Rastogi. Processing set expressions over continuous update streams. In *SIGMOD*, 2003.

[20] V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining data streams under block evolution. *SIGKDD Explorations*, 3(2):1–10, 2002.

[21] S. Guha, D. Gunopulos, and N. Koudas. Correlating synchronous and asynchronous data streams. In *KDD*, 2003.

[22] S. Guha, C. Kim, and K. Shim. XWAVE: Optimal and approximate extended wavelets for streaming data. In *VLDB*, 2004.

[23] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE TKDE*, 15(3):515–528, 2003.

[24] S. Haykin. *Adaptive Filter Theory*. Prentice Hall, 1992.

[25] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *KDD*, 2001.

[26] I. T. Jolliffe. *Principal Component Analysis*. Springer, 2002.

[27] E. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In *KDD*, 2004.

[28] J. Lin, M. Vlachos, E. Keogh, and D. Gunopulos. Iterative incremental clustering of time series. In *EDBT*, 2004.

[29] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *CIDR*, 2003.

[30] E. Oja. Neural networks, principal components, and subspaces. *Intl. J. Neural Syst.*, 1:61–68, 1989.

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

[32] S. Papadimitriou, A. Brockwell, and C. Faloutsos. Adaptive, hands-off stream mining. In *VLDB*, 2003.

[33] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. BRAID: Stream mining through group lag correlations. In *SIGMOD*, 2005.

[34] J. Sun, S. Papadimitriou, and C. Faloutsos. Online latent variable detection in sensor networks. In *ICDE*, 2005. (demo).

[35] N. Tatbul, U. Cetintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *VLDB*, 2003.

[36] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proc.ACM SIGKDD*, 2003.

[37] B. Yang. Projection approximation subspace tracking. *IEEE Trans. Sig. Proc.*, 43(1):95–107, 1995.

[38] Y. Yao and J. Gehrke. Query processing in sensor networks. In *CIDR*, 2003.

[39] B.-K. Yi, N. Sidiropoulos, T. Johnson, H. Jagadish, C. Faloutsos, and A. Biliris. Online data mining for co-evolving time sequences. In *ICDE*, 2000.

[40] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *SIGMOD*, 1996.

[41] Y. Zhu and D. Shasha. StatStream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, 2002.

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