Parsimonious Linear Fingerprinting for Time Series

Lei Li
School of Computer Science
Carnegie Mellon University
leili@cs.cmu.edu

B. Aditya Prakash
School of Computer Science
Carnegie Mellon University
badityap@cs.cmu.edu

Christos Faloutsos
School of Computer Science
Carnegie Mellon University
christos@cs.cmu.edu

ABSTRACT

We study the problem of mining and summarizing multiple time series effectively and efficiently. We propose PLiF, a novel method to discover essential characteristics ("fingerprints"), by exploiting the joint dynamics in numerical sequences. Our fingerprinting method has the following benefits: (a) it leads to *interpretable* features; (b) it is *versatile*: PLiF enables numerous mining tasks, including clustering, compression, visualization, forecasting, and segmentation, matching top competitors in each task; and (c) it is fast and *scalable*, with linear complexity on the length of the sequences.

We did experiments on both synthetic and real datasets, including human motion capture data (17MB of human motions), sensor data (166 sensors), and network router traffic data (18 million raw updates over 2 years). Despite its generality, PLiF outperforms the top clustering methods on clustering; the top compression methods on compression (3 times better reconstruction error, for the same compression ratio); it gives meaningful visualization and at the same time, enjoys a *linear* scale-up.

1. INTRODUCTION

Time sequences appear in countless applications, like sensor measurements [13], mobile object tracking [20], data center monitoring [27], motion capture sequences [19, 21], environmental monitoring (like chlorine levels in drinking water [25])and many more. Given multiple, interacting time sequences, how can we group them according to similarity? How can we find compact, numerical features ("*fingerprints*"), to describe and distinguish each of them?

Researches in time sequences form two broad classes:

(a) Feature extraction (and similarity search, indexing etc), using, say, Fourier or wavelet coefficients, piece-wise linear approximations and similar methods and
(b) forecasting, like an autoregressive integrated moving average model (ARIMA) and related methods.

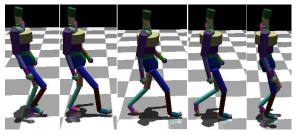
The former class is useful for querying: indexing, similarity searching, clustering. The latter class is useful for mining: forecasting, missing value imputation, anomaly detection. Can we develop a method that has the best of both worlds? Extracting the essence of time sequences is already very useful - it would be even more useful if those features are easy to interpret, and even better if they could help us do forecasting. Ability to forecast automatically leads to anomaly detection (every time-tick that deviates too much from our forecast), segmentation (a time interval deviating too much from our forecast), compression (storing the deltas from the forecasts), and missing value imputation, extrapolation and interpolation. And of course, we would like the method to be scalable, with linear complexity on the length of the sequences. Is it possible to achieve as many as possible of the above goals, any of which alone is already very useful? The proposed PLiF method gives a positive answer: the idea is to extract the essential numerical representation that characterizes the evolving dynamics of the sequences, specifically, to fit a *Linear Dynamical System* (LDS) on the collection of m sequences, and then we show how to extract a few, but meaningful features out of the LDS. We will refer to those features as the "*fingerprints*" of each sequence. We further show that the proposed *fingerprints* achieve all the above goals:

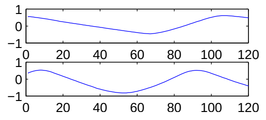
1. *Effectiveness:* the resulting features lead to a natural distance function, which agrees with human intuition and the provided ground truth. Thus, fingerprints lead to good clustering as well as visualization (see Fig. 1(d) and Fig. 5);
2. *Interpretability*: as we will show, the fingerprints correspond to groups of harmonics;
3. *Forecasting:* they naturally lead to forecasting and compression;
4. *Scalability:* the proposed PLiF method is fast and scalable on the size of the sequences.

Table 1 compares PLiF against traditional methods and illustrates their strengths and shortcomings: a checkmark (✓) indicates that the corresponding method (column) fulfills the corresponding requirement (row). Only PLiF has all entries as checkmarks. In more detail (also see Appendix B), the desirable fingerprints should allow for lags, and small variations in frequency. While,

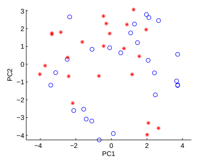
- Fourier analysis and wavelet methods could identify the frequencies in a *single* signal, but can not cross-correlate similar signals, nor do forecasting[1].

[1]One might argue that Fourier coefficients can do rudimen-

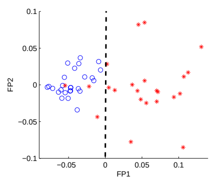

(a)



(b) z-value of right foot marker



(c) PCA



(d) fingerprints: FP1 vs FP2

Figure 1: Motion capture (mocap) sequences: sample data and visualization. 1(a): a film-strip of a human motion. 1(b): right foot position for a walking motion (top), and a running one (bottom). 1(c): Scatter-plot of two principal components for walking (blue circles) and running sequences (red stars), without clear separation. 1(d): Scatter-plot of the "fingerprints" (FP) by PLiF - first FP versus second FP, for all 49 mocap sequences. Notice the near-perfect separation of the walking motions (blue circles), from the running ones (red stars).

- Singular value decomposition (SVD) and its "centered" version, principal component analysis (PCA), do capture correlations (by doing soft clustering of similar sequences) and thus derive hidden ("latent") variables, but they can not do forecasting either, nor is it easy to interpret the derived hidden variables.
- Standard Linear Dynamical Systems (LDS) and Kalman filters can capture correlations, as well as do forecasting. However, the resulting hidden variables are hard to interpret (see Fig. 2(c)), and they do not lead to a natural distance function.

Finally, we do not show typical distance functions for time series clustering in Table 1: the Euclidean distance (sum of squares of differences at each time-tick) and the Dynamic Time Warping (DTW) distance. The reason is that none of them leads to forecasting, nor to feature extraction, and thus the interpretability requirement is out of reach. Moreover, the typical, un-constrained, version of DTW fails the scalability requirement, being quadratic on the length of the sequences.

To make the discussion more concrete, we refer to one of our motivating applications, motion capture (*mocap*) sequences: For each motion, we have 93 numbers (positions or angles of markers on the joints of the actor), with a variable number of frames (=time-ticks) for each such sequence, typically 60-120 frames per second. See Fig. 1(b) for two such example sequences, both plotting the right-foot z-value as a function of time, for one of the walking

Table 1: Capabilities of Approaches. Only PLiF meets all specs[3].

	SVD/ PCA	DFT/ DWT	LDS	PLiF
Correlation Discovery	✓		✓	✓
Interpretability		✓		✓
Forecasting			✓	✓

motions, and one of the running ones, from the publicly available `mocap.cs.cmu.edu` repository of mocap sequences.

Given a large collection of such human motion sequences, we want to find clusters and to group similar motions together. The desirable features/fingerprints would have the following properties:

- P1: Lag independence: two walking motions should be grouped together even if they start at different footstep or phase;
- P2: Frequency proximity: running motions with nearby speed of motion should be grouped together;
- P3: Harmonics grouping: Several sensor measurements, like human motion, human voice, automobile traffic, obey several periodicities, often with related frequencies ("harmonics"). We would like to detect such groups of harmonics.

Fig. 1(d) gives a quick preview of the visualization and effectiveness of the proposed PLiF method: For the 49 sequences we have, we map each to its two fingerprint values, thus making it a 2-d point. Those 49 points are shown in Fig. 1(d), using 'stars' for motions that were (manually) labeled as running motions, and circles for the walking motions. Notice how clearly the two groups can be separated by a vertical line at x=0.

The rest of the paper is organized in the typical way: In the upcoming sections, we give the problem definition and a running example, then we list the shortcomings of earlier methods, the description of PLiF, experiments, related work and conclusions.

2. PROBLEM DEFINITION & RUNNING EXAMPLE

For the sake of exposition, we provide an arithmetic example here to demonstrate our proposed PLiF method. As mentioned in the introduction, the problem is as follows:

PROBLEM 1 (TIME SEQUENCE FINGERPRINTING). **Given** *m time sequences of length T,* **Extract** *features that match the four goals in the introduction.*

The four goals are that (a) the features should be effective, capturing the essence of what humans consider in similar sequences; (b) interpretable (c) they should lead to forecasting and (d) their computation should be fast and scalable.

More specifically, for the first goal of effectiveness, we want the fingerprints to have properties P1-P3, namely, *lag independence*, *frequency proximity* and *harmonics grouping*.

Thus, we use the following illustrative sequences (see Figure 2(a)), of length T=500 time-ticks, as defined in Table 2.

tary forecasting, since they can generate values for time ticks outside the initial time range $(1, \ldots, T)$; however, these values will just lead to repeating the initial signal, with ringing phenomena if the signal has a trend.

[3]Scalability has not been shown as all methods here have computation time linear to the length of data sequences.

Table 2: Illustrative sequences

Equation	Comment
(a) $X_1 = \sin(2\pi t/100)$	period 100
(b) $X_2 = \cos(2\pi t/100)$	time-shifted of (a)
(c) $X_3 = \sin(2\pi t/100) + \cos(2\pi t/100)$	time-shifted & higher amplitude
(d) $X_4 = \sin(2\pi t/110) + 0.2\sin(2\pi t/30)$	mixture of two waves of periods 110 and 30
(e) $X_5 = \cos(2\pi t/110) + 0.2\sin(2\pi t/30 + \pi/4)$	same as (d) but lag in both components

The first three sequences have period 100, with differing lags and amplitudes and thus we would like them to fall into the same group. The remaining two combine two frequencies (with periods 110 and 30), and a small phase difference; according to the P1-P3 properties, we would expect them to form another group of their own.

Fig. 2(a) also shows the first sequence, in dashed line form, so that we can visually compare the five sequences. As mentioned in the introduction, and as we elaborate in Appendix B, the typical method for dimensionality reduction (and thus feature extraction) is PCA/ SVD; and the typical method for forecasting is autoregression and its more general form, Linear Dynamical Systems (LDS, where we specifically use the output matrix). We show the resulting features for each method as gray-scale heat-maps (Fig. 2(b)-2(d)), where rows are sequences, columns are features (= fingerprints), and black color indicates high values. Rows that are visually similar means that they have similar feature values and thus would end up in the same cluster.

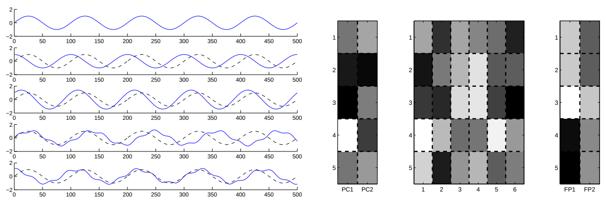
Interpreting the heat-maps or "Why not PCA or LDS?". In short, only PLiF gives effective features. Specifically, notice that PCA (Fig. 2(b)) yields similar rows for sequence (a) and (e) - they are indeed visually similar in their time-plots, too, with small Euclidean distance (sum of square of daily differences). This is not surprising, because PCA and SVD actually preserve the Euclidean distance as best as possible - except that the Euclidean distance fails our desirable goals, heavily penalizing lags. Similarly using the *output matrix* **C** from LDS (Fig. 2(c)) gives a poor grouping.

The heat-maps above explain why both PCA/SVD, as well as LDS, will lead to *poor clustering*, after we apply, say, k-means [11]. In contrast, the heat-map of our proposed method PLiF (Fig. 2(d)) gives the expected groupings: the last two sequences are clearly together, with dark color in their first column; and the first three sequences have dark color in their second column.

As we show in later sections, PLiF also gives the interpretation for each feature: the corresponding "harmonic" groups and the strength of them in each of the sequences (Fig. 3(c)). Although the above is synthetic data, such lag correlation and frequency combinations are common in time series data such as motion capture data and sensor data.

3. PROPOSED METHOD: PLiF

We describe our proposed method PLiF (Parsimonious Linear Fingerprinting) in this section. First we give the basic variation (PLiF-naive) and explain its steps and in Appendix C.3 we show how to make it even faster. Table 3 gives an overview of the symbols used and their definitions.



(a) Sample Signals (b) PCA (c) LDS (d) PLiF

Figure 2: Running example: 5 synthetic sequences (top 3, with period 100 and possibly shifts; rest 2, with periods 110 and 30. 2(a): the time plots. 2(b),2(c),2(d): 'heat-maps' of fingerprints for each sequence, using PCA, LDS and PLiF, respectively. PLiF gives similar fingerprints for the top 3 and the bottom 2 sequences respectively, while competitors do not.

Table 3: Symbols

m	number of sequences
T	duration (length) of sequences
h	number of hidden variables for each time tick
h'	number of harmonics excluding conjugates
$\mathbf{X}$	data matrix, $m \times \mathrm{T}$
$\vec{z}_n$	hidden variables for time n, $h \times 1$ vector
$\vec{\mu}_0$	initial state for hidden variable, $h \times 1$ vector
$\mathbf{A}$	transition matrix (like "Newton dynamics"), $h \times h$
$\mathbf{C}$	output matrix ("hidden to observation") $m \times h$
$\mathbf{V}$	compensation matrix, eigenvectors of $\mathbf{A}$, $h \times h$
$\mathbf{\Lambda}$	eigen-dynamics matrix (eigenvalues of $\mathbf{A}$), $h \times h$
$\mathbf{C}_h$	harmonic mixing matrix, $m \times h$
$\mathbf{C}_m$	harmonic magnitude matrix, $m \times h'$
$\mathbf{F}$	fingerprints

Goal. To recap, we want to solve Problem 1, *Given* multiple time sequences of same duration T, *find* features which have the four properties namely: (a) Effective (can be used for visualization and clustering); (b) Meaningful; (c) Generative (can be used for forecasting and compression); and (d) Scalable.

Each following subsection describes a step in our algorithm. At the high level, PLiF (a) discovers the "Newton"-like dynamics, using a modified, faster way of learning an LDS; (b) normalizes the resulting *transition matrix* **A**, which reveals the natural frequencies and exponential decays / explosions of the given set of sequences (which we refer to as *harmonics*, see Definition 1); and (c) groups some of those harmonics/hidden variables, after ignoring the phase, thus accounting for lag-correlations. The discovered groups of such frequencies are exactly the "fingerprints" (features) that PLiF is using for clustering, visualization, compression, etc.

3.1 Learning Dynamics

Intuition. To understand the hidden pattern in the multiple signals, we want to extract the hidden dynamics, similar to "Newtonian" dynamics like velocities and accelerations. Our basic intuition is to assume that there is a series of hidden variables, representing the states of the hidden pattern, which are evolving according to a linear transformation and are linearly transformed to the observed numerical sequences.

Theory. To obtain the dynamics in data, we use an underlying Linear Dynamical System (LDS) to model multiple time series (Eq. 11 and 12). We use the EM algorithm (described in Appendix B.3) for learning the model parameters.

$$\vec{z}_1 = \vec{\mu}_0 + noise \qquad \vec{z}_{n+1} = \mathbf{A}\vec{z}_n + noise \qquad \vec{x}_n = \mathbf{C}\vec{z}_n + noise$$

$(n = 1, \ldots, T)$.

The LDS model includes parameters of an initial state vector $\vec{\mu}_0$, a *transition matrix* $\mathbf{A}$ and an *output matrix* $\mathbf{C}$ (along with the noise covariance matrices). Similar to "Newtonian" dynamics, the transition matrix $\mathbf{A}$ will predict the hidden variables (like the velocity and acceleration in human motions) for the next time tick, and the output matrix $\mathbf{C}$ will tell us how the hidden variables (e.g. velocities and accelerations) map to the observed values (e.g. positions) at each time tick (each row of $\mathbf{C}$ corresponds to one sequence).

Note that as discussed before, the transition matrix $\mathbf{A}$ is not unique: it is subject to permutation, rotation and linear combinations, and so is the *output matrix* $\mathbf{C}$. Thus each row in $\mathbf{C}$ can not uniquely identify the characteristics of the corresponding series. Our subsequent steps are motivated by this observation.

Example. On using $h = 6$ hidden variables to learn the parameters for the 5 synthetic sequences shown in Figure 2(a), we will get a 6×6 transition matrix $\mathbf{A}$, a 5×6 output matrix $\mathbf{C}$ and a 6×1 initial state vector $\vec{\mu}_0$. Figure 2(c) shows the $\mathbf{C}$ matrix. Clearly, it is all jumbled up and doesn't show any clear pattern w.r.t. the sequences.

3.2 Canonicalization

Intuition. Equations of the linear system (see Appendix B, Eq. 11) tell that the hidden variables ($\vec{z_n}$) can have only a limited number of modes of operation that depend on the eigenvalues of the $\mathbf{A}$ matrix: The behavior can be exponential decay (real eigenvalue, with magnitude less than 1), exponential growth (real eigenvalue, with magnitude greater than 1), sinusoidal periodicity of increasing / constant / decreasing amplitude (complex eigenvalue $a + bi$ controlling both amplitude and frequency) and mixtures of the above. Those eigenvalues directly capture the amplitude and frequencies of the underlying signals of *hidden* variables, which we refer to as *harmonics* (Definition 1). Our goal in this step is to identify the canonical form of the hidden harmonics and how they mix in the observation sequences.

Theory. We know that a set of similar matrices share the same eigenvalues [9]. Hence, we propose to perform the eigenvalue decomposition of the transition matrix $\mathbf{A}$, and obtain the corresponding eigen-dynamics matrix and eigenvectors.

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^* \tag{1}$$

where $\mathbf{V} * \mathbf{V}^* = I$. The matrix $\mathbf{V}$ contains the eigenvectors of $\mathbf{A}$ and $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues of $\mathbf{A}$. We can justify doing the decomposition because over $\mathbb{C}$ almost every matrix is diagonalizable. Specifically, the probability that a square matrix of arbitrary fixed size with real entries is diagonalizable over $\mathbb{C}$ is 1 (see Zhang [32]). Also without loss of generality, we assume the eigenvalues are grouped into conjugate pairs (if any) and ordered according to their phases.

Note that the *output matrix* $\mathbf{C}$ in LDS represents how the hidden variables translate into observation sequences with linear combinations. In order to obtain the same observation sequences from $\mathbf{\Lambda}$ as the transition matrix, we need to compensate the *output matrix* $\mathbf{C}$ to get the *harmonic mixing matrix* $\mathbf{C}_h$.

$$\mathbf{C}_h = \mathbf{C} \cdot \mathbf{V} \tag{2}$$

Similarly, the *canonical hidden variables* will be:

$$\vec{\mu}_0^{new} = \mathbf{V}^* \cdot \vec{\mu}_0 \tag{3}$$

$$\vec{z}_n^{new} = \mathbf{V}^* \cdot \vec{z}_n \tag{4}$$

The following two lemmas tell how the *harmonic mixing matrix* $\mathbf{C}_h$ looks like and how the canonical hidden variables correspond to the original dynamical system:

LEMMA 1. *$\mathbf{V}$ has conjugate pairs of columns corresponding to the conjugate pairs of eigenvalues in $\mathbf{\Lambda}$. Hence, the* harmonic mixing matrix $\mathbf{C}_h$ *must contain conjugate pair of columns corresponding to the conjugate pairs of the eigenvalues in $\mathbf{\Lambda}$.*
Proof Sketch: See Appendix.

LEMMA 2.

$$\vec{z}_n^{new} = \mathbf{\Lambda}^{n-1} \cdot \vec{\mu}_0^{new} + noise \tag{5}$$

$$\vec{x}_n = \mathbf{C}_h \cdot \mathbf{\Lambda}^{n-1} \cdot \vec{\mu}_0^{new} + noise \tag{6}$$

Proof Sketch: See Appendix.

The intuition is that all hidden variables $\vec{z}_n$, all canonical hidden variables $\vec{z}_n^{new}$, and all observations $\vec{x}_n$ ($n = 1, \ldots, T$) are mixtures of a set of growing, shrinking or stationary sinusoid signals, of data-dependent frequencies; we refer to those signals as "*harmonics*", and their characteristic frequencies and amplitudes are completely defined by the eigenvalues of the *transition matrix* $\mathbf{A}$. "Harmonics" are formally defined as:

DEFINITION 1. *A signal y_n is said to be a harmonic if it is in the form of $y_n = (a+bi)^n$, where $a+bi$ is an eigenvalue of $\mathbf{A}$ and $i = \sqrt{-1}$.*

Our definition of *harmonic* is related to the frequencies that the Fourier transform would discover, with two major differences: (a) *exponential amplitude*: our harmonic functions could be growing or shrinking exponentially (for $a \neq 1$) (b) *generality*: the frequencies of the Discrete Fourier Transform (DFT) are always multiples of the base frequency $1/T$, while our harmonics could have any arbitrary frequency (b could take any value that fits the given data sequences).

Example. On performing this step on the learned *transition matrix* from the 5 synthetic sequences, we will get a 6×6 $\mathbf{\Lambda}$ matrix with eigenvalues $0.998 \pm 0.057i$, $0.998 \pm 0.063i$, and $0.978 \pm 0.208i$ on the diagonal. From our above discussion, we know that when the real part of the eigenvalue = 1, then the signal is a sinusoid - which is the case here. The imaginary part on the other hand corresponds to the frequencies. Thus $0.057 \approx 2\pi/110$ corresponds to frequency $1/110$, $0.063 \approx 2\pi/100$ corresponds to frequency $1/100$ and $0.208 \approx 2\pi/30$ corresponds to frequency $1/30$ - which are precisely the base frequencies in the data.

Figure 3(a) shows the matrix $\mathbf{C}_h$ obtained for the sample signals. We have shown the entries in the standard polar form: $Ae^{\phi i}$, A is the magnitude and ϕ is the angle (phase). For clarity, the values which are very small have been shown as 0 in the matrix. Firstly as expected from Lemma 1, we have conjugate pairs of columns corresponding to the eigenvalues which correspond to the frequencies. Secondly, note that signals (a) and (b) are the same sinusoid but with just different phases (specifically a phase difference of $\pi/2$). Hence in the matrix $\mathbf{C}_h$ they have the same frequency components (high values only in the conjugate columns 3 and 4) with the same weights (same A value) but with *different* phases (different ϕ values). So, if we directly try to cluster $\mathbf{C}_h$, we will not place them in the same cluster. Thirdly, the phase difference is also preserved in $\mathbf{C}_h$: $0.82 + 0.75 = 1.5708 = \pi/2 =$ the phase difference between signals (a) and (b). This can also be verified for the signals (d) and (e) which have different phases for the two constituent frequencies.

3.3 Handling Lag Correlation: Polar Form

Intuition. As specified in Section 1, the ideal features should catch lag correlation. After computing the *harmonic mixing matrix* $\mathbf{C}_h$, we have found the contribution of each harmonic in the resulting observation sequences. Each row in $\mathbf{C}_h$ represents the characteristics of each data sequence in the domain of the harmonics. Thus $\mathbf{C}_h$ can plausibly be used to cluster the sequences. However, the harmonic mixing matrix not only tells the strength of each eigen-dynamic but will also encode the required *phases* for different sequences. Thus we will fail to group similar motions just due to the lag or phase differences. Intuitively for example, suppose we have two almost identical walking motions, except that one starts from the left foot and another from the right foot. We want to extract features that could identify the walking behavior, no matter which foot it starts with, so that we would be able to group the two walking motions together.

Theory. We eliminate phase by taking the magnitude of the *harmonic mixing matrix* $\mathbf{C}_h$ $abs(\mathbf{C}_h)$. From Lemma 1 we will get the same column for those conjugate columns of $\mathbf{C}_h$; we drop these duplicate columns to get the *harmonic magnitude matrix* $\mathbf{C}_m$. The *harmonic magnitude matrix* $\mathbf{C}_m$ tells how strong each base harmonic participates in the observation time sequences and naturally leads to lag independent features (P1, Sec. 1).

Lemma 3. *$abs(\mathbf{C}_h)$ contains pairs of identical columns.*

Example. Figure 3 (b) and (c) show the matrix $\mathbf{C}_m$ obtained after applying this step on the generated $\mathbf{C}_h$ matrix for the synthetic signals. Note that $\mathbf{C}_m$ has begun to show some clear patterns corresponding to the underlying true clusters.

3.4 Grouping Harmonics

Intuition. The *harmonic magnitude matrix* $\mathbf{C}_m$ captures the contributing coefficients of each individual frequency. As we find harmonic frequency sets in music, in real time-series like motions, we will expect to usually find motion sequences composed of several major frequencies. Hence we now want to find such harmonics groups (P3 as stated in Section 1) capable of describing common characteristics of similar motion sequences, and the corresponding representations of each sequence in such harmonics group space. As a concrete example, say we want to determine that walking sequences are composed of 10 *units* of frequency 1 and 1 units of frequency 2, while running motions have say 10 units of frequency 2 and 1 units of frequency 3. Furthermore, a fast walking motion may in fact have a proper mix of both a walking frequency-group and running frequency-group.

Theory. To achieve this goal, we can use any dimensional reduction method such as SVD/PCA, ICA or nonnegative matrix factorization. For simplicity, we take the singular value decomposition (SVD) of the *harmonic magnitude matrix* $\mathbf{C}_m$. As we introduced earlier, SVD is capable of finding low rank projection of the data matrix. $\bar{\mathbf{C}_m} \approx \mathbf{U}_k \cdot \mathbf{S}_k \cdot \mathbf{V}_k^T$ where $\bar{\mathbf{C}_m}$ is column centered from $\mathbf{C}_m$, $\mathbf{U}_k$ and $\mathbf{V}_k$ are orthogonal matrices with k columns, and $\mathbf{S}_k$ is a diagonal matrix. The diagonal of $\mathbf{S}_k$ contains k singular values which are usually sorted by magnitude. Finally, we obtain the features as follows:

$$\mathbf{F} = \mathbf{U}_k \cdot \mathbf{S}_k \tag{7}$$

Example. Figure 2(d) shows the final $\mathbf{F}$ matrix obtained from the sample sequences. Note that this matrix very clearly brings out the correct groupings. Also notice that in the corresponding $\mathbf{C}_m$ matrix, sequences (d) and (e) had high components on columns 1 and 3 (which map to the 2 frequencies generating those signals). But after doing SVD, $\mathbf{F}$ gives us a clearer and simpler picture where they are shown to be more related by having the same 'group' of harmonics combined in the same way.

3.5 Discussion

Choosing h:. A particular issue in the learning algorithm is choosing a proper number h for the hidden dimension of underlying LDS. In practice, we use the "80%-95%" energy criterion to determine h [5]. That is, we take the Singular Value Decomposition of $\mathbf{X}$, rank the singular values, and then choose h at the rank with 95% of the total sum of squared singular values: $\hat{h} \leftarrow \arg_h \frac{\sum_{j=1}^{h} s_j^2}{\sum_{i=1}^{m} s_i^2}$, where s_i's are singular values of $\mathbf{X}$ in descending order.

Complexity:

Lemma 4. *The straightforward implementation of the algorithm (refered to as PLiF-naïve) costs $O(\#iteration \cdot \mathrm{T} \cdot (m^3 + h^3))$, where #iteration is the number of iterations for learning LDS.*

However, using the Woodbury matrix identity [9] and incrementally computing inverse of covariance matrix [25], the complexity can be reduced dramatically as stated in Lemma 5. See Appendix C.3 for details.

Lemma 5. *PLiF can be computed within time of $O(\#iteration\cdot (\mathrm{T} \cdot (m^2 \cdot h + h^3)) + m \cdot h^2)$.*

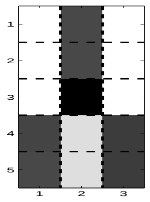
4. EXPERIMENTAL RESULTS

$$\left|\begin{array}{cccccc} 0 & 0 & 0.46e^{-0.82i} & 0.46e^{0.82i} & 0 & 0 \\ 0 & 0 & 0.46e^{0.75i} & 0.46e^{-0.75i} & 0 & 0 \\ 0 & 0 & 0.65e^{-0.03i} & 0.65e^{0.03i} & 0 & 0 \\ 0.50e^{0.70i} & 0.50e^{-0.70i} & \approx 0 & \approx 0 & 0.53e^{-0.48i} & 0.53e^{0.48i} \\ 0.50e^{-0.87i} & 0.50e^{0.87i} & \approx 0 & \approx 0 & 0.53e^{0.31i} & 0.53e^{-0.31i} \\ \multicolumn{2}{c}{\underbrace{\qquad\qquad}_{\text{Harmonic pair } H_1}} & \multicolumn{2}{c}{\underbrace{\qquad\qquad}_{\text{Harmonic pair } H_2}} & \multicolumn{2}{c}{\underbrace{\qquad\qquad}_{\text{Harmonic pair } H_3}} \end{array}\right|$$

(a) $\mathbf{C}_h$

$$\left|\begin{array}{ccc} 0 & 0.46 & 0 \\ 0 & 0.46 & 0 \\ 0 & 0.65 & 0 \\ 0.50 & \approx 0 & 0.53 \\ 0.50 & \approx 0 & 0.53 \end{array}\right|$$

(b) $\mathbf{C}_m$



(c) $\mathbf{C}_m$ heat-map

Figure 3: Running example: the synthetic, sinusoidal signals of Figure 2(a), and the output matrices according to PLiF: (a) The *harmonic mixing matrix* $\mathbf{C}_h$ and (b) the *harmonic magnitude matrix* $\mathbf{C}_m$ and (c) its heat-map (darker color - higher value in that cell). Near-zero values: omitted for clarity. Notice that (1) the columns of (a) are complex conjugates, in pairs; (2) the *harmonic magnitude matrix* $\mathbf{C}_m$ makes similar sequences to look similar (top 3, bottom 2).

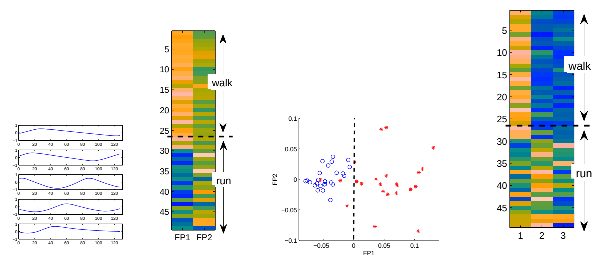
Please see a description of our experimental environment in the Appendix C.2.

4.1 Effectiveness: Visualization

As stated in the introduction section, our proposed method PLiF is capable of producing meaningful features: each feature column corresponds to a group of "harmonic" frequencies (one or more) and features represent the participation coefficients of the harmonic group in the sequence.

For the `MOCAP` dataset, we found interpretable and interesting patterns in its fingerprints (Fig. 4). In our experiment, we use hidden dimension $h = 7$ as suggested by the 95% criteria, and produce two fingerprints for each sequence ($k = 2$). The walking motions exhibit strong correlation with harmonics with eigen-values $0.998 \pm 0.053i$, equivalent to the frequency of $1/119$, while the running ones are correlated with eigenvalues $1.007 \pm 0.082i$ and $0.989 \pm 0.108i$, equivalent to the frequencies of $1/78$ and $1/58$.

We already presented meaningful features, both visually and numerically, extracted from multiple sequences by our proposed PLiF method. Thanks to those features, PLiF can be readily used for almost all mining tasks for time series, namely clustering, compression, forecasting and segmentation. While forecasting and segmentation are straightforward brought by the underlying dynamical system of our method, we will focus on the particular application of PLiF in time series clustering and compression.



(a) Original (b) Fingerprints (c) Scatter plot (d) *harmonic magnitude matrix* $\mathbf{C}_m$

Figure 4: Mocap fingerprints and visualization. 4(a) displays several sample sequences, top two of which are walking (#15 and #22), followed by two running ones (#45 and #38) and a running-to-stop motion (#8). 4(b): Each motion(row) displays two fingerprints. Upper 26 rows are walking motion, and the rest are running motion. 4(c): Walking motion are in blue circles, and running in red stars. Note the three red stars close to circles turn out to be abnormal motions: running to stop (#8 and #57), and right turn (#43).

4.2 Effectiveness: Clustering

The rationale in our clustering method lies in the fact that the fingerprints (features) computed by PLiF characterize how much each "harmonic" group participates in each of the sequences. Essentially, such a fingerprint tells the projection of each sequence onto the basis of the "harmonic" group. The final clustering result can be then obtained by applying any state-of-the-art clustering algorithm, such as k-means or spectral clustering [11] (Chap 14.3.6 & Chap 14.5.3) on the fingerprints.

In our experiments, we use simple thresholding (=0) on the first fingerprint (FP1) to tell the group, equivalent to k-means on FP1. In this way PLiF can produce two class grouping. But it can be easily extended to handle multiple class case, through the hierarchical framework [11] (Chap 14.3.12): applying PLiF-clustering in each level to produce bi-clustering and further dividing in proper descendants.

In `MOCAP` we test the clustering result on the right foot marker position with sequences of equal length ($\mathrm{T} = 107, m = 49$). Since we know the true labels of each motion in `MOCAP`, we adopt a standard measure of conditional entropy from the confusion matrix of prediction labels against true labels to evaluate the clustering quality. The conditional entropy (CE) tells the difference of two clustering (lower is better), based on the following equation: $CE = -\sum \frac{CM_{ij}}{\sum_{ij} CM_{ij}} \log \frac{CM_{ij}}{\sum_j CM_{ij}}$.

We use a commonly practiced method as the baseline for comparison: first projecting the multiple sequences into low dimensional principal components (#dim=#class=2) and then clustering by k-means with Euclidean distance. Tab. 4(a) and 4(b) show the confusion matrices and their

Table 4: Clustering on MOCAP right foot marker z-coordinate: Confusion matrix and conditional entropy. Note the ideal confusion matrix will be diagonal, which has conditional entropy of 0. Note in both way PLiF wins.

(a) PCA-Kmeans: $CE = 0.68$

predicted	walk	run
-1	15	13
1	11	10

(b) PLiF: $CE = 0.18$

predicted	walk	run
-1	26	3
1	0	20

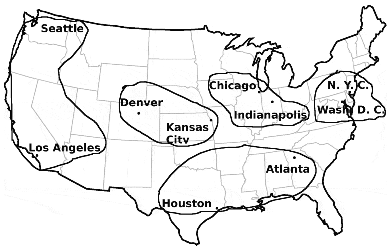


Figure 5: PLiF-clustering on BGP traffic data. Note how geographically close routers have been clustered together.

conditional entropies from the predicted grouping by the baseline and by PLiF clustering respectively. Note while baseline makes nearly random guesses, our method could identify all walking and almost all running motions correctly. The only three (out of 49) mistakes by PLiF turn out to be two running to stop motions and a right turn. As a typical example in Fig. 4(a), those mistakes have a very similar pattern with walking motion, so that even human would be confused.

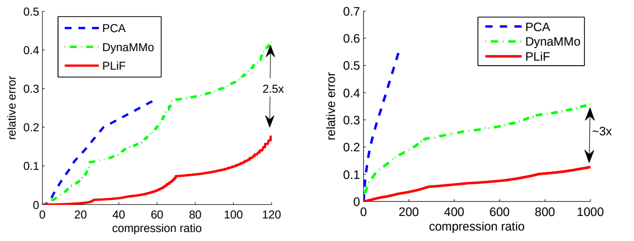
As an exploratory example, we use PLiF-clustering to find groups on BGP data - we do not have ground-truth labels of each sequence here. Fig. 5 shows the results (each cluster is shown encircled). Note that the results match well with the notion that geographically closer routers tend to be more correlated than others. This is because the BGP routing protocol itself tries to find shorter routes which results in packets being sent locally to nearby routers rather than routers far away. Thus closer routers may have time shifts and correlations that are captured by PLiF.

4.3 Compression

The fingerprints extracted by PLiF could be used in a compression setting as well. The basic idea is to store the eigen-dynamics matrix ($\mathbf{\Lambda}$), its associated projection matrix ($\mathbf{C}_h$) and a subset of expected value of hidden variables. From Sec. 3.2, the eigen-dynamics $\mathbf{\Lambda}$ is a diagonal matrix, so we only keep the diagonal part. We also keep $\mathbb{E}[\vec{z}_i]$ computed from the E-step of EM algorithm for LDS. To be able to recover from compression, we compute the hidden values using $\vec{\mu}_i = \mathbf{V}^* \cdot \mathbb{E}[\vec{z}_i]$. PLiF-compression finds a subset of $J \subseteq \{1, \ldots, \text{T}\}$, determining which time tick of hidden values will be stored. Here we use a similar idea as DynaMMo compression [21] to select the best subset of time tick index using dynamics programming. To recover the original signal, we project back the data matrix from those hidden variables and dynamics using the following equations:

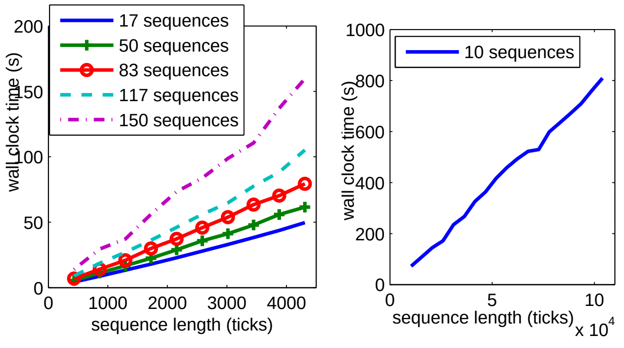
$$\vec{x}_i = \mathbf{C}_h \cdot \vec{\mu}_i \tag{8}$$

$$\vec{\mu}_j = \mathbf{\Lambda}^{j-i} \vec{\mu}_i \qquad \text{if } i \in J \wedge i+1, \ldots, j \notin J \tag{9}$$



(a) MOCAP walking(#22) (b) CHLORINE

Figure 6: Compression: normalized reconstruction error versus compression ratio. Note PLiF achieves up to three times better than state-of-the-art method DynaMMo compression.



(a) CHLORINE (b) BGP

Figure 7: PLiF computation time versus the length of sequences on CHLORINE and BGP datasets: linear as expected.

We did compression experiments on both MOCAP and CHLORINE data and evaluated the quality by relative error defined as: relative error $= \frac{mse(\hat{\mathbf{X}}-\mathbf{X}) \cdot m}{\sum_i var(X_i)}$ where mse denotes mean square error and var variance for each sequence. Fig. 6(a) and 6(b) show respectively PLiF-compression results for a walking motion (#22) and CHLORINE compared with PCA and DynaMMo [21]. Note here the statistics are generated by varying over different h and number of time ticks of hidden variables to keep, and we only plot the skyline of compression ratio and error.

4.4 Scalability

We now evaluate the scalability of PLiF on both MOCAP and CHLORINE data. We took various sizes of the CHLORINE sequences (by truncation) to test the scalability with respect to the length and the number of sequences.

Fig. 7(a) and 7(b) show the wall clock time of PLiF with respect to the length of sequences, on five different number of sequences from CHLORINE data, and on 10 sequences from BGP data (after taking the logarithm). In each experiment, we set the number of hidden variables $h = 15$ for CHLORINE and $h = 10$ for BGP and the learning step runs at the same number of iterations (= 20). In Fig. 7(a) and 7(b), all wall clock times fall in to almost straight line, indicating the linear scalability of PLiF over the length of sequences.

We did experiment on MOCAP dataset to compare the speed of PLiF and PLiF-naive. Fig. 8 presents wall clock time on

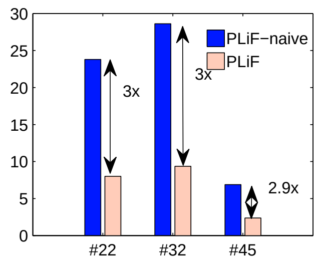


Figure 8: Wall clock time of PLiF versus PLiF-naive on and MOCAP: upto 3x gains. Similarly experiment on CHLORINE dataset obtains 3.5x speedup.

three typical MOCAP sequences, one walking motion (#22), one jumping motion and one running motion (#45). PLiF is 3 times faster PLiF-naive. Experiments on the CHLORINE dataset reveals similarly, PLiF scales much better than the basic algorithm PLiF-naive over the number of sequences and gets up to 3.5 times faster than the latter.

5. CONCLUSIONS

The main idea is the proposal and design of PLiF, for the extraction of "*fingerprints*" from a collection of co-evolving time sequences. PLiF has all of the following desirable characteristics,

1. *Effectiveness:* The resulting features correspond to membership weights in each harmonics group; thus, they capture correlations, despite the presence of lags, and despite small shifts in frequency. The resulting distance function agrees with human intuition and the provided ground truth. Thus, fingerprints lead to good clustering, as well as visualization.
2. *Interpretability*: The fingerprints correspond to groups of harmonics, which are easy to interpret.
3. *Forecasting:* PLiF can easily do forecasting, since it is based on linear dynamical systems and their corresponding difference equations. Thus, it can easily do forecasting and compression, outperforming SVD and state-of-the-art compression methods (see Figure 6(a),6(b)).
4. *Scalability:* PLiF is fast and scalable, being *linear* on the length of the sequences.

We showed the basic version of PLiF, as well as the final one. Both are linear on the length of sequences, but PLiF can be up to 3.5 times faster, thanks to our Lemma 5.

Future work could focus on testing PLiF's performance on additional datasets and its use for segmentation and anomaly detection, which as we mentioned are natural by-products of any method that can do forecasting. One limitation of the current proposed method is the inability to handle sequences of non-uniform lengths. In such cases, the naïve way of truncating sequences wastes part of data. Hence further work may also target extending PLiF to cluster time series with different lengths.

6. REFERENCES

[1] G. E. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice Hall, Englewood Cliffs, NJ, 3rd edition, 1994.

[2] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004.

[3] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, Minneapolis, MN, May 25-27 1994.

[4] A. W.-C. Fu, E. J. Keogh, L. Y. H. Lau, and C. A. Ratanamahatana. Scaling and time warping in time series querying. In *VLDB*, pages 649–660, 2005.

[5] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, San Diego, CA, 1990.

[6] M. Garofalakis, J. Gehrke, and R. Rastogi. *Data Stream Management: Processing High-Speed Data Streams*. Springer, 2009.

[7] Z. Ghahramani and G. E. Hinton. Parameter estimation for linear dynamical systems. Technical Report CRG-TR-96-2, February 1996.

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

[9] G. H. Golub and C. F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[10] D. Gunopulos and G. Das. Time series similarity measures and time series indexing. In *SIGMOD Conference*, Santa Barbara, CA, 2001. Tutorial.

[11] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, corrected edition, July 2003.

[12] M. Jahangiri, D. Sacharidis, and C. Shahabi. Shift-split: I/o efficient maintenance of wavelet-transformed multidimensional data. In *SIGMOD Conference*, pages 275–286, 2005.

[13] A. Jain, E. Y. Chang, and Y.-F. Wang. Adaptive stream resource management using kalman filters. In *SIGMOD*, pages 11–22, 2004.

[14] C. S. Jensen and S. Pakalnis. Trax - real-world tracking of moving objects. In *VLDB*, pages 1362–1365, 2007.

[15] I. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.

[16] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME C Journal of Basic Engineering*, (82 (Series D)):35–45, 1960.

[17] K. Kalpakis, D. Gada, and V. Puttagunta. Distance measures for effective clustering of arima time-series. In *ICDM*, pages 273–280, 2001.

[18] E. J. Keogh. Exact indexing of dynamic time warping. In *VLDB*, pages 406–417, 2002.

[19] E. J. Keogh, T. Palpanas, V. B. Zordan, D. Gunopulos, and M. Cardle. Indexing large human-motion databases. In *VLDB*, pages 780–791, 2004.

[20] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. *PODS*, pages 261–272, 1999.

[21] L. Li, J. McCann, N. Pollard, and C. Faloutsos. Dynammo: Mining and summarization of coevolving sequences with missing values. In *KDD*, New York, NY, USA, 2009. ACM.

[22] K. Mouratidis, M. L. Yiu, D. Papadias, and N. Mamoulis. Continuous nearest neighbor monitoring in road networks. In *VLDB*, pages 43–54, 2006.

[23] Ü. Y. Ogras and H. Ferhatosmanoglu. Online summarization of dynamic time series data. *VLDB J.*, 15(1):84–98, 2006.

[24] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala. Latent semantic indexing: A probabilistic analysis. In *PODS*, pages 159–168, 1998.

[25] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. *VLDB*, 2005.

[26] D. Rafiei and A. O. Mendelzon. Similarity-based queries for time series data. In *SIGMOD Conference*, pages 13–25, Tucson, AZ, 1997.

[27] G. Reeves, J. Liu, S. Nath, and F. Zhao. Managing massive time series streams with multiscale compressed trickles. *PVLDB*, 2(1):97–108, 2009.

[28] A. Safonova and J. K. Hodgins. Construction and optimal search of interpolated motion graphs. *ACM Trans. Graph.*, 26(3):106, 2007.

[29] R. H. Shumway and D. S. Stoffer. An approach to time series smoothing and forecasting using the em algorithm. *Journal of Time Series Analysis*, 3:253–264, 1982.

[30] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and indexing of moving objects with unknown motion patterns. In *SIGMOD*, pages 611–622, 2004.

[31] M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B*, 61:611–622, 1999.

[32] Z. N. Zhang. The jordan canonical form of a real random matrix. In *Numer. Math. J. Chinese Univ.*, 23(2001).

ACKNOWLEDGMENTS

This material is based on work supported by the Army Research Laboratory under Cooperative Agreement No. W911NF-09-2-0053, the National Science Foundation under Grants No. DBI-0640543, CNS-0721736, CNS-0721889 and under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (LLNL-CONF-404625), subcontracts B579447, B580840. This work is also partially supported by a Sprint gift. Any opinions, findings, and conclusions or recommendations in this material are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory, the U.S. Government, the National Science Foundation, or other funding parties. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

APPENDIX

A. ADDITIONAL RELATED WORK

There is a lot of work on time series analysis, on indexing, clustering, and forecasting.

Indexing, Signals and Streams: For indexing, the idea is to extract features [3] and then use a spatial access method. Typical features include the Fourier transform coefficients, wavelets (Gilbert et al., [8], Jahangiri et al. [12]) piece-wise linear approximations (Keogh et al. [18]).These are mainly useful for the Euclidean distance, or variations (Rafiei et al [26], Ogras et al [23]). Indexing for motion databases has also attracted attention, both in the database community (eg., Keogh et al. [19]) as well as in graphics (Safonova et al. [28]).

The typical distance function is the Euclidean distance; the other major competitor is the time warping distance, also known as Dynamic Time Warping (DTW) (e.g., see the tutorial by Gunopulos and Das [10]). The linear-time constrained versions of DTW (Itakura parallelogram, Sakoe-Chiba band) have been studied in [18, 4].

There is also vast, recent literature on indexing moving objects (Jensen et al. [14] Mouratidis et al. [22]), as well as streams (e.g., see the edited volume by Garofalakis et al. [6]). An additional recent application for time series is monitoring a data center (eg., Reeves et al. [27]), where the goal is to observe patterns in order to minimize energy consumption. An equally important monitoring application is environmental sensors (e.g., Deshpande et al. [2]).

Dimensionality reduction: There are numerous papers on the topic, with typical methods being PCA [15], SVD/LSI [5] and random projections [24].

Autoregression: Autoregression is the standard first step for forecasting. It is part of the ARIMA methodology, pioneered by Box and Jenkins [1], and it discussed in every textbook in time series analysis and forecasting. Kalpakis et al [17] used autoregression to extract features, using the so-called *cepstrum* method from voice processing. Kalman filters and Linear Dynamical Systems are closely related to autoregression, trying to detect hidden variables (like velocity, acceleration) at every time-tick, and use them for forecasting. In the database community, Kalman filters have been proposed for sensor data (Jain et al [13]) as well as for moving objects (Tao et al [30]).

All the above approaches are powerful and very popular for their intended problem. In fact, the proposed PLiF method uses some of them as stepping stones (LDS, PCA). However, none of them achieves all the goals we set in the introduction.

B. SPECIAL CASES & THEIR SHORTCOMINGS

There are several existing methods, but none matches all the desirable properties illustrated in Table 1. Thus, none is a head-on competitor to our proposed PLiF method. We elaborate on PCA, Discrete Fourier Transform and Linear Dynamical Systems here because (a) they are the typical competitors for some (but not all) of the target tasks and (b) they can help in describing our PLiF method as well.

B.1 Principal Component Analysis

Principal Component Analysis (PCA) is the textbook method of doing dimensionality reduction, by spotting redundancies and (linear) correlations among the given sequences. Technically, it gives the optimal low rank approximation for the data matrix $\mathbf{X}$. In our running example of Section 2, the matrix $\mathbf{X}$ would be a 5×500 matrix, with one row for each sequence and one column per time-tick. Singular value decomposition (SVD) is the typical method to compute PCA. For a data matrix $\mathbf{X}$ (assume $\mathbf{X}$ is zero-centered), SVD computes the decomposition

$$\mathbf{X} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T$$

where both $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrices, and $\mathbf{S}$ is a diagonal matrix with singular values on the diagonal. Using standard terminology from the PCA literature, $\mathbf{V}$ is called the *loading* matrix and $\mathbf{U} \cdot \mathbf{S}$ will be the component *score*. To achieve dimensionality reduction, small singular values are typically set to zero so that the retained ones maintain 80-90% of the energy (= sum of squares of eigenvalues). We shall refer to this rule of thumb as the *energy criterion* [5] (equivalently, this truncation produces a low rank projection). In our running example of Figure 2(a), the $\mathbf{U} \cdot \mathbf{S}$ component score matrix is a 5×2 matrix, since we are retaining 2 hidden variables.

PCA is effective in dimensionality reduction and in finding linear correlations, particularly for Gaussian distributed data [31]. However, the low dimensional projections are hard to interpret. Moreover, PCA can not capture time-evolving patterns (since it is designed to *not* care about the ordering of the rows or the columns), and thus it can not do forecasting. Fig. 2(b) shows the top two principal components for the synthetic five sequences. It does not show any clear pattern of underlying clusters; thus k-means indeed gives a poor final clustering result on it.

B.2 Discrete Fourier Transform

The T-point Discrete Fourier Transform (DFT) of sequence $(x_0, \ldots, x_{T-1})$ is a set of T complex numbers c_k, given by the formula

$$c_k = \sum_{t=0}^{\mathrm{T}-1} x_t e^{-\frac{2\pi i}{\mathrm{T}} kt} \qquad k = 0, \ldots, \mathrm{T}-1$$

where $i = \sqrt{-1}$ is imaginary unit.

The c_k numbers are also referred to as the *spectrum* of the input sequence. DFT is powerful in spotting periodicities in a single sequence, with numerous uses in signal, voice, and image processing. However, it is not clear how to assess the similarity between two spectra, and hence DFT can be unsuitable for clustering. Moreover, it has several limitations, namely:

1. it can not find arbitrary frequencies (only ones that are integer multiples of the base frequency),
2. it can not give partial credit for signals with nearby frequences ('frequency proximity', Property P2 in the introduction),
3. it can not do forecasting, other than blindly repeating the original signal.

Due to those limitations, we do not compare PLiF against DFT, in the experiments section under clustering.

B.3 Linear Dynamical Systems

Linear Dynamical Systems (LDS), also known as Kalman filters, have been used previously to model multi-dimensional continuous valued time series. The model is described by the following equations:

$$\vec{z}_1 = \vec{\mu}_0 + \vec{\omega}_1 \tag{10}$$
$$\vec{z}_{n+1} = \mathbf{A}\vec{z}_n + \vec{\omega}_{n+1} \tag{11}$$
$$\vec{x}_n = \mathbf{C}\vec{z}_n + \vec{\epsilon}_n \tag{12}$$

where $\vec{\mu}_0$ is initial state of the whole system.

The model assumes the observed data sequences ($\vec{x}_n$) are generated from the a series of hidden variables ($\vec{z}_n$) with a linear projection matrix $\mathbf{C}$, and the hidden variables are evolving over time with linear transition matrix $\mathbf{A}$, so that next time tick only depends on the previous time tick as in Markov chains. All noises ($\vec{\omega}$'s and $\vec{\epsilon}$'s) arising from the process are modeled as independent Gaussian noises with covariances $\mathbf{Q}_0$, $\mathbf{Q}$ and $\mathbf{R}$ respectively. Given the observation series, there exist algorithms for estimating hidden variables [16] and EM algorithms for learning the model parameters [29, 7], with publicly available implementations[4]. The EM algorithm maximizes $L(\theta)$, the expected log-likelihood defined in Eq. 13, iteratively. In the **E** step, it estimates the posterior distribution of the hidden variables conditioned on the data sequence with fixed model parameters; in the **M** step, it then updates the model parameters by maximizing the likelihood using some sufficient statistics (e.g. mean and covariance) from the posterior distribution.

$$\begin{aligned} L(\theta;\mathcal{X}) &= \mathbb{E}_{\mathcal{X},\mathcal{Z}|\theta}[-D(\vec{z}_1,\vec{\mu}_0,\mathbf{Q}_0) \\ &\quad -\sum_{t=2}^{\mathrm{T}} D(\vec{z}_t,\mathbf{A}\vec{z}_{t-1},\mathbf{Q}) - \sum_{t=1}^{\mathrm{T}} D(\vec{x}_t,\mathbf{C}\vec{z}_t,\mathbf{R}) \\ &\quad -\frac{1}{2}\log|\mathbf{Q}_0| - \frac{\mathrm{T}-1}{2}\log|\mathbf{Q}| - \frac{T}{2}\log|\mathbf{R}|] \end{aligned} \tag{13}$$

where $D()$ is the square of the Mahalanobis distance, i.e. $D(\vec{x},\vec{y},\Sigma) = (\vec{x}-\vec{y})^T\Sigma^{-1}(\vec{x}-\vec{y})$.

The difference between our proposed PLiF and LDS is that in addition to learning straight forward transitions and projections, PLiF will further discover deeper patterns behind them. The problem with LDS learning (see Fig. 2(c)) is that the learned model parameters are hard to interpret.

[4]http://people.cs.ubc.ca/∼murphy/software/kalman/kalman.html

C. EXPERIMENTS AND ALGORITHM DETAILS

C.1 Proof Sketches

Lemma 1. Consider the eigenvalue equation $\mathbf{A}\cdot x = \lambda x$, where x is the eigenvector. Taking the conjugate on both sides we get $A\cdot\overline{x} = \overline{\lambda}\overline{x}$. As $\mathbf{A}$ contains only real entries, $\mathbf{A}\cdot\overline{x} = \overline{\lambda}\overline{x}$. Hence, if the conjugate $\overline{\lambda}$ is an eigenvalue of $\mathbf{A}$, $\overline{x}$ is also a corresponding (conjugate) eigenvector. □

Lemma 2.

$$\begin{aligned} \vec{z}_n^{new} &= \mathbf{V}^*\cdot\vec{z}_n \\ &= \mathbf{V}^*\cdot\mathbf{A}\cdot\vec{z}_{n-1} + noise \\ &= \mathbf{V}^*\cdot\mathbf{V}\cdot\mathbf{\Lambda}\cdot\mathbf{V}^*\cdot\vec{z}_{n-1} + noise \\ &= \mathbf{\Lambda}\cdot\vec{z}_{n-1}^{new} + noise \\ \vec{x}_1 &= \mathbf{C}\cdot\vec{\mu}_0 + noise = \mathbf{C}\cdot\mathbf{V}\cdot\mathbf{V}^*\cdot\vec{\mu}_0 + noise \\ &= \vec{C}_h\cdot\vec{\mu}_0^{new} + noise \\ \vec{x}_2 &= \mathbf{C}\cdot\vec{z}_2 + noise = \mathbf{C}\cdot\mathbf{A}\cdot\vec{z}_1 + noise \\ &= \mathbf{C}\cdot\mathbf{V}\cdot\mathbf{\Lambda}\cdot\mathbf{V}^*\cdot\vec{\mu}_0 + noise \\ &= \mathbf{C}_h\cdot\mathbf{\Lambda}\cdot\vec{\mu}_0^{new} + noise \end{aligned}$$

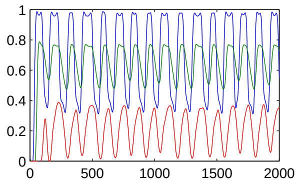
The result then follows by induction on the number of time ticks. □

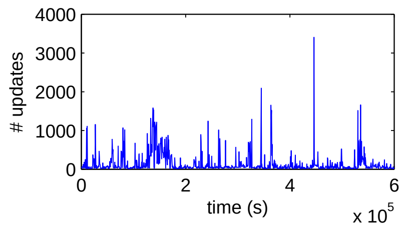
C.2 Experimental Setup

We describe here our experimental setup and datasets.

- Mocap data (`MOCAP`): Motion capture involves recording human motion through tracking the marker movement on human actors and then turn them into a series of multi-dimensional coordinates in 3d space. We use a publicly available mocap dataset from CMU[5]. It includes 49 walking and running motions of subject#16. Each motion sequence contains 93 positions for 31 markers in body local coordinate and three reference coordinates.
- Chlorine Data (`CHLORINE`): The chlorine dataset is publicly available[6] and it contains m=166 sequences of Chlorine concentration measurements on a water network over 15 days at the rate of one sample per 5 minutes (T=4310 time ticks). The dataset was produced by the EPANET 2 hydraulic analysis package[7], which reflects periodic patterns (daily cycles, dominating residential demand pattern) in the Chlorine concentration, with a few exceptions and time shifts (see sample sequences in Fig. 9(a)).
- Router Data (`BGP`): We examine BGP Monitor data containing 18 million BGP update messages over a period of two years (`09/2004` to `09/2006`) from the Datapository project[8]. We consider the number of updates received by a router every 10 minutes. A snippet is shown in Fig. 9(b). As the signals are very bursty, we take their logarithms (see Fig. 9(c)). The preprocessed

[5]http://mocap.cs.cmu.edu
[6]www.cs.cmu.edu/afs/cs/project/spirit-1/www/data/cl2fullLarge.zip
[7]http://www.epa.gov/nrmrl/wswrd/dw/epanet.html
[8]http://www.datapository.net/bgpmon/


(a) CHLORINE


(b) BGP: for the router at Washington DC

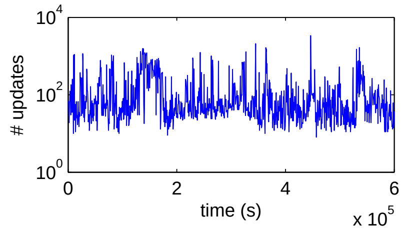

(c) BGP (Washington DC), in log scale

Figure 9: Sample snippets from datasets. (a) CHLORINE shows daily periodicity; (b) BGP is bursty with no periodicities, thus we take the logarithm (shown in part (c)). No obvious patterns, in neither (b) nor (c).

BGP time series in the experiment consists of m=10 sequences (routers) of T=103,968 time ticks. The routers are in 10 major centers (Atlanta, Washington DC, Seattle etc.).

Our algorithms are implemented in Matlab 2008b, and running on a machine with Windows XP, 3.2GHz dual core CPU and 2G RAM.

C.3 Proposed PLiF: Scaling Up

```
Algorithm 1: PLiF
   Input: X: $m$ sequences with duration T, and $k$
   Output: fingerprints F
1  choose $h$ by 80%-95% energy criterion ;
   // learning dynamics (see Sec.3.1 and Eq.13)
2  $\mathbf{A}, \mathbf{C} \leftarrow \arg\max_\theta L(\theta; \mathbf{X})$ ;
   // canonicalization, see Sec.3.2
3  compute $\mathbf{\Lambda}$, $\mathbf{V}$ s.t. $\mathbf{A}\cdot\mathbf{V} = \mathbf{V}\cdot\mathbf{\Lambda}$;
   // compensating, see Sec.3.2
4  $\mathbf{C}_h = \mathbf{C}\cdot\mathbf{V}$;
   // obtain polar form, see Sec.3.3
5  $\mathbf{D} \leftarrow$ keep conjugate pairs of columns in $\mathbf{C}_h$;
6  $\mathbf{E} \leftarrow$ take element-wise magnitude of $\mathbf{D}$;
7  $\mathbf{C}_m \leftarrow$ eliminate duplicate columns in $\mathbf{E}$;
   // finding harmonics grouping, see Sec.3.4
8  $\overline{\mathbf{C}_m} \leftarrow \mathbf{C}_m - \text{mean}(\mathbf{C}_m)$;
9  compute $\mathbf{U}_k, \mathbf{S}_k, \mathbf{V}_k \leftarrow \arg\min ||\overline{\mathbf{C}_m} - \mathbf{U}_k\cdot\mathbf{S}_k\cdot\mathbf{V}_k^T||_{fro}$
   ;
10 $\mathbf{F} \leftarrow \mathbf{U}_k\cdot\mathbf{S}_k$;
```

We refer to the algorithm given in Section 3 as PLiF-naive (pseudo-code given in Alg. 1), which gives the intuitive idea and motivation for each algorithmic step. In this section we will focus on the scalability of the algorithm and propose a faster implementation. Since PLiF-naive involves a fair amount of matrix inversion, it is cubic to the size of matrix of interest. To speed up the algorithm, our idea is to perform smarter and faster matrix inversion instead of straight-forward inversion. We will first analyze the complexity of PLiF-naive. We make an assumption here that the length of data sequence is much greater than the dimension, T $\gg m$, which is the common cases as otherwise the resulting LDS model will be under-determined.

Proof Sketch of Lemma 4. : In each iteration of learning LDS (Alg. 1, step 2), it does $m\times m$ and $h\times h$ matrix inversion for T length of sequences. Rest all steps including eigen value decomposition in canonicalization, taking polar form and grouping harmonics with SVD, take at most $O(m^3)$. Therefore, PLiF-naive has complexity of $O(\#iteration\cdot \mathrm{T}\cdot(m^3+h^3))$. □

The time complexity of PLiF-naive is linear with respect to the length of sequences, but cubic to the number of sequences. Without going too much into the details, we describe briefly where such cubic complexity arises in PLiF-naive. The major cubic computation involves calculation of the inverse of the following $m\times m$ matrix while learning LDS in the first step:

$$(\mathbf{C}\mathbf{P}_n\mathbf{C}^T+\mathbf{R})^{-1} \tag{14}$$

Here $\mathbf{C}$ is size $m\times h$, $\mathbf{P}_n$ is $h\times h$, and $\mathbf{R}$ is $m\times m$. $\mathbf{P}_n$ is a complicated matrix, hence we omit details of $\mathbf{P}_n$ for brevity (full details can be found in [16]). This is updated in each time tick while learning the LDS model (Alg. 1, step 2). Cubic computation elsewhere is only a negligible fraction in the typical case of $m \ll$ T.

How to scale up: We will focus on speeding up the computation of the inversion. The idea underlying our method is using the Woodbury matrix identity [9] to perform an inverse on a smaller matrix ($h\times h$) instead of direct large matrix ($m\times m$) inversion. In typical cases, h is much smaller than m, therefore it will be significantly faster. We can then substitute the matrix inversions in Alg. 1 step 2 with faster ones. We will refer to this faster version as PLiF.

Lemma 5. PLiF can be computed within time of $O(\#iteration\cdot \mathrm{T}\cdot(m^2\cdot h+h^3)+m\cdot h^2)$.

Proof Sketch. : We will analyze Alg. 1 step by step. The Woodbury formula tells the following identity for invertible $\mathbf{X}$:

$$(\mathbf{X}+\mathbf{YZY}^T)^{-1} = \mathbf{X}^{-1}-\mathbf{X}^{-1}\mathbf{Y}(\mathbf{Z}^{-1}+\mathbf{Y}^T\mathbf{X}^{-1}\mathbf{Y})^{-1}\mathbf{Y}^T\mathbf{X}^{-1}$$

By applying the identity, we obtain the following equation for computing Eq. 14:

$$\mathbf{R}^{-1}-\mathbf{R}^{-1}\mathbf{C}(\mathbf{P}_n^{-1}+\mathbf{C}^T\mathbf{R}^{-1}\mathbf{C})^{-1}\mathbf{C}^T\mathbf{R}^{-1} \tag{15}$$

Since $\mathbf{C}$ and $\mathbf{R}$ are fixed inside each iteration and only updated once at the end of each iteration, we can therefore precompute $\mathbf{R}^{-1}$, $\mathbf{R}^{-1}\mathbf{C}$ and $\mathbf{C}^T\mathbf{R}^{-1}\mathbf{C}$ in the beginning of each learning iteration. For each time tick of the data sequence, the EM algorithm for learning LDS requires the inversion of two $h\times h$ matrices and multiplication of $m\times h$, $h\times h$ and $h\times m$ matrices. Thus it takes $O(\mathrm{T}\cdot(m^2h+h^3))$ during each iteration of learning the LDS assuming T $\gg m$, and it follows that Alg. 1 step 2 takes $O(\#iteration \cdot \mathrm{T}(m^2h+h^3))$.

In addition, Alg. 1 step 3 takes $O(h^3)$, step 4 takes $O(mh^2)$, and step 10 takes $O(mh^2)$. $\square$