Fast Mining of a Network of Coevolving Time Series

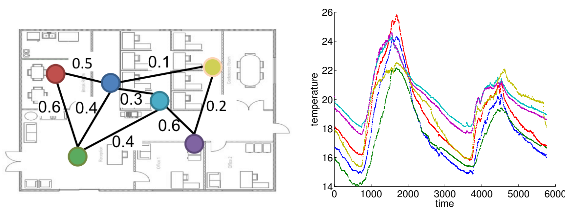
Yongjie Cai* Hanghang Tong† Wei Fan‡ Ping Ji*

Abstract

Coevolving multiple time series are ubiquitous and naturally appear in a variety of high-impact applications, ranging from environmental monitoring, computer network traffic monitoring, motion capture, to physiological signal in health care and many more. In many scenarios, the multiple time series data is often accompanied by some contextual information in the form of *networks*. In this paper, we refer to such multiple time series, together with its embedded network as *a network of coevolving time series*. In order to unveil the underlying patterns of a network of coevolving time series, we propose DCMF, a dynamic contextual matrix factorization algorithm. The key idea is to find the latent factor representation of the input time series and that of its embedded network simultaneously. Our experimental results on several real datasets demonstrate that our method (1) outperforms its competitors, especially when there are lots of missing values; and (2) enjoys a linear scalability w.r.t. the length of time series.

1 Introduction

Coevolving multiple time series are ubiquitous and naturally appear in a variety of high-impact applications, ranging from environmental monitoring, computer network traffic monitoring, motion capture, to physiological signal in health care and many more. In many scenarios, the multiple time series data is often accompanied by some contextual information in the form of networks. For example, in a smart building equipped with various sensors, each sensor generates a time series by measuring a specific type of room condition (e.g., temperature, humidity, etc.) over time. All these sensors or time series are connected with each other by the underlying sensor network; in epilepsy study, the various sensor measures collected from different brain regions of a patient are essentially inter-connected by his or her brain graph; in citation analysis, the citations of different journals or conferences are correlated, particularly for those from similar domains. In this paper, we refer to such multiple time series, together with its embedded network as *a network of coevolving time series*.



(a) Sensor network (b) Time series

Figure 1: An illustrative example of a network of time series. (a) A simplified sensor network. (b) Measured temperature time series, each of which corresponds to a node with the same color of the sensor network in (a). The multiple time series in (b) are inter-connected with each other by its embedded network in (a). (Best viewed in color).

Figure 1 presents an illustrative example of such time series. Figure 1(a) is a simplified version of sensor deployment at an office floor. The weights on the labeled edges indicate the closeness and connectivity between two sensors. Figure 1(b) is two-day temperature time series measured from (a)'s sensor network. The color is used to differentiate different sensors and their corresponding time series. As we can see, the purple line and the cyan line are very close to each other, and the weight between their sensors is relative high.

In order to unveil the underlying patterns of a network of coevolving time series, in particular to fill in the missing values of one or more input time series, we adapt the concept of collaborative filtering that multiple coevolving time series and/or the contextual network are determined by a few latent factors, and propose DCMF, a dynamic contextual matrix factorization algorithm to simultaneously find the common latent factor in both the input time series and its embedded network.

The main contributions of this paper can be summarized as follows.

- **Problem Definition**. We introduce a novel setting on mining a network of coevolving time series.
- **Algorithms and Analysis**. We propose a dynamic contextual matrix factorization algorithm to recover the missing values in a network of time se-

*The Graduate Center, CUNY, {ycai@gc,pji@jjay}.cuny.edu
†Arizona State University, hanghang.tong@asu.edu
‡Big Data Labs - Baidu USA, fanwei03@baidu.com

Table 1: Symbols and Definitions

Symbol	Definition and Description
$\mathbf{A}$,...	matrices (bold upper case)
$\mathbf{A}_{ij}$	the element at the i^{th} row and j^{th} column of matrix $\mathbf{A}$
$\mathbf{A}_j$	the j^{th} column of matrix $\mathbf{A}$
$\mathbf{A}(i,:)$	the i^{th} row of matrix $\mathbf{A}$
$\mathbf{A}'$	transpose of matrix $\mathbf{A}$
$\mathbf{a}, \mathbf{b}, ...$	column vectors (bold lower case)
$\odot$	element-wise multiplication
$\mathbf{X}$	observed time series $(\mathbf{x}_1, .., \mathbf{x}_T)$
$\mathbf{W}$	indicator matrix
$\mathbf{S}$	contextual matrix
$\mathbf{Z}$	time series latent matrix $(\mathbf{z}_1, ..., \mathbf{z}_T)$
$\mathbf{B}$	transition matrix
$\mathbf{U}$	object latent matrix
$\mathbf{V}$	contextual latent matrix
n	number of objects in $\mathbf{X}$ and $\mathbf{S}$
T	length of time series
l	dimension of latent variables

ries; and analyze its effectiveness, complexity and the relationship with the existing tools on mining multiple time series.

- **Empirical Evaluations**. Our experimental results on several real datasets demonstrate that our method (1) outperforms its competitors, especially when there are lots of missing values; and (2) enjoys a linear scalability w.r.t. the length of time series.

The rest of this paper is organized as follows: In Section 2, we introduce the notations and formally define the problem. We present the proposed solution and its analysis in Section 3 and Section 4, respectively. We provide the experimental results in Section 5. The related work is reviewed in Section 6, followed by the conclusions in Section 7.

2 Problem Definition

Table 1 lists the main symbols we use throughout this paper. Besides the standard notations, we use a matrix $\mathbf{X}^{n \times T}$ to denote observed time series with n dimensions and T time steps, where $\mathbf{X}_{it}$ denotes the data from i^{th} object at time t . We use an indicator matrix $\mathbf{W}^{n \times T}$ to indicate whether data is observed or missing in $\mathbf{X}$, i.e., $\mathbf{W}_{it} = 0$ when $\mathbf{X}_{it}$ is missing, otherwise $\mathbf{W}_{it} = 1$. We use a contextual matrix $\mathbf{S}^{n \times n}$ to represent the embedded network.

With the above notations, **a Network of Time Series (NoT)** can be formally defined as follows:

Definition 2.1. *A Network of Time Series (NoT). Given an n dimensional partially observed time series with T time steps $\mathbf{X}^{n \times T}$, an indicator matrix $\mathbf{W}^{n \times T}$, an $n \times n$ contextual matrix $\mathbf{S}^{n \times n}$ representing the network behind $\mathbf{X}$, and a one-to-one mapping function ζ, which maps each dimension of the time series $\mathbf{X}$ to a node in $\mathbf{S}$, a Network of Time Series is defined as the quadruplet $\mathcal{R} = \langle \mathbf{X}, \mathbf{W}, \mathbf{S}, \zeta \rangle$.*

The problem of time series missing value recovery can be defined as follows:

Problem 2.1. *NoT Missing Value Recovery.*
Given*: a network of time series $\mathcal{R} = \langle \mathbf{X}, \mathbf{W}, \mathbf{S}, \zeta \rangle$;*
Recover*: its missing parts indicated by the indicator matrix $\mathbf{W}$.*

3 Dynamic Contextual Matrix Factorization

In this section, we present our solution for NoT Missing Value Recovery problem (Problem 2.1). We start with the discussions of some baseline solutions and their limitations, followed by our formulation to encode the temporal smoothness and the corresponding algorithm.

3.1 Preliminaries and Challenges

We would like to point out that Problem 2.1 bears some similarity to the classic collaborative filtering problem. To be specific, if we treat the rows (time series) and the columns (time steps) of $\mathbf{X}$ as users and items, respectively; the observed entries in $\mathbf{X}$ as the 'ratings' between the corresponding users and items; and the contextual matrix $\mathbf{S}$ as the 'social network' among different users, Problem 2.1 can be viewed as inferring the missing or unknown ratings between the users (rows of $\mathbf{X}$) and the items (columns of $\mathbf{X}$). Having this in mind, it seems that many collaborative filtering methods can be used to solve Problem 2.1. For example, if we only use the partially observed time series matrix $\mathbf{X}$, we can use the matrix factorization based collaborative filtering (Baseline Solution 3.1) [10, 15]. If we further incorporate the contextual matrix $\mathbf{S}$, it is essentially a social recommendation problem (Baseline Solution 3.2) [8, 13].

Baseline Solution 3.1. *Collaborative Filtering (via Matrix Factorization)*
Given*: a rank size l, and a rating matrix $\mathbf{X}^{n \times m}$ with missing values indicated by $\mathbf{W}^{n \times m}$, where $\mathbf{X}_{ij}$ denotes user i's rating on item j and $\mathbf{X}_{ij} = 0$ if there is no rating,*
Find*: user latent factor $\mathbf{U}^{n \times l}$ and item latent factor $\mathbf{Z}^{l \times m}$ such that $\mathbf{X} \approx \mathbf{W} \odot (\mathbf{UZ})$. The rating predictions are estimated as the non-zero values of $\tilde{\mathbf{X}}^{n \times m}$, where $\tilde{\mathbf{X}} = (1 - \mathbf{W}) \odot (\mathbf{UZ})$.*

Baseline Solution 3.2. *Social Collaborative Filter-*

ing (via joint matrix factorization).
Given*: a rank size l, and a rating matrix $\mathbf{X}^{n\times m}$ with missing values indicated by $\mathbf{W}^{n\times m}$, where $\mathbf{X}_{ij}$ denotes user i's rating on item j and $\mathbf{X}_{ij}=0$ if there is no rating; a social network matrix $\mathbf{S}^{n\times n}$, where $\mathbf{S}_{ij}$ indicates how much user i trust or is influenced by user j,*
Find*: a user latent factor $\mathbf{U}^{n\times l}$, an item latent factor $\mathbf{Z}^{l\times m}$ and a social latent factor $\mathbf{V}^{l\times n}$ such that $\mathbf{X}\approx\mathbf{W}\odot(\mathbf{UZ})$ and $\mathbf{S}\approx\mathbf{UV}$. The rating predictions are estimated as the non-zero values of $\tilde{\mathbf{X}}^{n\times m}$, where $\tilde{\mathbf{X}}=(1-\mathbf{W})\odot(\mathbf{UZ})$.*

Baseline Solution 3.1 and 3.2 can be represented by the graphical models shown in Figure 2(a) and 2(b). Both solutions are to find the similarities between users indicated by the user latent factor $\mathbf{U}$. Baseline Solution 3.1 only uses the rating matrix, while Baseline Solution 3.2 fuses the rating matrix with the social matrix and finds the user latent factor shared by both matrices. In many multiple time series applications, we can also find such similarities between these time series. For example, in time series obtained from a sensor network, nearby sensors may collect similar measurements; in a water quality dataset, the chlorine concentration levels of two connected pipes can be very close.

However, in both Baseline Solution 3.1 and 3.2, the temporal smoothness, a unique characteristic of time series data, is ignored. For instance, the sensor data (e.g., temperature, light, humidity) or the chlorine concentration levels collected in two consecutive time steps might not change a lot. Ignoring such temporal smoothness is likely to lead to sub-optimal or even infeasible solutions of Problem 2.1. For example, both Baseline Solution 3.1 and 3.2 would fail if a certain number of columns of $\mathbf{X}$ are missing entirely.

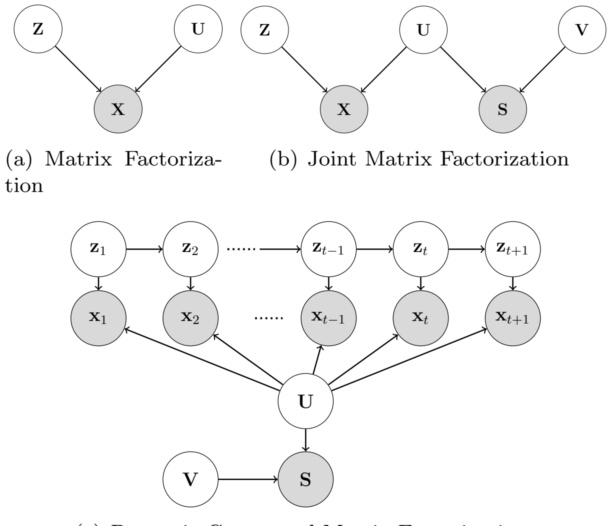
3.2 Proposed Formulation

We propose a novel algorithm, Dynamic Contextual Matrix Factorization (DCMF), based on joint matrix factorization and linear dynamic systems, as illustrated in Figure 2(c).

To model the temporal smoothness, we first re-write the time series $\mathbf{X}$ as column vectors $(\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_T)$, where each $\mathbf{x}_t$ represents n dimensional data at time t; and the corresponding hidden factors $\mathbf{Z}$ as $(\mathbf{z}_1,\mathbf{z}_2,...,\mathbf{z}_T)$. Then we define $\mathbf{z}_t$ to be linear to $\mathbf{z}_{t-1}$ with a transition process $\mathbf{B}$ and multivariate transition noise $\omega_t\sim\mathcal{N}(0,\Lambda)$. The state for the latent variable at the first time step $\mathbf{z}_1$ is defined by $\mathbf{z}_0$ with multivariate Gaussian noise $\omega_0\sim\mathcal{N}(0,\Psi_0)$:

$$\mathbf{z}_1=\mathbf{z}_0+\omega_0,\quad \mathbf{z}_t=\mathbf{B}\mathbf{z}_{t-1}+\omega_t, \tag{3.1}$$

Similar to the joint matrix factorization model, we



(a) Matrix Factorization

(b) Joint Matrix Factorization

(c) Dynamic Contextual Matrix Factorization

Figure 2: Comparison between the baseline solutions (a and b) and the proposed formulation (c).

define $\mathbf{x}_t$ to be linear to $\mathbf{z}_t$ through the latent factor $\mathbf{U}$ with multivariate Gaussian noise $\epsilon_t\sim\mathcal{N}(0,\Sigma)$, $\mathbf{s}_j$ to be linear to $\mathbf{v}_j$ through the latent factor $\mathbf{U}$ with multivariate Gaussian noise $\tau_j\sim\mathcal{N}(0,\Xi)$:

$$\mathbf{x}_t=\mathbf{W}_t\odot(\mathbf{U}\mathbf{z}_t+\epsilon_t), \tag{3.2}$$

$$\mathbf{s}_j=\mathbf{U}\mathbf{v}_j+\tau_j. \tag{3.3}$$

We also place zero-mean spherical Gaussian priors on $\mathbf{V}$ under the assumption that each entry of $\mathbf{S}$ is scaled to $[0,1]$:

$$p(\mathbf{V}|\Gamma)=\prod_{j=1}^{n}\mathcal{N}(\mathbf{v}_j|0,\Gamma). \tag{3.4}$$

In order to accommodate our model in sparse time series datasets, we simplify the multivariate Gaussian noises by assuming that the transition noises and observation noises are independent and identically distributed (*i.i.d.*), and the covariances of latent variable $\{\mathbf{v}_j\}$ are *i.i.d.*, i.e. $\mathbf{v}_j\sim\mathcal{N}(0,\sigma_V^2)$, and

$$\omega_t\sim\mathcal{N}(0,\sigma_Z^2),\epsilon_t\sim\mathcal{N}(0,\sigma_X^2),\tau_j\sim\mathcal{N}(0,\sigma_S^2),$$

where $t=1,...,T$ and $j=1,...,n$.

With Eq. (3.1)-(3.4) and the parameters $\theta=\{\mathbf{B},\mathbf{z}_0,\Psi_0,\sigma_Z^2,\sigma_X^2,\sigma_S^2,\sigma_V^2\}$, our goal here is to maximize the joint distribution described as follows:

$$\operatorname*{argmax}_{\mathbf{Z},\mathbf{U},\mathbf{V},\theta} p(\mathbf{Z},\mathbf{X},\mathbf{V},\mathbf{S},\mathbf{U}) = p(\mathbf{z}_1) \underbrace{\prod_{t=2}^{T} p(\mathbf{z}_t|\mathbf{z}_{t-1})}_{\text{temporal smoothness}}$$

$$\underbrace{\prod_{t=1}^{T} p(\mathbf{x}_t|\mathbf{z}_t,\mathbf{U})}_{\text{observed time series}} \times \prod_{j=1}^{n} p(\mathbf{v}_j) \underbrace{\prod_{j=1}^{n} p(\mathbf{s}_j|\mathbf{v}_j,\mathbf{U})}_{\text{contextual network}}, \tag{3.5}$$

where we omit the model parameters in the equation.

3.3 Proposed Algorithm

In Eq. (3.5), $\mathbf{U}$ and $\mathbf{z}_t$ jointly determine $\mathbf{x}_t$, and $\mathbf{U}$ and $\mathbf{v}_j$ jointly determine $\mathbf{S}$. Due to such coupling, it is difficult to find its global optimal solution. Hence, we aim to find its local optimal instead, by grouping $\{\mathbf{z}_t, \mathbf{v}_j\}$ and updating $\mathbf{U}$ and $\{\mathbf{z}_t, \mathbf{v}_j\}$ alternatively. Regarding $\mathbf{U}$ as a parameter of the model, the solution of our model contains parameters $\theta = \{\mathbf{U}, \mathbf{B}, \mathbf{z}_0, \Psi_0, \sigma_Z^2, \sigma_X^2, \sigma_S^2, \sigma_V^2\}$ and the latent factors $\{\mathbf{z}_t\}, \{\mathbf{v}_j\}$.

To be specific, we adopt the expectation-maximization (E-M) algorithm to find a local optimal solution of our model as follows. We first randomly set the initial values of the parameters θ^{old}, then iteratively perform E-M steps until the convergence criterion is satisfied. In the E step, we infer the posteriors of the latent variables $\{\mathbf{z}_t\}, \{\mathbf{v}_j\}$ based on the old parameters θ^{old}. In the M step, we update the new parameters θ^{new} that maximize the expectation of the distribution likelihood.

3.3.1 E Step: updating the latent factors

Given $\mathbf{U}$, $\{\mathbf{z}_t, \mathbf{x}_t\}$ are independent with $\{\mathbf{v}_j, \mathbf{s}_j\}$. We can estimate the posteriors of $\{\mathbf{z}_t\}$ and $\{\mathbf{v}_j\}$ independently w.r.t. $\mathbf{U}$ (steps 3-12 in Algorithm 1).

Eq. (3.1) and (3.2) are very similar to linear dynamic systems, except that only observed entries in $\mathbf{x}_t$ contribute to Eq. (3.5). Let $\mathbf{o}_t$ denote the indices of the observed entries of $\mathbf{x}_t$, and $\mathbf{x}_t^*$ denote the observed entries of $\mathbf{x}_t$, which is a compressed version of $\mathbf{x}_t$, we use $\mathbf{H}_t$ to represent the corresponding compressed version of $\mathbf{U}$. Formally, $\mathbf{o}_t$, $\mathbf{x}_t^*$ and $\mathbf{H}_t$ can be formulated as:

$$\begin{aligned} \mathbf{o}_t &= \{i|\mathbf{W}_{it} > 0, i = 1, ..., n\}, \\ \mathbf{x}_t^* &= \mathbf{x}_t(\mathbf{o}_t), \quad \mathbf{H}_t = \mathbf{U}(\mathbf{o}_t, :) \\ \mathbf{x}_t^* &= \mathbf{H}_t\mathbf{z}_t + \epsilon_t \end{aligned} \tag{3.6}$$

Then, we can apply the forward-backward algorithm [4] or Kalman Filter and RTS smoother [17, 19] to find the posteriors of the latent variables $\{\mathbf{z}_t\}$. Define

$$\begin{aligned} p(\mathbf{z}_t|\mathbf{x}_1, ..., \mathbf{x}_t) &= \mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_t, \boldsymbol{\Psi}_t), \\ p(\mathbf{z}_t|\mathbf{x}_1, ..., \mathbf{x}_T) &= \mathcal{N}(\mathbf{z}_t|\hat{\boldsymbol{\mu}}_t, \hat{\boldsymbol{\Psi}}_t). \end{aligned} \tag{3.7}$$

By using the forwarding algorithm, we have:

$$\begin{aligned} \boldsymbol{\mu}_t &= \mathbf{B}\boldsymbol{\mu}_{t-1} + \mathbf{K}_t(\mathbf{x}_t^* - \mathbf{H}_t\mathbf{B}\boldsymbol{\mu}_{t-1}) \\ \boldsymbol{\Psi}_t &= (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\mathbf{P}_{t-1} \\ \mathbf{P}_{t-1} &= \mathbf{B}\boldsymbol{\Psi}_{t-1}\mathbf{B}' + \sigma_Z^2\mathbf{I} \\ \mathbf{K}_t &= \mathbf{P}_{t-1}\mathbf{H}_t'(\mathbf{H}_t\mathbf{P}_{t-1}\mathbf{H}_t' + \sigma_X^2\mathbf{I})^{-1} \end{aligned} \tag{3.8}$$

with the initial status:

$$\begin{aligned} \boldsymbol{\mu}_1 &= \mathbf{z}_0 + \mathbf{K}_1(\mathbf{x}_1^* - \mathbf{H}_1\mathbf{z}_0) \\ \boldsymbol{\Psi}_1 &= (\mathbf{I} - \mathbf{K}_1\mathbf{H}_1)\boldsymbol{\Psi}_0 \\ \mathbf{K}_1 &= \boldsymbol{\Psi}_0\mathbf{H}_1'(\mathbf{H}_1\boldsymbol{\Psi}_0\mathbf{H}_1' + \sigma_X^2\mathbf{I})^{-1} \end{aligned} \tag{3.9}$$

By applying the backward algorithm, we have:

$$\begin{aligned} \hat{\boldsymbol{\mu}}_t &= \boldsymbol{\mu}_t + \mathbf{J}_t(\hat{\boldsymbol{\mu}}_{t+1} - \mathbf{B}\boldsymbol{\mu}_t) \\ \hat{\boldsymbol{\Psi}}_t &= \boldsymbol{\Psi}_t + \mathbf{J}_t(\hat{\boldsymbol{\Psi}}_{t+1} - \mathbf{P}_t)\mathbf{J}_t', \\ \mathbf{J}_t &= \boldsymbol{\Psi}_t\mathbf{B}'(\mathbf{P}_t)^{-1} \end{aligned} \tag{3.10}$$

$$\begin{aligned} \mathbb{E}[\mathbf{z}_t] &= \hat{\boldsymbol{\mu}}_t \\ \mathbb{E}[\mathbf{z}_t\mathbf{z}_{t-1}'] &= \hat{\boldsymbol{\Psi}}_t\mathbf{J}_{t-1}' + \hat{\boldsymbol{\mu}}_t\hat{\boldsymbol{\mu}}_{t-1}' \\ \mathbb{E}[\mathbf{z}_t\mathbf{z}_t'] &= \hat{\boldsymbol{\Psi}}_t + \hat{\boldsymbol{\mu}}_t\hat{\boldsymbol{\mu}}_t' \end{aligned} \tag{3.11}$$

Then, we apply Bayes' theorem on Eq. (3.3)-(3.4) to obtain the posteriors $p(\mathbf{v}_j|\mathbf{s}_j) = \mathcal{N}(\mathbf{v}_j|\boldsymbol{\nu}_j, \Upsilon)$:

$$\begin{aligned} \boldsymbol{\nu}_j &= \mathbf{M}^{-1}\mathbf{U}'\mathbf{s}_j, \quad \Upsilon = \sigma_S^2\mathbf{M}^{-1} \\ \mathbf{M} &= \mathbf{U}'\mathbf{U} + \sigma_V^{-2}\sigma_S^2\mathbf{I} \\ \mathbb{E}[\mathbf{v}_j] &= \boldsymbol{\nu}_j, \quad \mathbb{E}[\mathbf{v}_j\mathbf{v}_j'] = \Upsilon + \boldsymbol{\nu}_j\boldsymbol{\nu}_j'. \end{aligned} \tag{3.12}$$

3.3.2 M Step: updating the model parameters

In the M step, we update the parameters to maximize the expectation of the log likelihood (step 13 in Algorithm 1):

$$\begin{aligned} \theta^{new} &= \operatorname*{argmax}_{\theta} Q(\theta, \theta^{old}) \\ Q(\theta, \theta^{old}) &= \mathbb{E}_{\mathbf{Z},\mathbf{V}|\theta^{old}}[\ln p(\mathbf{Z},\mathbf{X},\mathbf{V},\mathbf{S}|\theta)], \end{aligned} \tag{3.13}$$

where $p(\mathbf{Z},\mathbf{X},\mathbf{V},\mathbf{S}|\theta)$ is defined as Eq. (3.5).

To obtain θ^{new}, we substitute the probabilities in Eq. (3.5) with the defined distributions, and calculate the derivatives of $Q(\theta, \theta^{old})$ w.r.t. each parameter, and then set each derivative to zero to get a new parameter estimation. The parameters are updated as follows (The detailed derivations are omitted for brevity):

$$
\begin{aligned}
\mathbf{z}_0^{new} &= \mathbb{E}[\mathbf{z}_1] \\
\Psi_0^{new} &= \mathbb{E}[\mathbf{z}_1\mathbf{z}_1'] - \mathbb{E}[\mathbf{z}_1]\mathbb{E}[\mathbf{z}_1'] \\
\mathbf{B}^{new} &= \left(\sum_{t=2}^{T}\mathbb{E}[\mathbf{z}_t\mathbf{z}_{t-1}']\right)\left(\sum_{t=2}^{T}\mathbb{E}[\mathbf{z}_{t-1}\mathbf{z}_{t-1}']\right)^{-1} \\
(\sigma_V^2)^{new} &= \frac{1}{nl}\sum_{j=1}^{n}\mathrm{tr}\left(\mathbb{E}[\mathbf{v}_j\mathbf{v}_j']\right) \\
(\sigma_Z^2)^{new} &= \frac{1}{(T-1)l}\,\mathrm{tr}\left(\sum_{t=2}^{T}\mathbb{E}[\mathbf{z}_t\mathbf{z}_t'] \right. \\
&\quad - \mathbf{B}^{new}\sum_{t=2}^{T}\mathbb{E}[\mathbf{z}_{t-1}\mathbf{z}_t'] - \sum_{t=2}^{T}\mathbb{E}[\mathbf{z}_t\mathbf{z}_{t-1}'](\mathbf{B}^{new})' \\
&\quad \left. +\mathbf{B}^{new}\sum_{t=2}^{T}\mathbb{E}[\mathbf{z}_{t-1}\mathbf{z}_{t-1}'](\mathbf{B}^{new})'\right)
\end{aligned}
\tag{3.14}
$$

For $\mathbf{U}$, we update each row $\mathbf{U}(i,:)$ individually:

$$
\begin{aligned}
&\mathbf{U}(i,:)^{new} = \mathbf{A}_1\mathbf{A}_2^{-1} \\
&\mathbf{A}_1 = \lambda\sigma_S^{-2}\sum_{j=1}^{n}\mathbf{S}_{ij}\mathbb{E}[\mathbf{v}_j'] + (1-\lambda)\sigma_X^{-2}\sum_{t=1}^{T}\mathbf{W}_{ij}\mathbf{X}_{it}\mathbb{E}[\mathbf{z}_t'] \\
&\mathbf{A}_2 = \lambda\sigma_S^{-2}\sum_{j=1}^{n}\mathbb{E}[\mathbf{v}_j\mathbf{v}_j'] + (1-\lambda)\sigma_X^{-2}\sum_{t=1}^{T}\mathbf{W}_{it}\mathbb{E}[\mathbf{z}_t\mathbf{z}_t'],
\end{aligned}
\tag{3.15}
$$

where λ is to trade off the contributions of the contextual matrix and time series. With $\mathbf{U}^{new}$, σ_S^2 and σ_X^2 are then updated as follows:

$$
\begin{aligned}
(\sigma_S^2)^{new} &= \frac{1}{n^2}\left[\sum_{j=1}^{n}\left(\mathbf{s}_j'\mathbf{s}_j - 2\mathbf{s}_j'\left(\mathbf{U}^{new}\mathbb{E}[\mathbf{v}_j]\right)\right)\right. \\
&\quad \left. + \mathrm{tr}\left(\mathbf{U}^{new}(\sum_{j=1}^{n}\mathbb{E}[\mathbf{v}_j\mathbf{v}_j'])(\mathbf{U}^{new})'\right)\right] \\
(\sigma_X^2)^{new} &= \frac{1}{\sum_{t=1}^{T}\sum_{i=1}^{n}\mathbf{W}_{it}}\left(\sum_{t=1}^{T}\mathrm{tr}\left(\mathbf{H}_t^{new}\mathbb{E}[\mathbf{z}_t\mathbf{z}_t'](\mathbf{H}_t^{new})'\right)\right. \\
&\quad \left. + \sum_{t=1}^{T}\left((\mathbf{x}_t^*)'\mathbf{x}_t^* - 2(\mathbf{x}_t^*)'(\mathbf{H}_t^{new}\mathbb{E}[\mathbf{z}_t])\right)\right),
\end{aligned}
\tag{3.16}
$$

where $\sum_{t=1}^{T}\sum_{i=1}^{n}\mathbf{W}_{it}$ is the number of the observed entries in $\mathbf{X}$ and $\mathbf{H}_t^{new}$ is derived from $\mathbf{U}^{new}$ based on Eq. (3.6).

3.3.3 The Overall Algorithm

Putting everything together, we have the overall algorithm (*DCMF*) to solve Eq. (3.5), which is summarized in Algorithm 1. Given a network of time series $\mathcal{R}$, the dimension of latent factors l, and the weight of contextual information λ, the algorithm aims to tune model parameters and approximate the values of the latent factors. It first randomly initializes the model parameters θ (including $\mathbf{U}$) (step 1); and then iteratively infers the expectations of $\mathbf{Z}$, $\mathbf{V}$ (step 3-11), and updates the model parameters θ (step 13) until convergence. The missing values can be inferred from the reconstructed time series $\hat{\mathbf{X}}$ (step 16).

Algorithm 1: Dynamic Contextual Matrix Factorization (DCMF)

```
   Input  : a network of time series
            $\mathcal{R} = \langle \mathbf{X}, \mathbf{W}, \mathbf{S}, \zeta \rangle$,
            dimension of latent factors $l$,
            weight of contextual information $\lambda$
   Output: $\theta$, $\mathbf{Z}, \mathbf{V}$
1  Initialize $\theta = \{\mathbf{U}, \mathbf{B}, \mathbf{z}_0, \Psi_0, \sigma_Z^2, \sigma_X^2, \sigma_S^2, \sigma_V^2\}$;
2  repeat
3      for t=1:T do
4          Construct $\mathbf{H}_t$ based on Eq. (3.6);
5          Estimate $\boldsymbol{\mu}_t, \boldsymbol{\Psi}_t$ using Eq. (3.8),(3.9)
6      end
7      for t=T:1 do
8          Estimate $\hat{\boldsymbol{\mu}}_t$, $\hat{\boldsymbol{\Psi}}_t$, $\mathbb{E}[\mathbf{z}_t\mathbf{z}_{t-1}']$, $\mathbb{E}[\mathbf{z}_t\mathbf{z}_t']$ using
           Eq. (3.10), (3.11)
9      end
10     for j=1:n do
11         Estimate $\mathbb{E}[\mathbf{v}_j], \mathbb{E}[\mathbf{v}_j\mathbf{v}_j']$ using Eq. (3.12)
12     end
13     Update $\theta$ by Eq. (3.14) - (3.16)
14 until convergence;
15 Set $\mathbf{Z} = (\hat{\boldsymbol{\mu}}_1, ..., \hat{\boldsymbol{\mu}}_T)$, $\mathbf{V} = (\boldsymbol{\nu}_1, ..., \boldsymbol{\nu}_T)$
16 Reconstructed $\hat{\mathbf{X}} = \mathbf{UZ}$
```

4 Analysis of The Algorithm

In this section, we analyze the proposed *DCMF* algorithm in terms of its effectiveness, complexity and relationship with the existing methods.

4.1 Effectiveness

In terms of effectiveness, our *DCMF* algorithm finds a local optimal solution for the model (Eq. (3.5)), following the standard EM algorithm procedure [4]. In the E step, we find the maximized posteriors of the latent factors $\mathbf{Z}$ and $\mathbf{V}$ with θ^{old} fixed. In the M step, we update θ so as to maximize the expectation of the complete-data log-likelihood Q with the derived distributions of $\mathbf{Z}$ and $\mathbf{V}$ in the E step.

4.2 Time Complexity

The time complexity for the algorithm is summarized in Lemma 4.1, where n_t represents the number of the observed entries of $\mathbf{x}_t$. Notice that $\sum_t(ln_t^2+n_t^3)$ is upper bounded by n^3T. Also, in reality, we often have that the length (T) of the time series is orders of magnitude larger than the number of the time series (n) (See the data description in Section 5 for some examples). Thus, the actual running time of our *DCMF* algorithm is dominated by those terms related to the length of the time series T, all of which are linear in T.

LEMMA 4.1. *The time complexity of the DCMF algorithm is* $O(\#iterations \cdot (l^2nT + l^2n^2 + \sum_t(ln_t^2 + n_t^3)))$.

Proof. Omitted for Brevity.

4.3 Space Complexity

The space complexity of the proposed algorithm is summarized in Lemma 4.2.

LEMMA 4.2. *The space complexity of the DCMF algorithm is* $O(nT + l^2T + n^2 + l^2n)$.

Proof. Omitted for Brevity.

4.4 Model Comprehension

Our model in Eq. (3.5) is *comprehensive* in the sense that it simultaneously captures (1) the correlation among different time series; (2) the contextual information encoded by the contextual matrix; and (3) the temporal smoothness along the time dimension. As such, our *DCMF* algorithm includes several existing methods as its special cases, including

1 *Kalman filter and smoother*. If we exclude the contextual matrix by setting λ to 0, and data in $\mathbf{X}$ are all fully observed, our model degenerates to the traditional Kalman filter and smoother.

2 *DynaMMo[12]*. Our model is similar to DynaMMo if we set λ to 0. We should point out that in this case, both models can deal with missing values, but in different ways. Our model infers the missing values directly from the observed values, while DynaMMo fills in the missing values using linear interpolation (or other methods) first, then apply Kalman filter and smoother to adjust the filled missing values. As we will show in Section 5, DynaMMo would not work well in sparse time series since the artificial (probably incorrect) data can mislead the optimization process.

3 *SoRec[13]*. If we remove temporal smoothness by setting $\mathbf{B}$ to 0 and Λ to Ψ_0, and add zero-mean spherical Gaussian priors on $\mathbf{U}$, our model becomes SoRec.

4 *PMF[15]*. For *SoRec*, if we further remove the contextual matrix by setting λ to 0, our model reduces to PMF.

5 *SmoothSoRec* and *SmoothPMF*. Both *SoRec* and *PMF* can naturally encode the temporal smoothness by requiring the distribution of $\mathbf{z}_t$ to be Gaussian with $\mathbf{z}_{t-1}$ mean. These are also the special cases of our DCMF algorithm by setting $\mathbf{B}$ as an identity matrix.

5 Experimental Results

In this section, we present the empirical evaluations on real datasets, which are designed to answer the following two questions:

- *Effectiveness*: how accurate is the proposed DCMF algorithm in terms of recovering the missing values of the input time series?
- *Efficiency*: how does the proposed DCMF algorithm scale w.r.t. the size of the input time series?

5.1 Experimental Setup

A. Baseline Methods. We compare our method with the following state-of-the-art algorithms: Probabilistic Matrix Factorization (PMF) [15], Social Recommendation (SoRec) [13], SmoothPMF, SmoothSoRec, and DynaMMo [12]. We also compare DMF, which is a simplified version of DCMF with $\lambda = 0$.
B. Datasets. We use the following three real datasets in our experiments.

Motes Dataset The Motes dataset consists of temperature measurement from the sensors deployed at the Intel Berkeley Research Lab over a month [2]. The dataset also contains the locations of the sensors (i.e., x-y coordinates) and the averaged connectivity probabilities (i.e., the probabilities of messages from a sensor successfully reaching the other) over time.

Let d_{ij} and c_{ij} be the two-dimensional Euclidean distance and the connectivity probability between Sensor i and Sensor j, respectively, we define each entry of the contextual matrix $\mathbf{S}$ as follows:

$$\mathbf{S}_{ij} = \alpha \cdot (1 - d_{ij}/\max_{i,j}(d_{ij})) + (1-\alpha) \cdot c_{ij},$$

where α controls the contributions of the two parts. We set it to be 0.5 in the experiments.

Chlorine Dataset The Chlorine dataset was generated by EPANET-2 [1], which simulates the hydraulic and chemical phenomena within drinking water distribution systems. Given a network as input, EPANET-2 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. This dataset contains time series of the chlorine concentration levels from 166 pipe junctions during 15 days with a total of 4310 time steps. Since no network structure data is published in this dataset, we use the cosine similarity between each pair of time series to define the contextual matrix **S**.

Motion Capture Dataset This dataset contains a set of full body motions using 41 markers placed on human bodies [3]. In each motion, it records three-dimensional coordinates of each marker, resulting in a total of 123 features of marker positions. Each motion consists of 200 to 1500 frames (i.e. time steps).

We define the contextual information based on the positions (x, y, z coordinates) of the markers. Let $d(i, j)$ be the three-dimensional Euclidean distance between the markers of the i^{th} feature and j^{th} feature, $\mathbf{S}_{ij}$ is defined as:

$$\mathbf{S}_{ij} = \alpha \cdot \left(1 - d(i,j)/\max_{i,j} d(i,j)\right),$$

where α is set to be 1 if i and j correspond to the same dimension, and 0.5 otherwise.

C. Evaluation Metrics. To evaluate the effectiveness, we use the root mean squared error (RMSE):

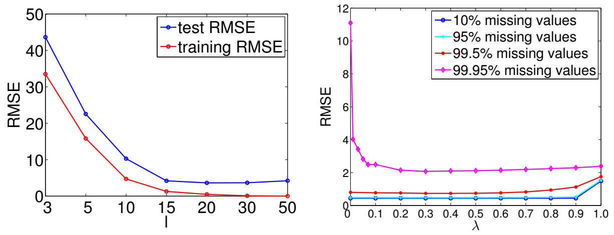
$$RMSE = \sqrt{\frac{\sum_{i,t}(1 - \mathbf{W}_{it})(\mathbf{X}_{it} - \hat{\mathbf{X}}_{it})^2}{\sum_{i,t}(1 - \mathbf{W}_{it})}},$$

where $\mathbf{X}_{it}$ is the observed value and $\hat{\mathbf{X}}_{it}$ is the corresponding reconstructed value, and **W** is the indicator matrix. $\mathbf{W}_{it} = 1$ indicates that $\mathbf{X}_{it}$ is an observed value (or is selected in the training set), and $\mathbf{W}_{it} = 0$ indicates that $\mathbf{X}_{it}$ is a missing value (or is selected in the test set).

5.2 Sensitivity Results

Our algorithm has two hyper-parameters: l and λ. l is the dimension of the latent factors . Figure 3(a) shows the impact of l on the training parts and test parts of a *running* instance in the Motion Capture dataset. As we can see, the training RMSE is constantly decreasing as l increases, while the test RMSE achieves the lowest value when l reaches to 15 and stabilizes after that with a slight increase. Based on that, we set $l = 15$ for the Motion Capture dataset. With similar experiments, we set $l = 5$ for the Motes dataset, and $l = 15$ for the Chlorine dataset.

λ balances the information from the contextual network **S** and time series **X**. If $\lambda = 0$, **S** is ignored. If $\lambda = 1$, only **S** is considered for learning **U**. Figure 3(b) shows the impact of λ on the Motes dataset. As we can see, the performance is quite stable when the training set of **X** contains sufficient information. But when it is very sparse (i.e., 99.95% missing values), the introduction of **S** significantly improves the accuracy.

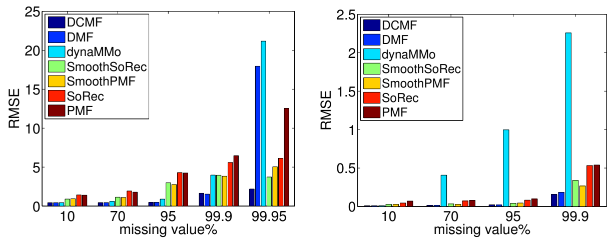


(a) impact of l ($n = 123$) (b) impact of λ

Figure 3: Parameter sensitivity results.

5.3 Effectiveness Results

To evaluate all the algorithms, we incrementally generate training and test sets of the Motes and Chlorine datasets with an increasing amount of test data (1%, 10%, ..., 99.9%) within time series. For example, to increase the size of test set from 1% to 10%, we randomly move 9% of the observed data from the training set to the test set. In this way, the subsequent test set always contains the test data of the previous one.



(a) Motes (n=54,T=14400) (b) Chlorine (n=166, T=4310)

Figure 4: Evaluation on missing value recovery. Lower is better. *DCMF* outperforms all of the competitors.

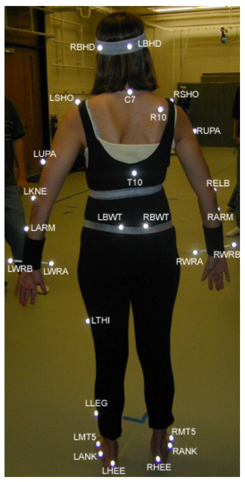
Figure 4 shows the results on the Motes and the Chlorine datasets. We can see that our DCMF algorithm consistently outperforms the other methods, especially when the percentage of the missing values is large.

As for the Motion Capture dataset, the missing values often occur consecutively [12]. Thus, we randomly choose some frame windows and mask all the values within these windows as test sets. Table 2 summarizes the result of missing value reconstruction errors on 6 motion instances. Again, our method DCMF and its simplified version DMF significantly outperform others in most cases.

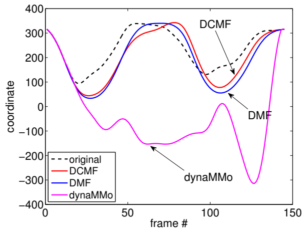
Figure 5 demonstrates a case study when most of the data of a marker are missing. Specifically, we hide the data of Marker LWRB from frame 13 to 142. The original data are shown as the dashed black lines in Figure 5(b)(c). The lines in red, blue, and purple are the reconstructed traces using DCMF, DMF and dynaMMo, respectively. As we can see, both

Table 2: RMSE for 10% (the same setting as dynaMMo [12]) missing values on the Motion Capture dataset ($l = 15, n = 123$). Notice that DMF is a simplified version of our DCMF algorithm.

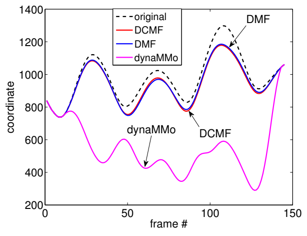
Motion	Running	Mawashi geri	Jump distance	Wave hello	Football throw	Boxing
T	145	1472	547	299	1091	773
DCMF	3.82	2.21×10^1	1.31×10^1	6.49	3.18×10^1	1.21×10^1
DMF	3.82	2.20×10^1	1.31×10^1	6.49	3.18×10^1	1.21×10^1
dynaMMo	4.65	3.33×10^1	2.69×10^1	1.97×10^1	4.09×10^1	2.19×10^1
SmoothSoRec	1.31×10^1	2.69×10^1	1.67×10^1	8.23	3.08×10^1	1.38×10^1
SmoothPMF	1.94×10^1	2.24×10^1	1.53×10^1	9.64	3.05×10^1	1.23×10^1
SoRec	1.59×10^1	3.48×10^1	1.80×10^1	1.67×10^1	3.66×10^1	1.68×10^1
PMF	2.06×10^1	2.62×10^1	1.82×10^1	1.22×10^1	3.10×10^1	1.74×10^1



(a) Marker Postions[1]

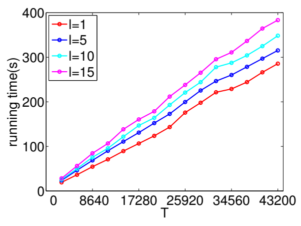


(b) y-coordinate

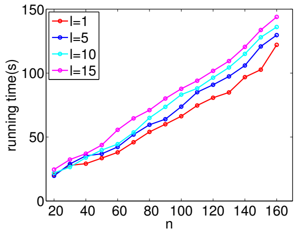


(c) z-coordinate

Figure 5: A running instance on Marker LWRB.



(a) Motes Dataset



(b) Chlorine Dataset

Figure 6: Scalability of DCMF.

DCMF and DMF lead to very good approximations of the missing values. We further examined the latent factor $\mathbf{U}$ derived by DCMF, and found that the three closest markers in terms of the latent vector $\mathbf{U}(i,:)$ are RWRB, RWRA and LWRA. This is consistent with their positions on the human body (See Figure 5(a)).

5.4 Efficiency Results

We test the scalability of the DCMF algorithm on a number of subsets of the Motes dataset and the Chlorine dataset with 10% missing values. As Figure 6(a) shows, our proposed DCMF algorithm scales linearly w.r.t. the sequence length T, which is consistent with our complexity analysis in section 4. Figure 6(b) demonstrates that with 10% missing values the running time of DCMF is close to linear w.r.t. the number of objects n.

[1]http://mocap.cs.cmu.edu/

6 Related Work

We review the related work from two aspects: low-rank matrix factorization and time series mining.

Low Rank Matrix Factorization Matrix factorization methods have been successfully applied in many data mining applications. The recommendation systems are most closely related to our work [15, 13, 10, 19]. These methods aim to find low-rank latent factors to represent users and items. The factors can be fit into the user-item rating matrix and can be further used to make the rating prediction. For example, [15] proposed a linear probabilistic model of matrix factorization, which scales linearly with the number of observations and performs well on large and sparse datasets. [13, 20] integrated the user social network and/or the item-item similarity with the user-item matrix in the probabilistic factor analysis. Recently, dynamic matrix factorization methods have been proposed to handle the user preferences changing over time [5, 19]. [19] built a dynamic state space model upon probabilistic matrix factorization to track the temporal dynamics of the user latent factor. [5] applied non-negative matrix factorization on linear dynamic systems, to model evolving user preferences on the user-item adoption problem.

Time Series Mining There has been a large amount of research work in time series mining [9], including representation [18, 16], classification [21, 6, 7], outlier detection [11] and etc. For instance, [7] proposed a voting framework for multi-dimensional time series classification by weighting the class prediction from each time series stream. [6] used an ensemble of models for mining data streams with concept-drifting and skewed distributions. [12] proposed a linear dynamic system based approach to infer the missing values from multiple time series. [14] combined a multi-level chain model and a cost model to find typical patterns and meaningful segments in multiple time series.

7 Conclusion

In this paper, we introduce a novel setting on mining a network of multiple coevolving time series and propose a Dynamic Contextual Matrix Factorization (DCMF) algorithm to infer the missing values from the input time series. Our method has three key advantages. First, it is *effective*, outperforming all the existing competitors especially when there are lots of missing values. Second, it is *scalable*, enjoying a linear scalability w.r.t. the length of time series. Third, it is *comprehensive*, simultaneously capturing (a) the correlation among different time series; (b) the contextual information encoded by the contextual network; and (c) the temporal smoothness along the time dimension. Future work includes modeling the dynamics of the contextual network as well as the lag correlation among multiple time series.

8 Acknowledgment

This material is supported by the National Science Foundation under Grant No. IIS1017415 and CNS-0904901, by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053, by Defense Advanced Research Projects Agency (DARPA) under Contract Number W911NF-11-C-0200 and W911NF-12-C-0028, by National Institutes of Health under the grant number R01LM011986, Region II University Transportation Center under the project number 49997-33 25.

The content of the information in this document does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

The authors would like thank Dr. Eamonn Keogh for his comments.

References

[1] Epanet-2. `http://www.epa.gov/nrmrl/wswrd/dw/epanet.html`.

[2] Motes dataset. `http://db.csail.mit.edu/labdata/labdata.html`.

[3] Motion dataset. `http://mocap.cs.cmu.edu`.

[4] C. M. Bishop et al. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.

[5] F. C. T. Chua, R. J. Oentaryo, and E.-P. Lim. Modeling temporal adoptions using dynamic matrix factorization. In *ICDM*, pages 91–100. IEEE, 2013.

[6] J. Gao, B. Ding, W. Fan, J. Han, and P. S. Yu. Classifying data streams with skewed class distributions and concept drifts. *IEEE Internet Computing*, 12(6):37–49, 2008.

[7] B. Hu, Y. Chen, J. Zakaria, L. Ulanova, and E. Keogh. Classification of multi-dimensional streaming time series by weighting each classifier's track record. In *ICDM*, pages 281–290. IEEE, 2013.

[8] M. Jiang, P. Cui, R. Liu, Q. Yang, F. Wang, W. Zhu, and S. Yang. Social contextual recommendation. In *CIKM*, pages 45–54, New York, NY, USA, 2012. ACM.

[9] E. Keogh. Tutorial: Machine learning in time series databases (and everything is a time series!). In *AAAI*, 2011.

[10] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[11] J.-G. Lee, J. Han, and X. Li. Trajectory outlier detection: A partition-and-detect framework. In *ICDE*, pages 140–149. IEEE, 2008.

[12] L. Li, J. McCann, N. S. Pollard, and C. Faloutsos. Dynammo: Mining and summarization of coevolving sequences with missing values. In *KDD*, pages 507–516. ACM, 2009.

[13] H. Ma, H. Yang, M. R. Lyu, and I. King. Sorec: social recommendation using probabilistic matrix factorization. In *CIKM*, pages 931–940. ACM, 2008.

[14] Y. Matsubara, Y. Sakurai, and C. Faloutsos. Autoplait: Automatic mining of co-evolving time sequences. In *SIGMOD*, pages 193–204, New York, NY, USA, 2014. ACM.

[15] A. Mnih and R. Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2007.

[16] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, pages 697–708. VLDB Endowment, 2005.

[17] H. E. Rauch, C. Striebel, and F. Tung. Maximum likelihood estimates of linear dynamic systems. *AIAA journal*, 3(8):1445–1450, 1965.

[18] J. Shieh and E. Keogh. isax: indexing and mining terabyte sized time series. In *KDD*, pages 623–631. ACM, 2008.

[19] J. Z. Sun, K. R. Varshney, and K. Subbian. Dynamic matrix factorization: A state space approach. In *ICASSP*, pages 1897–1900. IEEE, 2012.

[20] Y. Yao, H. Tong, G. Yan, F. Xu, X. Zhang, B. Szymanski, and J. Lu. Dual-regularized one-class collaborative filtering. In *CIKM*. ACM, 2014.

[21] L. Ye and E. Keogh. Time series shapelets: a new primitive for data mining. In *KDD*, pages 947–956. ACM, 2009.