NetDyna: Mining Networked Coevolving Time Series with Missing Values

Hairi
Arizona State University
fhairi@asu.edu

Hanghang Tong
University of Illinois at Urbana-Champaign
htong@illinois.edu

Lei Ying
University of Michigan, Ann Arbor
leiying@umich.edu

***Abstract*—This paper presents a novel algorithm for recovering missing values of co-evolving time series with partial embedded network information. The idea is to connect two sources of data (time series data and embedded network data) through a shared low dimensional latent space. The proposed algorithm, named NetDyna, is an Expectation-Maximization (EM) algorithm, and uses the Kalman filter and matrix factorization approaches to infer the missing values both in the time series and embedded network. Our experimental results on real datasets, including a Motes dataset and a Motion Capture dataset, show that (1) NetDyna outperforms other state-of-the-art algorithms, especially with partially observed network information; (2) its computational complexity scales linearly with the time duration of time series; and (3) the algorithm recovers the embedded network in addition to missing time series values.**

***Index Terms*—Co-evolving time series, Missing value recovery, EM algorithm, Kalman filter**

I. Introduction

Co-evolving time series are common in many real world applications such as temperature monitoring in smart buildings, mobile object tracking, and motion capturing for environmental monitoring. These co-evolving time series are generated from a collection of system components (such as sensors) over an extended time period. In many cases, because of reasons such as sensor duty cycling, packet losses in transmissions, and hardware malfunctions, we only observe incomplete time series with many missing values. Recovering missing values is important but challenging problem for many applications, e.g. for determining whether certain chemical level in the drinking water exceeds a threshold.

A novel approach to recover missing values in co-evolving time series is to explore the correlation of different data sources (i.e. the co-evolution) instead of treating time series as independent processes. In particular, an embedded network may be constructed for co-evolving time series. For a wireless sensor network, the embedded network is a graph in which an edge exists between two nearby sensors to indicate the correlation of their measurements. In epilepsy study, measurements collected from different brain regions of a patient can be connected via a brain graph that describes the correlation of brain activities. While exploiting the underlying correlation of time series is proven to be very effective, learning the underlying embedded network itself can be a daunting task. Often, similar to the time series data, we only have partial information of the embedded network.

To tackle the challenges of missing time series data and incomplete embedded network information, we propose **NetDyna**. The main results of this paper are summarized below.

- **Modeling:** We formulate the missing value problem as maximum likelihood problem, which can be solved using the Expectation Maximization (EM) method for recovering time series and embedded network data.
- **Algorithm Design:** We propose an effective algorithm for recovering missing values. Following the idea of latent factors, our algorithm projects both the time series data and network data into a shared lower dimensional latent space. We leverage the Kalman filter and matrix factorization methods in the expectation step to infer the distributions of time series data and embedded network data respectively, and then maximize the parameters iteratively in the maximization step. We analyze the complexity of the algorithm.
- **Evaluation:** We evaluated the performance of **NetDyna** using real datasets under different missing settings. The experimental results showed the effectiveness and efficiency when comparing with the-state-of-the-art algorithms. In particular, **NetDyna** outperforms other algorithms for recovering missing time series when the embedded network is partially known.

The rest of the paper is organized as follows: In Section 2, we review the related work. We formulate the problem in Section 3 and present our proposed algorithm, **NetDyna**, in Section 4. Section 5 presents the experimental results, accuracy, sensitivity and reconstruction errors of **NetDyna**, and the comparisons with existing algorithms. Section 6 concludes the paper.

II. Related Work

Missing value recovery problem is closely related to the low-rank matrix factorization [1], which has been extensively studied for recommendation systems. [2] proposed a probabilistic matrix factorization (PMF) method for user item rating matrix, which scales linearly with the number of observations. PMF performs well on large, sparse and imbalanced Netflix dataset. However, sequential dependence, which is the intrinsic difference of time series data from other data, is not encoded in the PMF. [3] proposed MSVD, a SVD based missing value recovery method, where missing values are initialized using a linear interpolation first. [4] proposed probabilistic matrix

978-1-7281-0858-2/19/$31.00 © 2019 IEEE
978-1-7281-0858-2/19/$31.00 ©2019 IEEE

factorization with the social network information. SoRec considered recovering missing values in user item matrix utilizing a social network, where it's been shown that social network information can help improve the recovery result.

[5] proposed an online algorithm, which discovers the correlation among multiple time series and jointly recover the missing values. [6], [7] provided general frameworks for data mining tasks, including missing value recovery. For recovering missing values in time series, [6] proposed Spirit, a PCA based learning model. [8] proposed DynaMMo, which learns the dynamics of latent variables. By filling missing values using linear interpolation or some other methods, DynaMMo uses Kalman filtering to estimate system parameters. However, there's no network data is used. [9] proposed DCMF, an algorithm that combines partially observed time series data with a fully observed embedded network. In [10], the authors have further developed DCMF into a higher dimension model by considering more than one type of measurements. However, both paper assume complete knowledge of the embedded network. Our paper considers partially observed network information.

Neural network approach has been introduced to time series problem. [11]–[13] proposed recurrent neural network based algorithms. However, they are looking at classification problems with missing values in time series. In this paper, we studied the problem of recovering the missing value itself.

We note that [4], [9], [10] proposed to connect two different data resources via a shared latent feature space, which motivated our algorithm, NetDyna, which use a common latent feature space for both time series and the embedded network. Finally, [14] introduces different types of missing value patterns. In this paper, we considers three different patterns including missing uniformly, missing as a block, and missing entirely. Our experimental results show that our algorithm outperforms other algorithms in most cases under these three different missing patterns.

III. Model and Problem Formulation

In this section, we present our model and formulate the problem of recovering coevolving time series under our model as a maximum likelihood problem.

A. Joint Embedding of Coevolving Time Series

We assume the coevolving time series are associated with a network where each node is the source of a time series and time series are correlated if the two sources are neighbors. Our goal is to recover missing values of coevolving time series with partial time series and partial network information.

Example: Consider a smart house with six rooms as shown in Figure 1, where sensors are placed in each room to track the temperature. A directed edge (i, j) represents the dependency of the temperature of room j on that of room i. The time series data are partially observed, as shown in Table I. Our goal is to recover missing values in time series data and missing edges of the network. □

TABLE I: Room Temperature Data of a Smart House

	t_1	t_2	t_3	t_4	t_5
room 1	72	73			74
room 2		74	73	72	
room 3	71	73			75
room 4	76		73		74
room 5	74	71			
room 6	75	73		72	

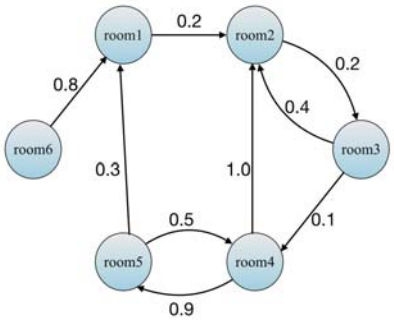


Fig. 1: The Network Associated with Room Temperature Data

Consider coevolving time series with N sources and over T time slots. Each source creates one measurement at each time slot. The coevolving temperature data can be represented by an $N \times T$ matrix $\mathbf{X}$. The weighted adjacency matrix of the network is an $N \times N$ matrix, denoted by $\mathbf{S}$. Note that without imposing any structure on the time series data, missing measurements can be arbitrary values, so it is impossible to recover these missing values. A widely observed structure in real-world datasets is that high-dimensional data can often be embedded into a low-dimensional space, which makes recovery of missing values possible. For example, matrix completion algorithms such as collaborative filtering are often based on this assumption. By leveraging this observation, we propose joint time series and network embedding. In particular, we assume each node (say node i) is associated with an L-dimensional latent vector $\mathbf{U}_i \in \mathbf{R}^L$. We define $\mathbf{U}$ to be an $L \times N$ matrix such that the ith column $\mathbf{U}_i \triangleq \mathbf{U}_{(:,i)}$ is the latent vector of object i. In addition, there is L-dimensional time series $\mathbf{Z}_t \in \mathbf{R}^L$ which evolves as a linear system such that

$$\mathbf{Z}_t = \mathbf{B}\mathbf{Z}_{t-1} + \mathbf{W}_t$$

where $\mathbf{B}$ is an $L \times L$ transition matrix, $\mathbf{Z}_1 \sim \mathcal{N}(z_0, \mathbf{\Psi}_0)$ and $\mathbf{W}_t \sim \mathcal{N}\left(0, \sigma_Z^2 \mathbf{I}\right)$. Note that $\mathbf{Z}_1$ is an L-dimensional Gaussian vector with a general distribution and $\mathbf{W}_t$ is zero mean Gaussian vector with i.i.d. components, representing i.i.d. Gaussian noise.

The time series data $\mathbf{X}_t$ (an N-dimensional vector) is generated based on $\mathbf{U}$ and $\mathbf{Z}_t$ such that

$$\mathbf{X}_t = \mathbf{U}'\mathbf{Z}_t + \boldsymbol{\epsilon}_t,$$

where $\epsilon_t \sim \mathcal{N}\left(0, \sigma_X^2 \mathbf{I}\right)$ is a N-dimensional zero mean Gaussian noise.

The network matrix $\mathbf{S}$ is generated based on $\mathbf{U}$ such that

$$\mathbf{S} = \mathbf{V}'\mathbf{U} + \tau \tag{1}$$

where $\mathbf{V}$ is an $L \times N$ matrix such that the ith column $\mathbf{V}_i \triangleq \mathbf{V}_{(:,i)} \sim \mathcal{N}\left(\mathbf{U}_i, \sigma_V^2\mathbf{I}\right)$ and $\tau_i \triangleq \tau_{(:,i)} \sim \mathcal{N}\left(0, \sigma_S^2\mathbf{I}\right)$.

Figure 2 summarizes the joint embedding model, where $\mathbf{U}$ and $\mathbf{Z}$ are latent variables and $\mathbf{X}$ and $\mathbf{S}$ are observed variables.

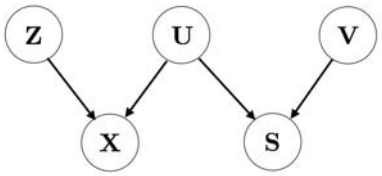


Fig. 2: Time Series and Network Joint Embedding

B. Missing Values Recovery Problem and Maximum Likelihood Formulation

With the model and notation defined above, we introduce the maximum likelihood formulation for tackling the missing value recovery problem defined below. We define an $N \times T$ matrix $\mathbf{M}$ such that $\mathbf{M}_{nt} = 1$ when $\mathbf{X}_{nt}$ is observed and $\mathbf{M}_{nt} = 0$ if $\mathbf{X}_{nt}$ is missing; and define an $N \times N$ matrix $\tilde{\mathbf{M}}$ such that $\tilde{\mathbf{M}}_{ij} = 1$ if the edge weight (i, j) is observed and $\tilde{\mathbf{M}}_{ij} = 0$ otherwise. These two matrices specify the entries of missing values.

Summarizing the model we have introduced, we have

$$\mathbf{Z}_t = \mathbf{B}\mathbf{Z}_{t-1} + \mathbf{W}_t \tag{2}$$

$$\mathbf{X}_t = \mathbf{M}_t \odot (\mathbf{U}'\mathbf{Z}_t + \epsilon_t) \tag{3}$$

$$\mathbf{S} = \tilde{\mathbf{M}} \odot (\mathbf{V}'\mathbf{U} + \tau), \tag{4}$$

where $\mathbf{M}_t = \mathbf{M}_{(:,t)}$ is the tth column of matrix $\mathbf{M}$.

Let $\hat{\mathbf{X}}$ and $\hat{\mathbf{S}}$ denote the observed time series and weighted adjacency matrix, where the missing values are set to be zero. The missing value recovery problem is defined below.

Missing Value Recovery:

Input: Partially observed time series from a networked data sources and a partially observed network, i.e. $\hat{\mathbf{X}}$, $\mathbf{M}$, $\hat{\mathbf{S}}$, $\tilde{\mathbf{M}}$.

Output: Complete time series data and network data $\mathbf{X}$ and $\mathbf{S}$ that match the observed data, i.e. $\mathbf{X} \odot \mathbf{M} = \hat{\mathbf{X}}$ and $\mathbf{S} \odot \tilde{\mathbf{M}} = \hat{\mathbf{S}}$.

Note that the co-evolving time series is a high-dimensional random process defined by (2)-(4), which is characterized by the parameter set

$$\boldsymbol{\theta} = \{\mathbf{U}, \mathbf{B}, \mathbf{z}_0, \boldsymbol{\Psi}_0, \sigma_Z, \sigma_X, \sigma_V, \sigma_S\}.$$

We propose the following maximum likelihood problem for recovering the missing values:

$$\max_{\boldsymbol{X},\boldsymbol{S},\boldsymbol{\theta}} p\left(\mathbf{X}, \mathbf{S} | \boldsymbol{\theta}, \hat{\mathbf{X}}, \hat{\mathbf{S}}\right) \tag{5}$$

Note that the condition $\hat{\mathbf{X}}$ and $\hat{\mathbf{S}}$ mean

$$\mathbf{X} \odot \mathbf{M} = \hat{\mathbf{X}} \quad \text{and} \quad \mathbf{S} \odot \tilde{\mathbf{M}} = \hat{\mathbf{S}}.$$

Table II summarizes the key notations to be used throughout the paper.

TABLE II: Symbols and Definitions

Symbol	Definitions and Descriptions
$\mathbf{A}$	matrix (bold upper case)
$\mathbf{A}_{ij}$	the element at row i and column j of matrix $\mathbf{A}$
$\mathbf{A}(:, j)$	the jth column of matrix $\mathbf{A}$
$\mathbf{A}(i, :)$	the ith row of matrix $\mathbf{A}$
$\mathbf{A}^T$	transpose of matrix $\mathbf{A}$
$\mathbf{a}$	column vector (bold lower case)
$\odot$	Hadamard product
$\mathbf{X}$	time series matrix
$\mathbf{M}$	indicator matrix for time series matrix
$\mathbf{S}$	embedded network matrix
$\tilde{\mathbf{M}}$	indicator matrix for embedded network matrix
$\mathbf{Z}$	time series latent matrix
$\mathbf{V}$	network latent matrix
$\mathbf{B}$	transition matrix
$\mathbf{U}$	object latent matrix
T	time duration
N	number of measurements in system
L	dimension of latent space
$\mathcal{N}(\mathbf{z}\|\mu, \mathbf{\Sigma})$	Gaussian distribution random variable with mean μ and covariance matrix $\mathbf{\Sigma}$

IV. Proposed Algorithm: NetDyna

Since directly maximizing likelihood (5) is intractable [15], [16], we first use the expectation-maximization (EM) algorithm to maximize the evidence lower bound, defined below, to find $\boldsymbol{\theta}$ based on the observed data

$$E_{\mathbf{Z},\mathbf{V}|\boldsymbol{\theta},\hat{\mathbf{X}},\hat{\mathbf{S}}}[\ln p(\hat{\mathbf{X}}, \hat{\mathbf{S}}, \mathbf{Z}, \mathbf{V}|\boldsymbol{\theta})]. \tag{6}$$

After obtaining $\boldsymbol{\theta}$, and distributions of latent variables $\mathbf{Z}$ and $\mathbf{U}$, we then recover the missing time series and network data by solving the maximum likelihood problem.

The EM algorithm first initializes parameter set $\boldsymbol{\theta}$. In the expectation step, we compute the distributions of latent variables $\mathbf{Z}$ and $\mathbf{V}$ conditioned on the current parameter set $\boldsymbol{\theta}$ and observed data and then calculate (6). In the maximization step, we iteratively update the parameter set $\boldsymbol{\theta}$ based on the current distribution of latent variables. The algorithm terminates after the value of the evident lower bound converges.

A. The E-Step

In this step, parameter set $\boldsymbol{\theta}$ is fixed. According to Figure 2, time series data $\mathbf{X}$ and network data $\mathbf{S}$ are conditionally independent given $\mathbf{U}$, so can be inferred separately. In particular, we have

$$E_{\mathbf{Z},\mathbf{V}|\boldsymbol{\theta},\hat{\mathbf{X}},\hat{\mathbf{S}}}[\ln p(\hat{\mathbf{X}}, \hat{\mathbf{S}}, \mathbf{Z}, \mathbf{V}|\boldsymbol{\theta})] \tag{7}$$

$$= E\left[\ln p(\mathbf{Z}_1|\boldsymbol{\theta}) \prod_{t=2}^{T} p(\mathbf{Z}_t|\mathbf{Z}_{t-1}, \boldsymbol{\theta}) \prod_{t=1}^{T} \prod_{i:\mathbf{M}_{it}=1} p(\mathbf{X}_{it}|\mathbf{Z}_t, \boldsymbol{\theta}) \times \prod_{i=1}^{N} p(\mathbf{V}_i|\boldsymbol{\theta}) \prod_{i=1}^{N} \prod_{j:\tilde{\mathbf{M}}_{ij}=1} p(\mathbf{S}_{ij}|\mathbf{V}_i, \boldsymbol{\theta})\right]. \tag{8}$$

To calculate expectation, the first step is to calculate the distributions of latent variables $\mathbf{Z}$ and $\mathbf{V}$ given the parameter set $\boldsymbol{\theta}$ and observed data $\hat{\mathbf{X}}$ and $\hat{\mathbf{S}}$.

1) Distributions of Latent Variables for Time Series Data: Equations (2) and (3) show the dynamic of latent process $\{\mathbf{Z}_t\}_{t=1}^T$ and partially observed $\{\mathbf{X}_t\}_{t=1}^T$. To simplify the notation, for each time slot $t = 1, \cdots, T$, define

$$\begin{aligned}\mathbf{O}_t &= \{i|\mathbf{M}_{it} > 0, i = 1, \cdots, N\}\\ \mathbf{X}_t^* &= \mathbf{X}_t(\mathbf{O}_t)\\ \mathbf{H}_t &= \mathbf{U}'(\mathbf{O}_t, :)\end{aligned} \tag{9}$$

where $\mathbf{O}_t$ is the index set of observed entries at time t, $\mathbf{X}_t^*$ is the observed entries at time t and $\mathbf{H}_t$ is a compressed version of matrix $\mathbf{U}'$ at time t. For example, suppose

$$\mathbf{U}' = \begin{pmatrix} 1 & 0 \\ 0.5 & 0.5 \\ 0 & 1 \end{pmatrix},\ \mathbf{Z}_t = \begin{pmatrix} 0.7 \\ 0.3 \end{pmatrix},$$

$$\mathbf{X}_t = \begin{pmatrix} 0.7 \\ 0.5 \\ 0 \end{pmatrix} \text{ and } \mathbf{M}_t = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}.$$

Then we have

$$\mathbf{O}_t = \{1, 2\},\ \mathbf{X}_t^* = \begin{pmatrix} 0.7 \\ 0.5 \end{pmatrix},\ \text{and } \mathbf{H}_t = \begin{pmatrix} 1 & 0 \\ 0.5 & 0.5 \end{pmatrix}.$$

We can rewrite equation (3) as follows:

$$\mathbf{X}_t^* = \mathbf{H}_t\mathbf{Z}_t + \epsilon_t. \tag{10}$$

Since noises and the initial condition are Gaussian, so are the posteriors of latent variable $\{\mathbf{Z}_t\}_{t=1}^T$. Then, we apply the forward-backward algorithm [16] or Kalman filter and smoother [17], [18] to calculate the posteriors. Define the following two posterior distributions, one conditioned on observations up to time t and the other on all observations:

$$\begin{aligned}p(\mathbf{Z}_t|\mathbf{X}_1^*, \cdots, \mathbf{X}_t^*) &= \mathcal{N}(\mathbf{Z}_t|\mu_t, \mathbf{\Psi}_t)\\ p(\mathbf{Z}_t|\mathbf{X}_1^*, \cdots, \mathbf{X}_T^*) &= \mathcal{N}(\mathbf{Z}_t|\tilde{\mu}_t, \tilde{\mathbf{\Psi}}_t).\end{aligned}$$

By applying the forward algorithm (or the Kalman filter) to estimate μ_t and $\mathbf{\Psi}_t$, we have

$$\begin{aligned}\mathbf{P}_{t-1} &= \mathbf{B}\mathbf{\Psi}_{t-1}\mathbf{B}' + \sigma_Z^2\mathbf{I}\\ \mathbf{K}_t &= \mathbf{P}_{t-1}\mathbf{H}_t'(\mathbf{H}_t\mathbf{P}_{t-1}\mathbf{H}_t' + \sigma_X^2\mathbf{I})^{-1}\\ \mu_t &= \mathbf{B}\mu_{t-1} + \mathbf{K}_t(\mathbf{X}_t^* - \mathbf{H}_t\mathbf{B}\mu_{t-1})\\ \mathbf{\Psi}_t &= (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\mathbf{P}_{t-1}\end{aligned} \tag{11}$$

with initial conditions

$$\begin{aligned}\mathbf{K}_1 &= \mathbf{\Psi}_0\mathbf{H}_1'(\mathbf{H}_1\mathbf{\Psi}_0\mathbf{H}_1' + \sigma_X^2\mathbf{I})^{-1}\\ \mu_1 &= \mathbf{z}_0 + \mathbf{K}_1(\mathbf{X}_1^* - \mathbf{H}_1\mathbf{B}\mathbf{z}_0)\\ \mathbf{\Psi}_1 &= (\mathbf{I} - \mathbf{K}_1\mathbf{H}_1)\mathbf{\Psi}_0.\end{aligned} \tag{12}$$

Then, by applying the backward algorithm (or Kalman smoother), we have

$$\begin{aligned}\mathbf{J}_t &= \mathbf{\Psi}_t\mathbf{B}'(\mathbf{P}_t)^{-1}\\ \tilde{\mu}_t &= \mu_t + \mathbf{J}_t(\tilde{\mu}_{t+1} - \mathbf{B}\mu_t)\\ \tilde{\mathbf{\Psi}}_t &= \mathbf{\Psi}_t + \mathbf{J}_t(\tilde{\mathbf{\Psi}}_t - \mathbf{P}_t)\mathbf{J}_t'.\end{aligned} \tag{13}$$

The expectations are

$$\begin{aligned}E[\mathbf{Z}_t] &= \tilde{\mu}_t\\ E[\mathbf{Z}_t\mathbf{Z}_{t-1}'] &= \tilde{\mathbf{\Psi}}_t\mathbf{J}_{t-1}' + \tilde{\mu}_t\tilde{\mu}_{t-1}'\\ E[\mathbf{Z}_t\mathbf{Z}_t'] &= \tilde{\mathbf{\Psi}}_t + \tilde{\mu}_t\tilde{\mu}_t'\end{aligned} \tag{14}$$

which are needed for parameter update in maximization step.

2) Distributions of Latent Variables for Network Data: Now the posterior distributions of network latent variables can also be similarly derived. Rewrite (4)

$$\mathbf{S}' = \tilde{\mathbf{M}}' \odot (\mathbf{U}'\mathbf{V} + \tau), \tag{15}$$

and define

$$\begin{aligned}\tilde{\mathbf{O}}_j &= \{i|\tilde{\mathbf{M}}_{ij}' = \tilde{\mathbf{M}}_{ji} > 0, i = 1, \cdots, N\}\\ \mathbf{S}_j^* &= \mathbf{S}_j'(\tilde{\mathbf{O}}_j)\\ \mathbf{G}_j &= \mathbf{U}'(\tilde{\mathbf{O}}_j, :)\end{aligned} \tag{16}$$

where $\tilde{\mathbf{O}}_j$ is the index set of observed entries of column $\tilde{\mathbf{M}}_j'$(or row $\tilde{\mathbf{M}}(j, :)$), $\mathbf{S}_j^*$ is the observed entries of column vector $\mathbf{S}_j'$ and $\mathbf{G}_j$ is the compressed version of $\mathbf{U}'$. Further rewrite (4) as follows:

$$\mathbf{S}_j^* = \mathbf{G}_j\mathbf{V}_j + \tau_j$$

Noises for network data are also Gaussian, so are the posteriors of latent variables $\{\mathbf{V}_i\}_{i=1}^N$. Note that we assume that for $\mathbf{V}$, columns are independent of each other. Define

$$p(\mathbf{V}_j|\mathbf{S}_j^*) = \mathcal{N}(\mathbf{V}_j|\boldsymbol{\nu}_j, \boldsymbol{\gamma}_j)$$

By applying Bayes' theorem, we have

$$\begin{aligned}\boldsymbol{\gamma}_j &= (\sigma_V^{-2}\mathbf{I} + \sigma_S^{-2}\mathbf{G}_j'\mathbf{G}_j)^{-1}\\ \boldsymbol{\nu}_j &= \boldsymbol{\gamma}_j(\sigma_S^{-2}\mathbf{G}_j'\mathbf{S}_j^* + \sigma_V^{-2}\mathbf{U}_j).\end{aligned} \tag{17}$$

The expectations are

$$\begin{aligned}E[\mathbf{V}_j] &= \nu_j\\ E[\mathbf{V}_j\mathbf{V}_j'] &= \gamma_j + \nu_j\nu_j'\end{aligned} \tag{18}$$

which are also used in the maximization step.

After obtaining the distributions of latent variables, we can obtain the following approximation for the evidence lower bound. The details are omitted due to the page limit. Note that we add a scalar λ into the approximation which balances the contributions of observed network data and time series data in inferring the latent variable $\mathbf{U}$, which turns out to be important as we will see in the numerical evaluation.

$$\begin{aligned}Q(\boldsymbol{\theta}) \triangleq& E_{\mathbf{Z},\mathbf{V}|\boldsymbol{\theta},\hat{\mathbf{X}},\hat{\mathbf{S}}}[\ln p(\hat{\mathbf{X}}, \hat{\mathbf{S}}, \mathbf{Z}, \mathbf{V}|\boldsymbol{\theta})] \\ =&(1-\lambda)\left(-\frac{1}{2}\operatorname{tr}(\mathbf{\Psi}_0^{-1}(E[\mathbf{Z}_1\mathbf{Z}_1'] - E[\mathbf{Z}_1]\mathbf{z}_0' - \mathbf{z}_0E[\mathbf{Z}_1'] \right.\\ &+ \mathbf{z}_0\mathbf{z}_0')) + \frac{1}{2}\ln|\mathbf{\Psi}_0^{-1}| + \frac{\sigma_Z^{-2}}{2}\sum_{t=2}^{T}(-\operatorname{tr}(E[\mathbf{Z}_t\mathbf{Z}_t'])\\ &+ \operatorname{tr}(\mathbf{B}'E[\mathbf{Z}_t\mathbf{Z}_{t-1}'] + \operatorname{tr}(\mathbf{B}E[\mathbf{Z}_{t-1}\mathbf{Z}_t']))\\ &\left.- \operatorname{tr}(\mathbf{B}'\mathbf{B}E[\mathbf{Z}_{t-1}\mathbf{Z}_{t-1}'])) - \frac{L(T-1)}{2}\ln\sigma_Z^2\right.\end{aligned} \tag{19}$$

$$
\begin{aligned}
&- \frac{1}{2\sigma_Z^2}\sum_{t=1}^{T}((\mathbf{X}_t^*)'\mathbf{X}_t^* - 2(\mathbf{X}_t^*)'\mathbf{H}_t E[\mathbf{Z}_t] \\
&+ \mathrm{tr}(\mathbf{H}_t'\mathbf{H}_t E[\mathbf{Z}_t\mathbf{Z}_t'])) - \frac{1}{2}\sum_{t=1}^{T}\sum_{i=1}^{N}\mathbf{M}_{it}\ln\sigma_S^2\Bigg) \\
&+ \lambda\left(\frac{\sigma_V^{-2}}{2}\sum_{i=1}^{N}(\mathrm{tr}(E[\mathbf{V}_i\mathbf{V}_i']) - 2\mathbf{U}_i'E[\mathbf{V}_i] + \mathbf{U}_i'\mathbf{U}_i)\right. \\
&- \frac{NL}{2}\ln\sigma_V^2 - \frac{1}{2\sigma_S^2}\sum_{i=1}^{N}((\mathbf{S}_i^*)'\mathbf{S}_i^* - 2(\mathbf{S}_i^*)'\mathbf{G}_i E[\mathbf{V}_i] \\
&\left.+ \mathrm{tr}(\mathbf{G}_i'\mathbf{G}_i E[\mathbf{V}_i\mathbf{V}_i'])) - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\tilde{\mathbf{M}}_{ji}\ln\sigma_S^2\right) \\
&+ \mathrm{const}
\end{aligned}
$$

B. The M Step

In the M step, we iteratively update the parameter set

$$\boldsymbol{\theta} = \{\mathbf{U}, \mathbf{B}, \mathbf{z}_0, \boldsymbol{\Psi}_0, \sigma_Z, \sigma_X, \sigma_V, \sigma_S\}$$

by fixing the distributions of the latent variables. In particular, to update their values, we sequentially setting the derivative of the parameters in parameter set $\boldsymbol{\theta} = \{\mathbf{U}, \mathbf{B}, \mathbf{z}_0, \boldsymbol{\Psi}_0, \sigma_Z, \sigma_X, \sigma_V, \sigma_S\}$ to be 0.

In a summary, after the E-step, we update the parameter set $\boldsymbol{\theta}$ with the following update rules:

$$
\begin{aligned}
\mathbf{z}_0^{\text{new}} &= E[\mathbf{Z}_1] \\
\boldsymbol{\Psi}_0^{\text{new}} &= E[\mathbf{Z}_1\mathbf{Z}_1'] - E[\mathbf{Z}_1]E[\mathbf{Z}_1'] \\
\mathbf{B}^{\text{new}} &= (\sum_{t=2}^{T}E[\mathbf{Z}_t\mathbf{Z}_{t-1}'])(\sum_{t=2}^{T}E[\mathbf{Z}_{t-1}\mathbf{Z}_{t-1}'])^{-1} \\
(\sigma_Z^2)^{\text{new}} &= \frac{1}{L(T-1)}\sum_{t=2}^{T}\mathrm{tr}(E[\mathbf{Z}_t\mathbf{Z}_t'] - E[\mathbf{Z}_t\mathbf{Z}_{t-1}'](\mathbf{B}^{\text{new}})' \\
&\quad - \mathbf{B}^{\text{new}}E[\mathbf{Z}_t\mathbf{Z}_{t-1}'] + \mathbf{B}^{\text{new}}E[\mathbf{Z}_{t-1}\mathbf{Z}_{t-1}'](\mathbf{B}^{\text{new}})') \\
(\sigma_V^2)^{\text{new}} &= \frac{1}{NL}\sum_{i=1}^{N}(\mathrm{tr}(E[\mathbf{V}_i\mathbf{V}_i']) - 2\mathbf{U}_i'E[\mathbf{V}_i] + \mathbf{U}_i'\mathbf{U}_i) \\
\mathbf{U}_i^{\text{new}} &= \mathbf{A}_1\mathbf{A}_2^{-1} \\
\mathbf{A}_1 &= \frac{\lambda}{\sigma_S^2}\sum_{j=1}^{N}\tilde{\mathbf{M}}_{ji}\mathbf{S}_{ji}E[\mathbf{V}_j]' + \frac{(1-\lambda)}{\sigma_X^2}\sum_{t=1}^{T}\mathbf{M}_{it}\mathbf{X}_{it}E[\mathbf{Z}_t]' \\
&\quad + \frac{\lambda}{(\sigma_V^2)^{\text{new}}}E[\mathbf{V}_i]' \\
\mathbf{A}_2 &= \frac{\lambda}{\sigma_S^2}\sum_{j=1}^{N}\tilde{\mathbf{M}}_{ji}E[\mathbf{V}_j\mathbf{V}_j'] + \frac{(1-\lambda)}{\sigma_X^2}\sum_{t=1}^{T}\mathbf{M}_{it}E[\mathbf{Z}_t\mathbf{Z}_t'] \\
&\quad + \frac{\lambda}{(\sigma_V^2)^{\text{new}}}\mathbf{I} \\
(\sigma_X^2)^{\text{new}} &= \frac{1}{\sum_{t=1}^{T}\sum_{i=1}^{N}\mathbf{M}_{it}}\sum_{t=1}^{T}((\mathbf{X}_t^*)'\mathbf{X}_t^* - 2(\mathbf{X}_t^*)'\mathbf{H}_t^{\text{new}}E[\mathbf{Z}_t] \\
&\quad + \mathrm{tr}(\mathbf{H}_t^{\text{new}}E[\mathbf{Z}_t\mathbf{Z}_t'](\mathbf{H}_t^{\text{new}})'))
\end{aligned}
$$

Algorithm 1: NetDyna

Input : Data set $\{\mathbf{X}, \mathbf{M}, \mathbf{S}, \tilde{\mathbf{M}}\}$, latent dimension L and network weight λ
Output: Parameter set $\boldsymbol{\theta}$ and estimated $\mathbf{X}, \mathbf{S}$

repeat
  for $t = 1 : T$ **do**
    Construct $\mathbf{H}_t$ based on Eq.(9);
    Estimate $\mu_t, \boldsymbol{\Psi}_t$ based on Eq.(11) and Eq.(12);
  end
  for $t = T : 1$ **do**
    Estimate $\tilde{\mu}_t, \tilde{\boldsymbol{\Psi}}_t$ based on Eq.(13);
    Estimate $E[\mathbf{Z}_t], E[\mathbf{Z}_t\mathbf{Z}_{t-1}'], E[\mathbf{Z}_t\mathbf{Z}_t']$ based on Eq.(14);
  end
  for $j = 1 : N$ **do**
    Construct $\mathbf{G}_j$ based on Eq.(16);
    Estimate ν_j, γ_j based on Eq.(17);
    Estimate $E[\mathbf{v}_j], E[\mathbf{v}_j\mathbf{v}_j']$ based on Eq.(18);
  end
  Update parameter set $\boldsymbol{\theta}$ based on Eq.(20);
until *converge*;
Set $\mathbf{Z} = (\tilde{\mu}_1, \cdots, \tilde{\mu}_T)$ and $\mathbf{V} = (\nu_1, \cdots, \nu_N)$;
Reconstruct $\mathbf{X} = \mathbf{M} \odot \hat{\mathbf{X}} + (\mathbb{1}_{N\times T} - \mathbf{M}) \odot \mathbf{U}'\mathbf{Z}$ and $\mathbf{S} = \tilde{\mathbf{M}} \odot \hat{\mathbf{S}} + (\mathbb{1}_{N\times N} - \tilde{\mathbf{M}}) \odot \mathbf{V}'\mathbf{U}$;

$$
\begin{aligned}
(\sigma_S^2)^{\text{new}} &= \frac{1}{\sum_{j=1}^{N}\sum_{i=1}^{N}\tilde{\mathbf{M}}_{ij}}\sum_{j=1}^{N}((\mathbf{S}_j^*)'\mathbf{S}_j^* - 2(\mathbf{S}_j^*)'\mathbf{G}_j^{\text{new}}E[\mathbf{V}_j] \\
&\quad + \mathrm{tr}(\mathbf{G}_j^{\text{new}}E[\mathbf{V}_j\mathbf{V}_j'](\mathbf{G}_j^{\text{new}})')) \qquad (20)
\end{aligned}
$$

The new parameter set

$$\boldsymbol{\theta} = \{\mathbf{U}^{\text{new}}, \mathbf{B}^{\text{new}}, \mathbf{z}_0^{\text{new}}, \boldsymbol{\Psi}_0^{\text{new}}, \sigma_Z^{\text{new}}, \sigma_X^{\text{new}}, \sigma_V^{\text{new}}, \sigma_S^{\text{new}}\}$$

will be used in the next E-step.

C. Recovering Missing Values

After the EM algorithm converges, we set $\mathbf{Z} = (\tilde{\mu}_1, \cdots, \tilde{\mu}_T)$ and $\mathbf{V} = (\nu_1, \cdots, \nu_N)$. Then, we recover the missing values by setting

$$
\begin{aligned}
\mathbf{X} &= \mathbf{M} \odot \hat{\mathbf{X}} + (\mathbb{1}_{N\times T} - \mathbf{M}) \odot \mathbf{U}'\mathbf{Z} \\
\mathbf{S} &= \tilde{\mathbf{M}} \odot \hat{\mathbf{S}} + (\mathbb{1}_{N\times N} - \tilde{\mathbf{M}}) \odot \mathbf{V}'\mathbf{U}.
\end{aligned}
$$

Note that these recovery rules are obtained based on the fact that the means are the maximum likelihood solutions given $\mathbf{Z}$, $\mathbf{V}$ and $\mathbf{U}$.

D. NetDyna

The proposed algorithm is summarized in Algorithm 1 named **NetDyna**. And the following two theorems summarize the time complexity and the memory complexity of the algorithm.

Theorem 1: The time complexity of **NetDyna** is $O(\#\text{iteration} \cdot (TL^3 + NL^3 + \sum_t(n_tL^2 + n_t^2L + n_t^3) + \sum_j(n_j^2L + n_jL^2))$.

Here, n_t denotes the number of observed entries in $\mathbf{X}_t$, i.e. cardinality of $\mathbf{O}_t$, and n_j denotes the number of observed entries in $\mathbf{S}(j,:)$, i.e. cardinality of $\tilde{\mathbf{O}}_j$.

Proof 1: The overall time complexity is composed of 4 parts, computing time series data in E step, computing network data in E step, updating parameter set in M step and computing the evidence lower bound.

For each iteration, the complexity for time series in E step is $O(L^3T + \sum_t L^2 n_t + Ln_t^2 + n_t^3)$, including computation for forward algorithm, backward algorithm and expectations; the complexity for network data is $O(L^3N + L^2 \sum_j n_j)$, including computation for inferring and expectations; the complexity for updating parameter set is $O(TL^3 + NL^3 + L^2 \sum_j n_j + L^2 \sum_t n_t + L \sum_t n_t^2 + L \sum_j n_j^2)$; the complexity for computing evidence lower bound is $O(L^3 + L^2T + L \sum_t n_t + NL + L \sum_j n_j)$. So, for each iteration, the complexity is $O(TL^3 + NL^3 + \sum_t (n_t L^2 + n_t^2 L + n_t^3) + \sum_j (n_j^2 L + n_j L^2))$. Thus, the overall complexity is the result in the theorem.

Theorem 2: The memory complexity of **NetDyna** is $O(NT + L^2T + N^2 + L^2N)$.

Proof 2: The space complexity is composed of 4 parts, storing input dataset, parameter set, intermediate values in E step and loglikelihood values. For input dataset, the space complexity is $O(NT + N^2)$; for parameter set, the complexity is $O(LN)$; for intermediate values in E step, the complexity is $O(L^2T + L^2N)$; for loglikelihood, the complexity is $O(1)$. Thus, overall space complexity is $O(NT + L^2T + N^2 + L^2N)$.

V. Experimental Results

This section presents a comprehensive experimental evaluation of **NetDyna**, in terms of its effectiveness, sensitivity and efficiency, with two real datasets; and its comparison with five existing algorithms.

A. Experimental Setup

To evaluate the reconstruction performance of **NetDyna**, we use the root mean squared error(RMSE) for both time series data and network data

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

$$\text{RMSE}_{\text{net}} = \sqrt{\frac{\sum_{i,j}(1 - \tilde{\mathbf{M}}_{ij})(\mathbf{S}_{ij} - \hat{\mathbf{S}}_{ij})^2}{\sum_{i,j}(1 - \tilde{\mathbf{M}}_{ij})}}$$

where $\hat{\mathbf{X}}_{it}, \hat{\mathbf{S}}_{ij}$ are observed values and $\mathbf{X}_{it}, \mathbf{S}_{ij}$ are reconstructed values. Note that we will divide the dataset into training and test parts, the metrics are on test data.

1) Motes Dataset: The Motes dataset[1] consists of temperature measurements from 54 sensors deployed at the Intel Berkeley Research Lab over a month. The temperature measurements are the time series data. The dataset also contains locations of the sensors and the connectivity probabilities among sensors. So using these information, we define each entry of the network matrix as following:

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

where d_{ij} is the Euclidian distance between sensor i and sensor j, c_{ij} is the connectivity probability between sensor i and sensor j, and α is weight control of the two parts. In the experiment, we set $\alpha = 0.5$. In all the Motes dataset related simulations, we use data from all 54 sensors and time slots from 1 to 2880. Note that 2880 time slots is roughly duration of a whole day.

2) Motion Capture Dataset: Motion Capture dataset[2] consists of body movement measurements from markers placed on human body. We used the Mawashi Geri data, a "spin kick" martial art movement, in which 41 markers have been used and there are 1472 frames. Each marker has 3-dimensional coordinates so we have 123 features. Using these data, we define each entry of network matrix as following:

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

where d_{ij} is the Euclidian distance between two markers, while $\alpha = 1$ if i and j are the same coordinate, otherwise $\alpha = 0.5$.

Note that we have standardized, i.e. subtracted the mean and divided by the standard deviation, both datasets before applying **NetDyna** and other state-of-the-art algorithms.

We consider three types of synthetic missing value patterns for time series data: missing as a block fashion, missing uniformly and missing entirely. For the network data, we assume each entry is missing uniformly at random.

- Missing as a block, also called occlusion in [8]: We randomly pick a sensor j and a starting time slot t for this marker. Then, we choose the duration of the missing block for the selected sensor and starting time as a Poisson distributed random variable according to the observed statistics. We repeat this step until the percentage of missing values reaches the threshold we set.
- Missing uniformly: For each observed value, we uniformly at random remove it according to a pre-selected probability.
- Missing entirely: Randomly choose one of the time series and remove it entirely.

B. Effectiveness

For effectiveness, we compared **NetDyna** with the following algorithms.

1) Dynamical Contextual Matrix Factorization(DCMF) [9]: It is a time series data mining algorithm with fully observed embedded network information. DCMF infers the missing values based on both observed values in time series data and complete network data. However,

[1] http://db.csail.mit.edu/labdata/labdata.html

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

the fully observed network information is based only a zero mean prior, which is lack of accountability for network information.

2) Dynamical Matrix Factorization(DMF) [9]: DMF is also a special case of DCMF, where missing values are inferred solely based on observed time series data. If we set $\lambda = 0$, **NetDyna** becomes DMF. We omit the result of that when comparing the algorithms.
3) DynaMMo [8]: It is a time series data mining algorithm with no network information. DynaMMo utilizes only time series data $\mathbf{X}$, where missing values are filled by interpolation or other methods first. Then, learn the system parameter by maximizing the likelihood of time series data, which include both observed data and interpolated data. Note that interpolated data are also used in the objective; however, in a missing as a block setting, where missing values are evolving far from linear interpolation, the learning process will be misled.
4) Missing value Singular Value Decomposition(MSVD) [3]: It combines the idea of SVD with interpolation. The method first uses linear interpolation to fill the missing values, then iteratively applies SVD to the time series matrix and updates the missing values accordingly. Again, like DynaMMo, MSVD will also be potentially misled by interpolated data.
5) Probabilistic Matrix Factorization(PMF) [2]: PMF is a collaborative filtering algorithm, where it learns the latent variables by maximizing a posteriori of the observed values. However, PMF does not consider the dynamics of time series evolution.

Throughout the simulations, we set $L = 15$ for all algorithms for both Motes dataset and Motion Capture dataset.

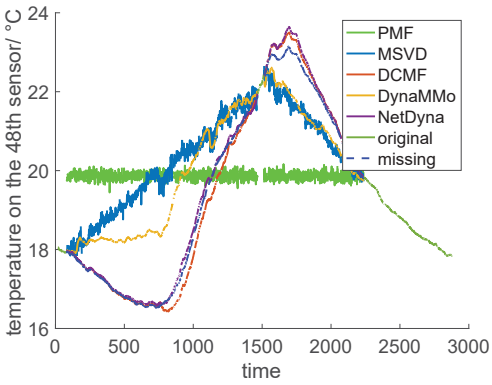


Fig. 3: quantative results for Motes dataset

Figure 3 shows the results of recovering missing time series values under the algorithms mentioned above and **NetDyna**. We selected the 48th sensor in the Motes dataset, two consecutive parts are missing, whose true values are denoted by blue dashed lines; three consecutive parts are observed, which are denoted by green lines. In this case, all algorithms are trying to recover the missing values and the algorithm that tracks blue dashed curve the closest is the best. PMF, MSVD, DCMF, DynaMMo and **NetDyna** are denoted by shiny green, blue, red, yellow and purple curves respectively. As we can see that in the first missing parts, **NetDyna** i.e. the purple curve tracks the missing values the best. Although in the second missing parts DCMF is very close to **NetDyna**, the overall performance of **NetDyna** beats all state-of-the-art algorithms.

TABLE III: Root-Mean-Square-Error (RMSE) with the Motes dataset under missing as a block

Missing	NetDyna	DCMF	DynaMMo	PMF	MSVD
(10%,0%)	0.1581	0.1604	0.4190	0.9118	0.6000
(10%,10%)	0.1628	0.2603			
(10%,20%)	0.1487	0.2696			
(50%,0%)	0.3191	0.3173	0.6599	0.9736	0.7735
(50%,10%)	0.3186	0.3765			
(50%,20%)	0.3082	0.4417			
(90%,0%)	0.6420	0.7238	0.8990	1.0284	0.9834
(90%,10%)	0.6198	0.6722			
(90%,20%)	0.6108	0.7239			

TABLE IV: Root-Mean-Square-Error (RMSE) with the Motes dataset under missing uniformly random

Missing	NetDyna	DCMF	DynaMMo	PMF	MSVD
(10%,0%)	0.0135	0.0133	0.0140	0.0436	0.0257
(10%,10%)	0.0134	0.0134			
(10%,20%)	0.0135	0.0133			
(50%,0%)	0.0141	0.0145	0.0490	0.0614	0.0237
(50%,10%)	0.0142	0.0148			
(50%,20%)	0.0142	0.0145			
(90%,0%)	0.0226	0.0243	0.1684	0.4160	0.0214
(90%,10%)	0.0234	0.0236			
(90%,20%)	0.0226	0.0237			

From Table III to Table VII, we will show the quantitative comparisons of time series data error of different algorithms with different missing patterns on both datasets.

The quantitative comparisons with the state-of-the-art algorithms are shown in Table III and Table IV under different missing percentages for the Motes dataset. In the missing column, the first number is the missing percentage of the time series data and the second number is the missing percentage of network data. Note that the DCMF algorithm also considers network information, however it only allows complete network information. Therefore, for partially observed network information, we set network value to be 0, i.e. no connection, for the unobserved entries.

Under the missing as a block model with different missing percentage settings, **NetDyna** outperforms other algorithms almost in all cases. We note that with fully observed network information, DCMF performs closely to **NetDyna**; however, with partially observed network, **NetDyna** outperforms significantly. Also, under different percentages of partially observed network information, the RMSEs are similar to that of completely observed network information. Under the missing uniformly model, **NetDyna** slightly outperforms DCMF and outperforms other algorithms in almost all cases by a larger margin.

Table V and Table VI summarize the results using the Motion Capture dataset. Under missing as a block setting,

TABLE V: Root-Mean-Square-Error (RMSE) with the Motion Capture dataset under missing as a block

Missing	NetDyna	DCMF	DynaMMo	PMF	MSVD
(10%,0%)	0.0486	0.0487	0.0676	0.8291	0.2350
(10%,10%)	0.0476	0.0626			
(10%,20%)	0.0484	0.0838			
(30%,0%)	0.0808	0.1035	0.1591	0.8443	0.2661
(30%,10%)	0.0823	0.1206			
(30%,20%)	0.0867	0.1072			
(50%,0%)	0.1991	0.2520	0.1854	0.8792	0.2797
(50%,10%)	0.1813	0.2562			
(50%,20%)	0.1750	0.2654			
(90%,0%)	0.8720	0.9981	0.3358	0.9618	0.3092
(90%,10%)	0.7780	0.7241			

TABLE VI: Root-Mean-Square-Error (RMSE) with the Motion Capture dataset under missing uniformly random

Missing	NetDyna	DCMF	DynaMMo	PMF	MSVD
(10%,0%)	0.0139	0.0139	0.0139	0.9995	0.0152
(10%,10%)	0.0139	0.0139			
(10%,20%)	0.0140	0.0139			
(50%,0%)	0.0142	0.0142	0.0145	0.9998	0.0177
(50%,10%)	0.0142	0.0142			
(50%,20%)	0.0142	0.0142			
(90%,0%)	0.0267	0.0297	0.1193	1.0010	0.0280
(90%,10%)	0.0268	0.0299			
(90%,20%)	0.0260	0.0299			

NetDyna outperforms all other algorithms up to missing 50% missing time series data. With sparse Motion Capture data such as when 90% time series data are missing, **NetDyna** is not as accurate as DynaMMo and MSVD. Under the missing uniformly model, **NetDyna** outperforms other algorithms.

Table VII shows the results when one time series is completely missing and the number in the missing column denotes the percentage of missing network data. In this case, DynaMMo, MSVD and PMF algorithms are not defined. To have more comparisons, we introduced two more heuristic baselines.

- heuristic average: Recover the missing time series as the average of other time series whose sensors are connected with the missing sensor.
- heuristic weighted average: Recover the missing time series as the weighted average of other time series whose sensors are connected with the missing sensor.

As a result, **NetDyna** outperforms all the algorithms in the Motes dataset under all settings; with the Motion Capture dataset, **NetDyna** also outperforms in most of the cases, especially with partially observed network.

C. Sensitivity Results

The experimental studies in section focus on the performance of **NetDyna** with different network weights and different levels of network sparsity.

1) Network Weight: We will first show that considering network data helps the recover missing time series data. Besides, the impact of network weight on **NetDyna** will be seen.

TABLE VII: Root-Mean-Square-Error (RMSE) with the Motes and Motion Capture datasets under one time series is entirely missing

RMSE	NetDyna	DCMF	Average	Weighted Average
Motes(0%)	0.1133	0.1375	0.3603	0.3354
Motes(10%)	0.1154	0.1356	0.3637	0.3407
Motes(20%)	0.1155	0.2805	0.3627	0.3311
Motion(0%)	0.2248	0.2216	0.9287	0.7113
Motion(10%)	0.2185	0.2351	0.9247	0.7106
Motion(20%)	0.2183	0.3061	0.9386	0.7152

Figure 4 shows the performance with different network weights under the missing as a block model, missing uniformly model and missing entirely model, respectively, for the Motes dataset. Here we set network data to be fully observed for all these missing patterns. From the results, we can see that considering network information, i.e. $\lambda > 0$, in general improve the accuracy compared with only considering time series data, i.e. $\lambda = 0$.

Under the missing as a block mode, the best λ is between $(0.97, 0.99)$ in our experiments. Under the missing uniformly model, utilizing network information, i.e. $\lambda \in (0, 1]$, helps improve the performance significantly when time series data is sparse, e.g. when 90% are missing as shown in Figure 4b. Under the missing entirely model, we can see that considering network data significantly improves the recovery results.We have similar results for Motion Capture dataset as well in Figure 5.

2) Network Sparsity: The experiment results focus on using partially observed network data for recovering missing values of time series data.

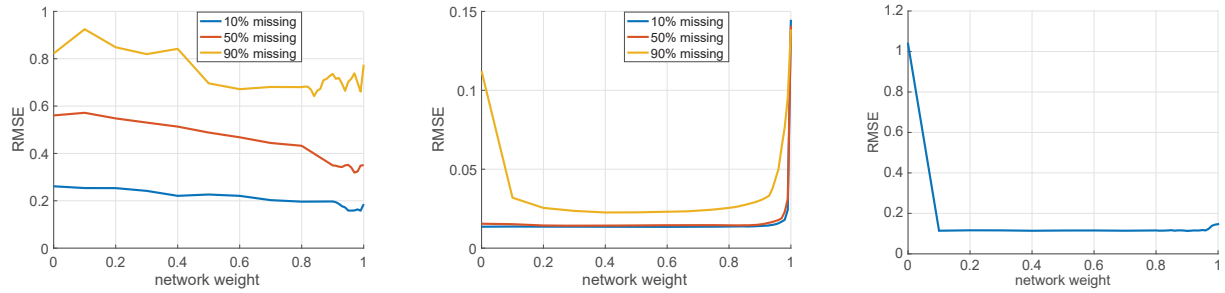
Figure 6 shows the performance with different amounts of network data under the missing as a block model, missing uniformly model and missing entirely model, respectively, for the Motes dataset.

From Figure 6a, we can see that even missing up to 30% of the network information, **NetDyna** performs closely to that of with fully observed network information. The best empirical network weights are in the interval $(0.9, 1)$.

For the uniformly missing case, we can observe from Figure 4b that network data is more helpful in the sparse time series data case, thus the result shown is when 90% time series data are missing. Similarly, with partially observed network information, the performance of **NetDyna** can also be enhanced. The best empirical network weight is in interval $(0.4, 0.6)$. For the entirely missing model, Figure 4c shows that with partially observed network information, the recovery results are as good as that of full network information. For the Motion Capture dataset, we have similar results as shown in Figure 7.

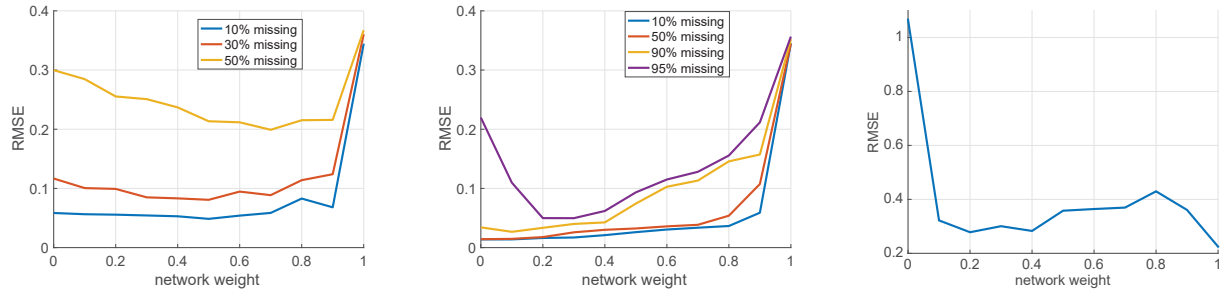
D. Network Data Recovery

In addition to recovering time series data, **NetDyna** can also recover the embedded network. In the presence of missing values in network information, it is inferred from observed network data and observed time series data. With 95% of time



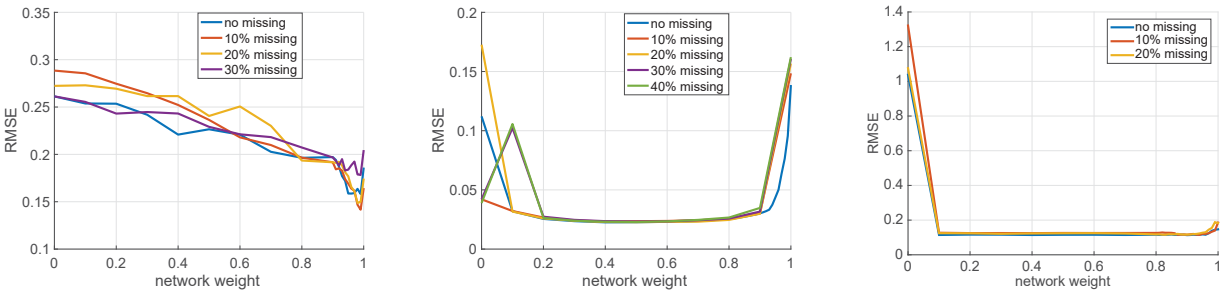
(a) under missing as a block fashion (b) under missing uniformly fashion (c) under missing entirely fashion

Fig. 4: The impact of the network weight on the performance with the Motes dataset



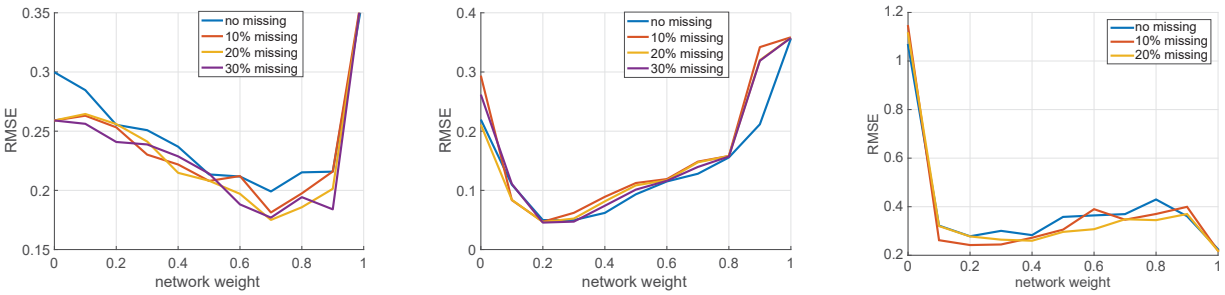
(a) under missing as a block fashion (b) under missing uniformly fashion (c) under missing entirely fashion

Fig. 5: The impact of the network weight on the performance with the Motion Capture dataset



(a) under missing as a block fashion (b) under missing uniformly fashion (c) under missing entirely fashion

Fig. 6: The impact of sparsity of network information on the performance with the Motes dataset



(a) under missing as a block fashion (b) under missing uniformly fashion (c) under missing entirely fashion

Fig. 7: The impact of sparsity of network information on the performance with the Motion Capture dataset

series data missing, we vary the sparsity of network data for Motes dataset. The reconstruction error for network data is shown in Figure 8. For network data recovery, the accuracy of recovering network data can increase when the network weight increases as shown in the figure.

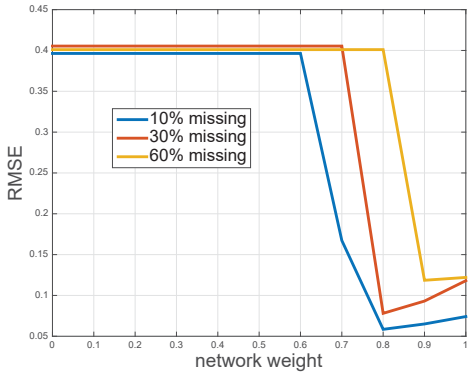


Fig. 8: network data sparsity on network data recovery

E. Efficiency Results

As we discussed in Section 4 that the complexity of **NetDyna** is $O(\#\text{iteration} \cdot (TL^3 + NL^3 + \sum_t (n_t L^2 + n_t^2 L + n_t^3) + \sum_j (n_j^2 L + n_j L^2))$. Figure 9 shows the running time of the algorithm on the Motes dataset versus the sequence length under missing as a block setting with different time series missing percentages and fully observed network information. We can see that the running time is almost linear to the sequence duration; and the algorithm is faster with sparser time series data.

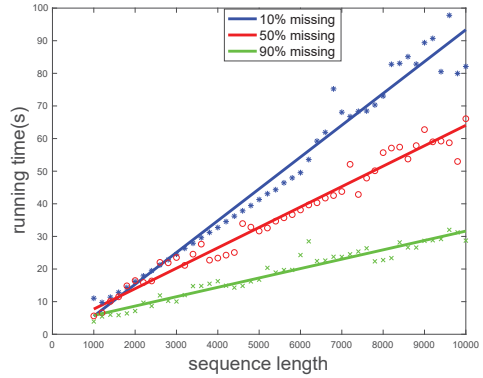


Fig. 9: running time versus the sequence length

VI. Conclusion

In this paper, we introduced **NetDyna** to recover the missing values both in co-evolving time series data and embedded network. With the idea that projecting both data sources onto a shared low dimensional space, we leveraged Kalman filter and matrix factorization based techniques to derive our algorithm. By optimizing the evidence lower bound, **NetDyna** infer the missing values through learning the observed data from both sources.

The real dataset experiments show that **NetDyna** can achieve high accuracy in a robust setting. The sensitivity results also show that (1) **NetDyna** has effectively utilized even partially observed network information and outperforms other state-of-the-art algorithms.s; (2) even with incomplete network data and the whole time series for a node is missing, **NetDyna** is able to recover it with higher accuracy than state-of-the-art algorithms; (3) **NetDyna** scales linearly w.r.t. time duration T.

VII. Acknowledgement

The work is supported in part by NSF grants CNS 1618768, CNS 1813392, IIS 1715385 and IIS 1947135.

References

[1] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, 2009.
[2] A. Mnih and R. R. Salakhutdinov, "Probabilistic matrix factorization," in *Advances in neural information processing systems*, 2008, pp. 1257–1264.
[3] N. Srebro and T. Jaakkola, "Weighted low-rank approximations." ICML, 2003.
[4] H. Ma, H. Yang, M. R. Lyu, and I. King, "Sorec: social recommendation using probabilistic matrix factorization," in *Proceedings of the 17th ACM conference on Information and knowledge management*. ACM, 2008, pp. 931–940.
[5] B.-K. Yi, N. D. Sidiropoulos, T. Johnson, H. Jagadish, C. Faloutsos, and A. Biliris, "Online data mining for co-evolving time sequences," in *Data Engineering, 2000. Proceedings. 16th International Conference on*. IEEE, 2000, pp. 13–22.
[6] S. Papadimitriou, J. Sun, and C. Faloutsos, "Streaming pattern discovery in multiple time-series," in *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 2005, pp. 697–708.
[7] L. Li, "Fast algorithms for mining co-evolving time series," Ph.D. dissertation, Carnegie Mellon University, 2011.
[8] L. Li, J. McCann, N. S. Pollard, and C. Faloutsos, "Dynammo: Mining and summarization of coevolving sequences with missing values," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 507–516.
[9] Y. Cai, H. Tong, W. Fan, and P. Ji, "Fast mining of a network of coevolving time series," in *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 2015, pp. 298–306.
[10] Y. Cai, H. Tong, W. Fan, P. Ji, and Q. He, "Facets: Fast comprehensive mining of coevolving high-order time series," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 79–88.
[11] Z. C. Lipton, D. C. Kale, and R. Wetzel, "Modeling missing data in clinical time series with rnns," *Machine Learning for Healthcare*, 2016.
[12] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *Scientific reports*, vol. 8, no. 1, p. 6085, 2018.
[13] W. Cao, D. Wang, J. Li, H. Zhou, L. Li, and Y. Li, "Brits: Bidirectional recurrent imputation for time series," in *Advances in Neural Information Processing Systems*, 2018, pp. 6775–6785.
[14] R. J. Little and D. B. Rubin, *Statistical analysis with missing data*. John Wiley & Sons, 2014, vol. 333.
[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press Cambridge, 2016, vol. 1.
[16] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.
[17] A. Gelb, *Applied optimal estimation*. MIT press, 1974.
[18] M. Y. Byron, K. V. Shenoy, and M. Sahani, "Derivation of kalman filtering and smoothing equations," 2004.