BiCycle: Item Recommendation with Life Cycles

Xinyue Liu*, Yuanfang Song†, Charu Aggarwal‡, Yao Zhang§ and Xiangnan Kong*
*Worcester Polytechnic Institute, Worcester, MA, USA
†University of Wisconsin-Madison, Madison, WI, USA
‡IBM T.J. Watson Research Center, Yorktown Heights, NY, USA
§Fudan University, Shanghai, China
xliu4@wpi.edu, ysong243@wisc.edu, charu@us.ibm.com, zhang_yao15@fudan.edu.cn, xkong@wpi.edu

***Abstract*—Recommender systems have attracted much attention in last decades, which can help the users explore new items in many applications. As a popular technique in recommender systems, item recommendation works by recommending items to users based on their historical interactions. Conventional item recommendation methods usually assume that users and items are stationary, which is not always the case in real-world applications. Many time-aware item recommendation models have been proposed to take the temporal effects into the considerations based on the absolute time stamps associated with observed interactions. We show that using absolute time to model temporal effects can be limited in some circumstances. In this work, we propose to model the temporal dynamics of both users and items in item recommendation based on their *life cycles*. This problem is very challenging to solve since the users and items can co-evolve in their life cycles and the sparseness of the data become more severe when we consider the life cycles of both users and items. A novel time-aware item recommendation model called BiCycle is proposed to address these challenges. BiCycle is designed based on two important observations: 1) correlated users or items usually share similar patterns in the similar stages of their life cycles. 2) user preferences and item characters can evolve gradually over different stages of their life cycles. Extensive experiments conducted on three real-world datasets demonstrate the proposed approach can significantly improve the performance of recommendation tasks by considering the inner life cycles of both users and items.**

***Index Terms*—Item Recommendation; Collaborative Filtering; Life Cycle; Evolution Inference; Data Mining**

I. Introduction

Collaborative filtering (CF) [1]–[8] is widely used technique for *item recommendation*, which recommends items to users based on their historical interactions, such as ratings, reviews, transactions. Conventional CF methods usually assume that the preferences of users and the characteristics of items are stationary, as shown in Fig. 2a. However, this is not always the case in the real-world systems. First of all, the interests of users may evolve over time. People may like cartoon movies during their childhood and early teen years, but may not like them after they enter college. Moreover, even the characters of items are more stable than user interests, they can subject to change due to various external factors. For instance, in movie recommendation system, the popularity of a movie may exhibit a pattern resembles the well-known hype curve [9], in which the popularity increases rapidly after release until it reaches the peak, then it decreases to its lowest point gradually, at last it bounds up to the average level and becomes relatively stable over time, as shown in Fig. 1.

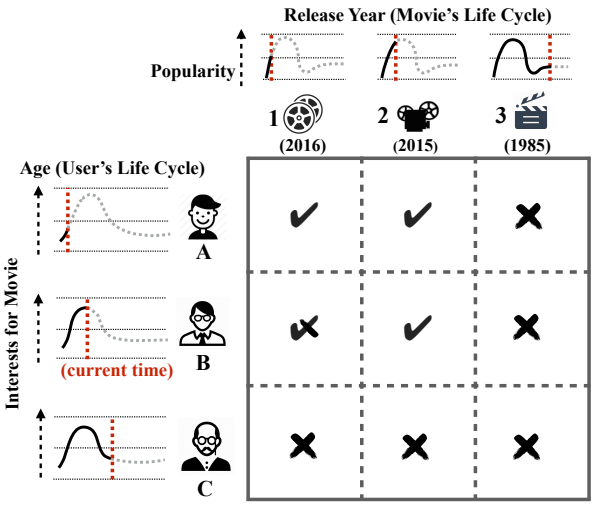
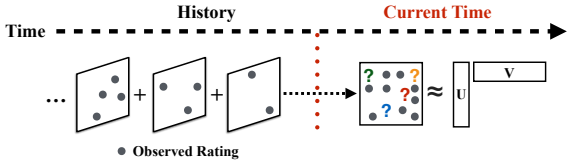
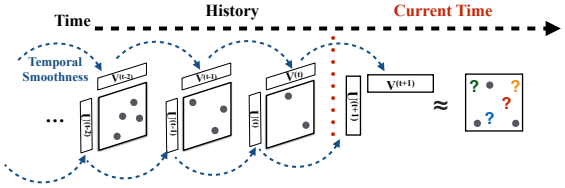


Fig. 1: An illustration of item recommendation with user and item life cycles. The red dotted lines denote the current time in the absolute time-line.

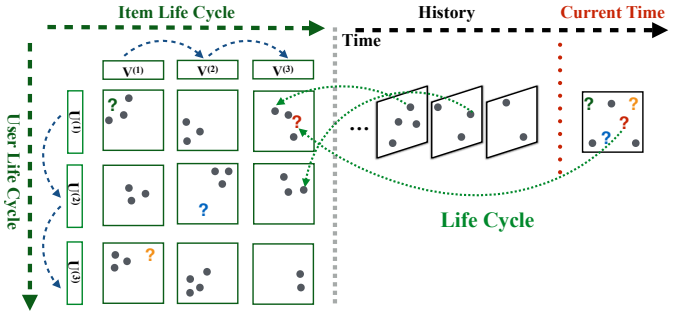
Given its significance, the modeling of temporal effects of collaborative filtering has been extensively studied. Remarkably, the latest progress in the Netflix Prize contest is attributed to a temporal model [3]. Some other works have been proposed in this direction [6]–[8], in which the researchers show the superiority of incorporating temporal effects into the CF models. Conventional methods for temporal CF mainly focus on exploiting the change of user's interests or item characters over the *absolute* time-line, as illustrated in Fig 2b. However, in many real-world applications, users start their involvement at different time and items are released at different time, which makes it challenging to learn a unified time-aware model for the system. Furthermore, the users and items are usually following their internal evolving patterns, which are called *life cycles*. They can exhibit similar preferences or characters in similar stages of their life cycles. We illustrate a toy example of movie recommendation systems in Fig. 1, in which three users are at different stages of their life cycles. All of them share similar evolving patterns with respect to their life cycles. The three movies in Fig. 1 are also at different stages of their life cycles, and their evolving patterns obey



(a) Static Collaborative Filtering [1]



(b) Collaborative Filtering with absolute time [3]



(c) Collaborative Filtering with life cycles [This paper]

Fig. 2: Comparison of different recommendation models. The static CF model in (a) considers all observed ratings together and ignore the time. The temporal CF model in (b) considers learning separate factors of users and items in different time windows. Our model illustrated in (c) considers the inherent life cycles of users and items, and it learns factors for different stages of life cycles for users and items with temporal regularizations.

the hype curve. It is easy for any CF model to group these users and movies together by simply looking at their historical ratings. However, even all three users shows some interests to romance movies, we may not want to recommend Movie 3 to them since it is no longer a popular choice. Similarly, we may not want to recommend any romance movies to User C since he already lost the interest to the movie genre. For User A who is still young on romance, we can recommend Movie 1 and Movie 2 to him with high confidence since we believe his life cycle pattern will be similar to User B and User C (to have high interests in romance in the near future). For User B, the situation is similar except Movie 1 may be a dubious choice since they are very likely to miss each other's peak. It is obvious that other time-aware CF models which do not consider the life cycle will generate different recommendations, for example, they may not recommend any romance movies to User A since A resembles User C at current time and C no longer watches any romance movies. Hence, it is beneficial to model the life cycles in collaborative filtering.

In this paper, we study the problem of *time-aware item recommendation with life cycles*, where the goal is to learn a unified time-aware item recommendation model by considering the inherent life cycles of users and items. This problem is very natural and important due to the universal existence of life cycles in various applications, *e.g.,* life cycles of scholars and research topics (academia), life cycles of consumers and merchandises (on-line business) *etc.* Despite the significance of it, collaborative filtering with life cycles is highly challenging, as summarized below.

Life Cycle versus Wall Time: Users and items may exhibit dramatically different characteristic along the wall time (absolute time-line), but they may behave similar in each stage of their life cycles (relative time-line). Exploiting such similarity across the relative time-line may help the item recommendation task. However, it is still an open problem that how to incorporate the concept of life cycles into recommendation models given all the data distributed on the absolute time-line. It is even more challenging when we attempt to model the life cycles of both users and items simultaneously.

Sparsity: Considering the large number of users and items in a real-world application, the data of interactions between users and items are relatively sparse throughout the absolute time-line. The sparseness aggravates the prediction problem when we allocate the data into different stages of life cycles to learn separate factors for each life stage. If we do not have enough data in some life stages, there may not be enough information for us to learn the patterns for these life stages, which may lead to inaccurate recommendations.

In this paper, we propose a novel method called BiCycle to address the above issues, as illustrated in Fig. 2c. To the best of our knowledge, the collaborative filtering problem has not been studied in the context of life cycle in the literature.

II. Problem Formulation

In this section, we first introduce the preliminary and the formulation of the conventional collaborative filtering. Then we extends the problem to the time-aware variation. Throughout this paper, we use capital alphabet in boldface, *e.g.,* $\mathbf{X}$, to denote a matrix, and x_{ij} refers to the entry of $\mathbf{X}$ at i-th row and j-th column. We use lowercase alphabet in boldface, *e.g.,* $\mathbf{x}$, to denote a column-based vector, and x_i refers to the i-th entry of $\mathbf{x}$. We use $||\mathbf{X}||_F$ to denote the Frobenius norm of $\mathbf{X}$. We use calligraphic letters to denote sets, *e.g.,* $\mathcal{A}, \mathcal{B}, \mathcal{C}$.

A. Collaborative Filtering

In a recommender system, there are m users and n items. Each observed feedback is a tuple (i, j, r_{ij}) denoting the rating assigned to item i by user u, where the user index $i \in \{1, \ldots, m\}$, the item index $j \in \{1, \ldots, n\}$, and the rating values $r_{ij} \in \mathbb{R}$. We use $\Omega = \{(i, j)|r_{ij} \text{ is observed}\}$ to denote the the set of observed user-item interactions. The goal of conventional collaborative filtering problem is to estimate the unobserved rating $\{r_{ij}|(i, j) \notin \Omega\}$ based on the observed ones. However, the feedbacks are implicit [4], [5] rather than

TABLE I: Notations

Symbol	Definition
$\mathbf{R}$	the original rating matrix
$\mathbf{R}^{(pq)}$	the rating matrix of p-th user life stage and q-th item life stage
$\mathbf{U}^{(p)}$	the user latent factor matrix of p-th life stage
$\mathbf{V}^{(q)}$	the item latent factor matrix of p-th life stage
$\tilde{\mathbf{R}}^{(pq)}$	the smoothed rating matrix of p-th user life stage and q-th item life stage
λ	the regularization parameter
α	the variance regularization parameter
β	the fused regularization parameter
μ	the user interest decay parameter
π	the item interest decay parameter
M	the total number of user life stages
N	the total number of item life stages
D	the number of factors in the matrix factorization model
m	the number of users in the system
n	the number of items in the system

explicit in many applications, *e.g.,* , users' video viewing and product purchase history. In such cases, the value of a rating r_{ij} does not deliver any preference information of user i with respect to item j, and it only denotes an interaction between them is observed or not observed. In such cases, the goal of collaborative filtering is to estimate $\{r_{ij} \in \mathbb{R} | (i,j) \notin \Omega\}$ based on the observed interactions, where each r_{ij} is a real value indicating how likely user i will consume item j in the near future.

B. Matrix Factorization

One of the most successful approaches to collaborative filtering problem is based on a matrix factorization model [1], [2]. Matrix factorization models map uses and items to a joint low-rank latent factor space, such that the feedbacks are modeled as the inner products in that space. In this type of models, each user i is assigned a vector $\mathbf{u}_i \in \mathbb{R}^D$, and each item j is assigned a vector $\mathbf{v}_j \in \mathbb{R}^D$, where D is the dimensionality of the latent factor space. The rating of item j assigned by user i is predicted using the inner product of the corresponding vectors in the latent factor space as $\hat{r}_{ij} = \mathbf{u}_i^\top \mathbf{v}_j$. If we use $\mathbf{U} = (\mathbf{u}_1^\top, \ldots, \mathbf{u}_m^\top)$ to denote the latent factor matrix of all users, and $\mathbf{V} = (\mathbf{v}_1^\top, \ldots, \mathbf{v}_n^\top)$ to denote the latent factor matrix of all items in the system. Then the predicted matrix can be computed as the product of the corresponding vectors in the latent factor space: $\hat{\mathbf{R}} = \mathbf{U}^\top \mathbf{V}$, which is a dense matrix with all unobserved values in original rating matrix $\mathbf{R}$ filled.

In the setting of implicit feedbacks, $\mathbf{U}$ and $\mathbf{V}$ can be learned by solving the following optimization problem [4], [5]:

$$\underset{\mathbf{U},\mathbf{V}}{\text{minimize}} \ \frac{1}{2}||\mathbf{R} - \mathbf{U}^\top \mathbf{V}||_F^2 + \frac{\lambda}{2}(||\mathbf{U}||_F^2 + ||\mathbf{V}||_F^2) \qquad (1)$$

where λ is the parameter controls the extent of regularization, and can be tuned by cross-validation. It is worth noting that Eq. (1) is not a convex problem with respect to $\mathbf{U}$ and $\mathbf{V}$, but it is bi-convex. Hence, one can either use alternating least squares (ALS) or stochastic gradient descent to obtain the solutions $\mathbf{U}$ and $\mathbf{V}$ towards Eq. (1).

C. Time-aware CF

In conventional CF problems, we do not consider temporal information about the interactions or we assume that the data do not contain such information. However, in many real world applications, the time stamp associated with each ratings, reviews or visiting records is very useful and informative, and should not be ignored. In this section, we extend the problem of collaborative filtering to time-aware version. To formally define the problem of time-aware CF, we first give the definition of time window as follows.

DEFINITION II.1 *(Time Window)*: A time window is a left-closed-right-open time interval $[t_s, t_e)$ on the absolute time-line with the start time stamp t_s and the end time stamp t_e.

DEFINITION II.2 *(Time-aware CF Problem)*: In a recommender system with m users and n items, given a set of historical user-item feedbacks $\mathcal{F} = \{(i, j, r_{ijt}) | (i,j) \in \Omega, t \in [t_s, t_e)\}$, one want to estimate the the preference of the users towards the same set of items in the next time window $[t_e, t_f)$, *i.e.,* what items will a user consume in the future time span $[t_e, t_f)$. So the goal of time-aware CF problem is equivalent to infer a $m \times n$ preference matrix $\hat{\mathbf{R}}$, where $\hat{r}_{ij} \in \mathbb{R}$ denotes the preferences of user i towards item j in the future time window $[t_e, t_f)$. In the context of implicit feedbacks, $\hat{r}_{ij}$ corresponds to the confidence level on user i will consume item j in the future time window $[t_e, t_f)$. Thus, if $\hat{r}_{ij} > \hat{r}_{ij'}$, it indicates that user i is more likely to consume item j compared to item j' in $[t_e, t_f)$.

It is straightforward that the static matrix factorization models we discussed in Sec. II-B can be applied to infer the future preference matrix $\hat{\mathbf{R}}$ if we simply ignore the time stamps. However, the prediction results fails to capture the dynamic temporal effects exist in users as well as in items.

III. PROPOSED METHOD

In this section we introduce the proposed model BiCycle. In Sec. III-A, we first introduce the concept of life cycle. In Sec. III-B, we address the sparsity problem of the life-cycled matrix using exponential smoothing. Finally, in Sec. III-C, we discuss the temporal regularizations we propose and put all pieces together to present our solution.

A. Life Cycle

A major drawback of the conventional matrix factorization models is that they fail to consider temporal effects. One may

propose to simply partition the observed data by time window and learn a separate model using the data in each partition. However this approach not only suffers from severe sparsity, but isolates the interactions from different time periods, which makes the model difficult to capture the evolution patterns. As we discussed in Sec. I, similar users may share similar patterns during the same stage of their life cycles, and their evolution trend may also resemble each other. Similarly, similar items may also share similar patterns of popularity, availability and demand across life stages. Thus, it is beneficial to exploit the co-evolution patterns of users and items across the stages of their life cycle. In this work, we propose to model the co-evolution of users and items in a system using the concept of *life cycles*, which is formally defined as follows.

DEFINITION III.1 *(Life Cycle)*: In a system with users and items, we assume there are life cycles exist in both user side and item side, where each user has at most M life stages his/her life cycle, and each item has at most N life stages in its life cycle. And the concept of life cycle refers to the entire life span of users and items included the unobserved (future) time. Thus, given a certain time point, some users/items may already experienced all M or N life stages, while other users/items may only experienced partial life stages.

DEFINITION III.2 *(Life Stage)*: There are multiple life stages for a user or item. The ℓ-th life stage of user/item i is denoted by time window $[t_{s\ell}^{(i)}, t_{e\ell}^{(i)})$. Specially, we consider the M-th stage of user i or N-th stage of item j as a open time window $[t_{sM}^{(i)}, \infty)$ or $[t_{sN}^{(j)}, \infty)$ with no explicit end time. This is sensible since in real-world evolution patterns, the user or item usually enter a stable stage eventually at some time point, *e.g.,* the stable stage in the hype curve.

There are many ways to determine the life stages of users and items. In this paper, we use a fixed length for the first $M-1$ or $N-1$ stages, with the length of each user stage equals to l_u and the length of each item stage equals to l_i. We left the M-th or N-th stage with open end. Note that the determination of life stages is orthogonal to our model, any other sensible approach can be adopted with minor efforts.

Thus, for any user in the system, we can allocate their historical interactions into at most M bins, with each bin corresponds to a life stage. Similarly, for any item in the system, we can distribute their historical interactions into at most N stages.

In the static matrix factorization method, the historical feedbacks are cast into a $m \times n$ matrix $\mathbf{R}$. Similarly, by considering the life cycles of both users and items, we can cast the same historical feedbacks into $M \times N$ separate matrices of the same size, *i.e.,* $\{\mathbf{R}^{(pq)} \in \mathbb{R}^{m \times n} | p = 1, \ldots, M, q = 1, \ldots, N\}$. Hence, $\mathbf{R}^{(pq)}$ contains all the ratings given by users of p-th life stages to items of q-th life stages, *i.e.,*

$$r_{ij}^{(pq)} = \begin{cases} r_{ijt}, \exists r_{ijt} \in \mathcal{F} \text{ where } p = \mathcal{S}_U(i,t),\ q = \mathcal{S}_I(j,t). \\ 0, \text{ otherwise} \end{cases} \tag{2}$$

where $\mathcal{S}_U(i,t)$ is a function computes which life stage is user i in at the time stamp t, while $\mathcal{S}_I(j,t)$ computes which life stage is item j in at time stamp t. In this paper, we define $\mathcal{S}_U(i,t)$ as follows

$$\mathcal{S}_U(i,t) = min(M, \lceil \frac{t - t_0^{(i)}}{l_u} \rceil) \tag{3}$$

where $t_0^{(i)}$ is the time stamp of first observed interaction of user i. And $\mathcal{S}_I(j,t)$ follows the same form. The reason that we determine the life stage based upon the first observed interaction time is two-folds. First, the age of the users and items is not always available, relying on this kind of side information hurts the generality of the model. Second, the latent characteristic of users or items are not necessarily tied to their age, but to their experience, which can be learned from the historical interactions of them. For example, a user's taste on movies evolves while he/she watches more movies, and a movie's popularity and status also changes while it receives more reviews and criticism.

Hence, given the historical feedbacks of a system, we can cast them into a set of life-cycled preference matrices $\{\mathbf{R}^{(pq)}\}$. One can employ the matrix factorization model discussed in Sec. II-B to learn a latent factor matrix of users for each stage, *i.e.,* $\{\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(M)}\}$, and a latent factor matrix of items for each stage, *i.e.,* $\{\mathbf{V}^{(1)}, \ldots, \mathbf{V}^{(N)}\}$. Although this approach can capture users and items patterns at different life stages, it has some limitations. Firstly, it suffers from severe sparsity, since the original data is sparse and we allocate them into $M \times N$ separate matrices of original size. With such sparsity, it is difficult for factorization model to learn meaningful latent factors in each life stage. Secondly, it does not force any regularization in the temporal dimension, which makes the model highly possible to over-fitting the data.

B. Smoothing

To address the first issue we discussed above, we need to alleviate the sparseness of $\mathbf{R}^{(pq)}$. Inspired by the works of modeling interest forgetting in music recommendation [10], we build a smoothed counterpart for each $\mathbf{R}^{(pq)}$ as follows.

$$\tilde{\mathbf{R}}^{(pq)} = \begin{cases} \mathbf{R}^{(pq)}, \text{ if } p = q = 1. \\ \mathbf{R}^{(pq)} + \mu \tilde{\mathbf{R}}^{((p-1),q)}, \text{ if } q = 1 \text{ and } p \geq 2 \\ \mathbf{R}^{(pq)} + \pi \tilde{\mathbf{R}}^{(p,(q-1))}, \text{ if } p = 1 \text{ and } q \geq 2 \\ \mathbf{R}^{(pq)} + \mu \tilde{\mathbf{R}}^{((p-1),q)} + \pi \tilde{\mathbf{R}}^{(p,(q-1))}, \text{ otherwise} \end{cases} \tag{4}$$

where $\mu \in [0,1]$ and $\pi \in [0,1]$ are two parameters to control the rate of interest decay over the user life stages and item life stages respectively. This strategy extends the well-known exponential smoothing [11] in time series studies to fit our application. The intuitions behind this strategy are: (1) the interactions in previous life stages will affect the consumption or interactions in the future stages. (2) interactions in earlier stage has less impacts on current or future life stages, which can also be explained by people tends to forget their interest far back in early stages of life. (3) the impact decays exponentially

over life stages, which is a general assumption of many temporal models.

By applying the smoothing as shown in Eq. (4), we obtain a set of denser preference matrices $\{\tilde{\mathbf{R}}^{(pq)}\}$. Then we can perform the matrix factorization on $\{\tilde{\mathbf{R}}^{(pq)}\}$ to learn M user factor matrices and N item factor matrices, which can be formulated as the following optimization problem.

$$\begin{aligned}\min_{\{\mathbf{U}^{(p)}\},\{\mathbf{V}^{(q)}\}} & \frac{1}{2}\sum_{p=1}^{M}\sum_{q=1}^{N}||\tilde{\mathbf{R}}^{(pq)}-\mathbf{U}^{(p)}(\mathbf{V}^{(q)})^{\top}||_F^2 \\ & +\frac{\lambda}{2}(\sum_{p=1}^{M}||\mathbf{U}^{(p)}||_F^2+\sum_{q=1}^{N}||\mathbf{V}^{(q)}||_F^2)\end{aligned} \tag{5}$$

This problem is an analog of Eq. (1), the major difference here is that we are trying to learn M separate user factor matrices and N separate item factor matrices instead of one for each. Even though the smoothing alleviate the sparsity of the preference matrices, it is still possible that the model in Eq. (5) over-fits the data due to the lack of temporal regularizations.

Algorithm 1 BiCycle Algorithm

Require: $\mathbf{R}, M, N, \lambda, \alpha, \beta, \pi, \mu$

1: Build the life-cycled preference matrices $\{\mathbf{R}^{(pq)}\}(p = 1,\ldots,M, q = 1,\ldots,N)$ using Eq. (2) and Eq. (3).
2: Obtain the smoothed preference matrices $\tilde{\mathbf{R}}^{(pq)}$ using Eq. (4).
3: Build the temporal variance regularization using Eq. (6).
4: Build the temporal fused regularization using Eq. (7).
5: Solving the Eq. (8) using Eq. (9) and Eq. (11) to obtain $\{\mathbf{U}^{(p)}\}(p = 1,\ldots,M)$ and $\{\mathbf{V}^{(q)}\}(q = 1,\ldots,N)$.
6: $\hat{\mathbf{R}}^{(pq)} \leftarrow \mathbf{U}^{(p)}\mathbf{V}^{(q)}$.
7: **Return** $\{\hat{\mathbf{R}}^{(pq)}\}$,

C. Temporal Regularization

In this section, we consider two types of temporal regularizations for our model. The first one is called *variance regularizer*, which has the following form,

$$\begin{aligned}&L_v(\mathbf{U}^{(1)},\ldots,\mathbf{U}^{(M)},\mathbf{V}^{(1)},\ldots,\mathbf{V}^{(N)}) \\ &=\sum_{p=1}^{M}||\mathbf{U}^{(p)}-\frac{1}{m}\mathbf{J}^{m\times m}\mathbf{U}^{(p)}||_F^2 \\ &+\sum_{q=1}^{N}||\mathbf{V}^{(q)}-\frac{1}{n}\mathbf{J}^{n\times n}\mathbf{V}^{(q)}||_F^2\end{aligned} \tag{6}$$

where $\mathbf{J}^{m\times m}$ corresponds to the all-ones matrix of size $m\times m$. The geometrical meaning of $\frac{1}{m}\mathbf{J}^{m\times m}\mathbf{U}^{(p)}$ is simply a matrix with each row equals to the mean of all rows of $\mathbf{U}^{(p)}$. Thus, $||\mathbf{U}^{(p)}-\frac{1}{m}\mathbf{J}^{m\times m}\mathbf{U}^{(p)}||_F$ is the magnitude of the variance of the rows in $\mathbf{U}^{(p)}$. Since each row in $\mathbf{U}^{(p)}$ corresponds to a user's feature vector in the p-th life stage, and each row in $\mathbf{V}^{(q)}$ refers to a item's feature vector in the q-th life stage, it is obvious that Eq. (6) denotes the overall variance of user features and item features inside each life stage of them. When minimizing the objective function shown in Eq. (5), we also want to keep this variance regularizer fairly small to force the users and items sharing similar patterns and characters inside the same life stage, as we discussed in Sec. I.

The second regularizer we consider is called *fused regularizer*, which is named after fused Lasso. This regularizer is formulated as follows

$$\begin{aligned}&L_f(\mathbf{U}^{(1)},\ldots,\mathbf{U}^{(M)},\mathbf{V}^{(1)},\ldots,\mathbf{V}^{(N)}) \\ &=\sum_{p=1}^{M-1}||\mathbf{U}^{(p+1)}-\mathbf{U}^{(p)}||_F^2+\sum_{q=1}^{N-1}||\mathbf{V}^{(q+1)}-\mathbf{V}^{(q)}||_F^2\end{aligned} \tag{7}$$

It is obvious that the fused regularizer computes the difference between any two feature matrices of two continuous life stages. By adding this regularizer to the minimization problem, it encourages the change of any two continuous life stages of users and items to be small. In other words, the user features and item features are usually changing gradually instead of rapidly.

When we consider the two types of regularizations together with the major objective, we can adopt the standard approach of adding weight parameters for the regularizers to merging them into one problem. Accordingly, the BiCycle optimization problem is as follows

$$\begin{aligned}\min_{\{\mathbf{U}^{(p)}\},\{\mathbf{V}^{(q)}\}} & \frac{1}{2}\sum_{p=1}^{M}\sum_{q=1}^{N}||\mathbf{R}^{(pq)}-\mathbf{U}^{(p)}(\mathbf{V}^{(q)})^{\top}||_F^2 \\ & +\frac{\lambda}{2}(\sum_{p=1}^{M}||\mathbf{U}^{(p)}||_F^2+\sum_{q=1}^{N}||\mathbf{V}^{(q)}||_F^2) \\ & +\frac{\alpha}{2}L_v(\mathbf{U}^{(1)},\ldots,\mathbf{U}^{(M)},\mathbf{V}^{(1)},\ldots,\mathbf{V}^{(N)}) \\ & +\frac{\beta}{2}L_f(\mathbf{U}^{1)},\ldots,\mathbf{U}^{(M)},\mathbf{V}^{(1)},\ldots,\mathbf{V}^{(N)})\end{aligned} \tag{8}$$

where λ keeps the same meaning as in Eq. (1), $\alpha \geq 0$ controls the extent of variance regularizer and $\beta \geq 0$ controls the level of fused regularizer. The values of these parameters should be determined using cross-validation. Obviously, this problem is not convex to $\mathbf{U}^{(1)},\ldots,\mathbf{U}^{(M)}$ and $\mathbf{V}^{(1)},\ldots,\mathbf{V}^{(N)}$. However, it is convex and differentiable when we fix all but one of them. Hence, we can utilize gradient descent to solve Eq.(8) iteratively.

D. Optimization

In this section, we briefly discuss the gradient computation in each iteration of our algorithm. Suppose we denote the objective function in Eq. (8) for short as $F(\{\mathbf{U}^{(p)}\},\{\mathbf{V}^{(q)}\})$. At the c-th iteration of the algorithm, we first fix all other variables to update $\mathbf{U}^{(p)}$. We can derive the gradient of F over $\mathbf{U}^{(p)}$ as

$$\begin{aligned}\frac{\partial F(\{\mathbf{U}^{(p)}\},\{\mathbf{V}^{(q)}\})}{\partial \mathbf{U}^{(p)}} =& \sum_{q=1}^{N}\mathbf{U}^{(p)}(\mathbf{V}^{(q)})^{\top}\mathbf{V}^{(q)}-\tilde{\mathbf{R}}^{(pq)}\mathbf{V}^{(q)} \\ & +\lambda\mathbf{U}^{(p)}+\alpha(\mathbf{U}^{(p)}-\frac{1}{n}\mathbf{J}^{m\times m}\mathbf{U}^{(p)} \\ & +C\end{aligned} \tag{9}$$

where C is defined as follows

$$C = \begin{cases} \beta(\mathbf{U}^{(p+1)} - \mathbf{U}^{(p-1)}), \text{ if } 1 < p < M. \\ \beta(\mathbf{U}^{(p+1)} - \mathbf{U}^{(p)}), \text{ if } p = 1 \\ \beta(\mathbf{U}^{(p)} - \mathbf{U}^{(p-1)}), \text{ if } p = M \end{cases} \tag{10}$$

Thus, the updated $\mathbf{U}^{(p)}$ can be obtained by

$$\mathbf{U}^{(p)} = \mathbf{U}^{(p)}_{old} - \zeta \frac{\partial F(\{\mathbf{U}^{(p)}\}, \{\mathbf{V}^{(q)}\})}{\partial \mathbf{U}^{(p)}} \tag{11}$$

where $\zeta > 0$ is the step size parameter which is usually small. After $\mathbf{U}^{(p)}$ is updated, we follow the similar approach to update other variables until the algorithm converges. The situation of fixing all other variables and update $\mathbf{V}^{(q)}$ is similar and symmetrical to above, so we omit the detailed derivation for succinct. To summarize, the pseudocode of our proposed bicycle algorithm is presented in Algorithm 1.

E. Complexity Analysis

We provide a brief complexity analysis of the proposed BiCycle Algorithm. The time complexity for each iteration of BiCycle is $\mathcal{O}(MN|\Omega|D^2 + MN(m+n)D^3)$, where M and N usually do not change when m or n increases and $D \ll \min(m, n)$. Thus, the time complexity can be denoted as $\mathcal{O}(|\Omega|D^2+(m+n)D^3)$, which is similar to the conventional ALS algorithm for collaborative filtering [12].

The space complexity is $\mathcal{O}(MNmn+DMm+DNn)$. Due to the same reason, we can simplify it as $\mathcal{O}(mn)$. Furthermore, considering the sparsity of the preference matrix, the actual space usage is far less than that. As we will show latter in Sec. IV-F, the best choice of M and N is usually small for all datasets tested, making the proposed algorithm almost as efficient as the conventional matrix factorization method.

IV. Experiments

A. Dataset

In order to validate the performances, we apply our methodology to three real-world datasets.

- **Epinions** [13]: The first dataset is extracted from the well-known general consumer review site Epinions[1], which was established in 1999. We randomly select 1,000 active users and 1,000 popular items from the raw data, and the time span of their interactions is from 2001 to 2010. All ratings are binarized to simulate the context of implicit feedbacks.
- **DBLP-Terms** [14]: The second dataset is a bibliography keywords dataset extracted from DBLP[2] information network. We select a subset of active authors who published more than 5 papers in top conferences of four areas[3] that related to data mining from 1980 to 2010 and keep the 1000 top authors. Then we build a stemmed terms representation (stop words removed) of all paper titles of the subset using gensim[4]. We filter out all terms that appear less than 20 times to get 847 keywords. The interactions between authors and terms in each year are used as implicit ratings.
- **ML100K** [15]: MovieLens 100K[5] is a widely used movie recommendation dataset that contains 100,000 ratings from 943 users on 1,682 movies, each user rate at least 20 movies. The time span of this dataset is from Sept. 1997 to Apr. 1998. We divide the data into 29 time windows of equal length. And all real-valued ratings are binarized to simulate the context of implicit feedbacks.

B. Compared Methods

In order to demonstrate the effectiveness of our method, the following baselines and state-of-the-art methods are tested:

- POP [5]: Recommend popular items to users.
- MF [1]: This method is the static matrix factorization with ℓ_2-norm regularization we described in Sec. II-B.
- SVD++ [2]: This is an extended version of matrix factorization method which couples SVD and neighborhood model together. SVD++ also takes the implicit feedbacks into consideration.
- TIMESVD [3]: This is a popular state-of-art time-aware collaborative filtering method that captures temporal effects by incorporating a time-variant bias for each user and item at every individual time window.
- ITEMCYCLE [this paper]: This is a degenerated variation of our proposed method, which only considers the life cycles of the item side.
- USERCYCLE [this paper]: This is another degenerated variation of our proposed method, which only considers the life cycles of the user side.
- BiCycle [this paper]: the proposed BiCycle Algorithm.

C. Experimental Settings

For the purpose of quantitative evaluations, we follow the standard testing protocol with minor alterations. Given a set of consumption data in k continuous time windows, we select some test window $1 < c \leq k$. Then we randomly sample 20% data in test window c, together with all data in time windows $1, \ldots, c-1$ as the training set. The remaining 80% data in c is used as test set. To ensure reliability, for each dataset, we choose multiple test windows to run the experiments and report the results to reflect the overall performance of compared methods over the time-line. We select the parameters used for the proposed BiCycle algorithm using grid search on a validation set. Due to the number of parameters to tune is relatively large (7 for the BiCycle method), an extensive grid search is very expensive. We can employ an alternative approach by fixing other parameters and searching the best values for one or two target parameters. An extensive parameter study of our BiCycle algorithm is provided in Sec. IV-F.

[1] http://www.epinions.com

[2] http://dblp.uni-trier.de/db/

[3] Data Mining: KDD, PKDD, ICDM, SDM, PAKDD; Database: SIGMOD Conference, VLDB, ICDE, PODS, EDBT; Information Retrieval: SIGIR, ECIR, ACL, WWW, CIKM; and Machine Learning: NIPS, ICML, ECML, AAAI, IJCAI.

[4] https://radimrehurek.com/gensim/

[5] http://grouplens.org/datasets/movielens/

TABLE II: Prec@k (rank) and MRR (rank) of compared methods. $D = 10$. The best performer is in **boldface**.

		Window 1				Window 2				Window 3				Ave.
Dataset	Methods	Prec@1	Prec@5	Prec@10	MRR	Prec@1	Prec@5	Prec@10	MRR	Prec@1	Prec@5	Prec@10	MRR	Rank
Epinions	BiCycle	**0.0282** (1)	0.0175 (2)	0.0155 (2)	**0.0245** (1)	**0.0165** (1)	0.0107 (2)	0.0079 (2)	**0.0167** (1)	**0.0118** (1)	**0.0094** (1)	0.0076 (2)	0.0206 (2)	1.5
	UserCycle	0.0113 (4)	0.0051 (6)	0.0073 (4)	0.0143 (4)	0.0083 (3)	0.0058 (4)	0.0070 (3)	0.0138 (4)	0.0059 (3)	0.0047 (4)	0.0047 (5)	0.0128 (5)	4.1
	ItemCycle	0.0141 (3)	0.0079 (4)	0.0056 (5)	0.0128 (5)	0.0041 (4)	0.0025 (7)	0.0041 (6)	0.0078 (6)	0.0059 (3)	**0.0094** (1)	0.0053 (3)	0.0179 (3)	4.6
	MF	0.0000 (6)	0.0056 (5)	0.0054 (6)	0.0113 (6)	0.0000 (6)	0.0041 (6)	0.0025 (7)	0.0097 (5)	0.0000 (6)	0.0011 (7)	0.0029 (6)	0.0105 (6)	6.0
	SVD++	0.0085 (5)	0.0085 (3)	0.0099 (3)	0.0168 (3)	0.0041 (4)	0.0058 (4)	0.0050 (5)	0.0152 (3)	0.0059 (3)	0.0047 (4)	0.0053 (3)	0.0130 (4)	3.7
	TimeSVD	0.0254 (2)	**0.0192** (1)	**0.0158** (1)	0.0244 (2)	0.0124 (2)	**0.0140** (1)	**0.0087** (1)	0.0165 (2)	0.0118 (1)	0.0082 (3)	**0.0147** (1)	**0.0208** (1)	1.5
	Pop	0.0000 (6)	0.0023 (7)	0.0019 (7)	0.0024 (7)	0.0000 (6)	0.0066 (3)	0.0070 (3)	0.0018 (7)	0.0000 (6)	0.0012 (6)	0.0012 (7)	0.0015 (7)	6.0
DBLP-Terms	BiCycle	**0.2874** (1)	**0.1767** (1)	**0.1336** (1)	**0.1434** (1)	**0.2995** (1)	**0.1753** (1)	**0.1286** (1)	**0.1408** (1)	**0.2680** (1)	**0.1658** (1)	0.1198 (2)	**0.1358** (1)	1.1
	UserCycle	0.2781 (3)	0.1749 (3)	0.1286 (4)	0.1365 (3)	0.2930 (3)	0.1622 (4)	0.1220 (4)	0.1352 (3)	0.2576 (2)	0.1603 (4)	0.1198 (2)	0.1317 (3)	3.2
	ItemCycle	0.2848 (2)	0.1754 (2)	0.1318 (2)	0.1407 (2)	0.2943 (2)	0.1734 (2)	0.1286 (1)	0.1393 (2)	0.2510 (4)	0.1622 (2)	**0.1213** (1)	0.1327 (2)	2.0
	MF	0.2741 (4)	0.1762 (4)	0.1306 (3)	0.1357 (4)	0.2878 (4)	0.1698 (3)	0.1234 (3)	0.1352 (3)	0.2522 (3)	0.1608 (3)	0.1172 (4)	0.1294 (4)	3.5
	SVD++	0.1992 (5)	0.1538 (5)	0.1152 (6)	0.1114 (6)	0.1875 (5)	0.1591 (5)	0.1135 (5)	0.1144 (5)	0.2102 (5)	0.1469 (5)	0.1037 (6)	0.1132 (5)	5.3
	TimeSVD	0.1738 (6)	0.1441 (6)	0.1178 (5)	0.1143 (5)	0.1836 (6)	0.1445 (6)	0.1111 (6)	0.1115 (6)	0.1800 (6)	0.1396 (6)	0.1064 (5)	0.1120 (6)	5.8
	Pop	0.0067 (7)	0.0043 (7)	0.0040 (7)	0.0062 (7)	0.0000 (7)	0.0049 (7)	0.0048 (7)	0.0137 (7)	0.0000 (7)	0.0024 (7)	0.0032 (7)	0.0138 (7)	7.0
ML100K	BiCycle	**0.2800** (1)	**0.2000** (1)	**0.1680** (1)	**0.1311** (1)	**0.3939** (1)	**0.3515** (1)	**0.3182** (1)	**0.2125** (1)	**0.3690** (1)	**0.3071** (1)	**0.2678** (1)	**0.1911** (1)	1.0
	UserCycle	0.2200 (3)	0.1880 (2)	0.1620 (3)	0.1282 (3)	0.3333 (3)	0.3060 (4)	0.2909 (3)	0.2041 (3)	0.3571 (2)	0.2976 (2)	0.2667 (2)	0.1889 (2)	2.7
	ItemCycle	**0.2800** (1)	0.1880 (2)	0.1640 (2)	0.1296 (2)	0.3484 (2)	0.3242 (2)	0.2954 (2)	0.2118 (2)	0.3214 (3)	0.2833 (3)	0.2583 (3)	0.1868 (3)	2.3
	MF	0.2200 (3)	0.1680 (4)	0.1540 (4)	0.1237 (4)	0.3333 (3)	0.3090 (3)	0.2909 (3)	0.1962 (4)	0.2857 (4)	0.2880 (4)	0.2571 (4)	0.1799 (4)	3.3
	SVD++	0.1800 (5)	0.1360 (5)	0.1240 (5)	0.0920 (5)	0.2273 (6)	0.2030 (6)	0.1901 (6)	0.1322 (5)	0.2381 (6)	0.2071 (6)	0.1821 (6)	0.1192 (5)	5.5
	TimeSVD	0.1800 (5)	0.1320 (6)	0.1080 (6)	0.0740 (6)	0.3030 (5)	0.2273 (5)	0.2106 (5)	0.1069 (6)	0.2619 (5)	0.2309 (5)	0.2083 (5)	0.1020 (6)	5.4
	Pop	0.0200 (7)	0.0080 (7)	0.0060 (7)	0.0098 (7)	0.0000 (7)	0.0121 (7)	0.0121 (7)	0.0130 (7)	0.0119 (7)	0.0071 (7)	0.0083 (7)	0.0100 (7)	7.0

D. Evaluation Metric

Three commonly used metrics in information retrieval are adopted to evaluate the quality of recommendation given by compared methods. They are Average Recall at Position k (Recall@k), Average Precision at Position k (Prec@k) and Mean Reciprocal Rank (MRR). The definitions of these metrics are as follows

$$\text{Recall@}k = \frac{1}{n}\sum_{i=1}^{n}\frac{\text{Hit}_i(k)}{|\mathcal{Q}_i|} \tag{12}$$

$$\text{Prec@}k = \frac{1}{n}\sum_{i=1}^{n}\frac{\text{Hit}_i(k)}{k} \tag{13}$$

$$\text{MRR} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{|\mathcal{Q}_i|}\sum_{j=1}^{|\mathcal{Q}_i|}\frac{1}{\text{rank}_{ij}} \tag{14}$$

where $\mathcal{Q}_i$ denotes the set of items consumed by user i in the test window, $\text{Hit}_i(k)$ refers to the number of hits in the top k recommendations of user i given by the model, and rank_{ij} denotes the rank of item j consumed by user i in the recommendations.

E. Experimental Results

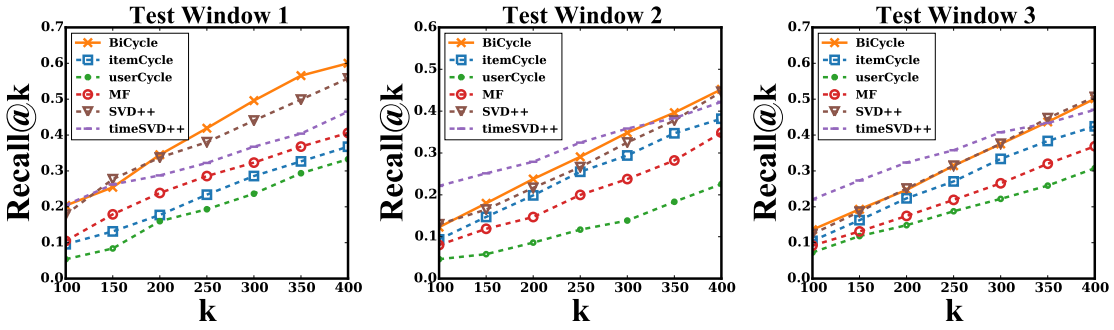
In this section, we summarize our the findings on the experimental results. Table II presents the results of compared models in terms of Precision@k scores and MRR score. Fig. 3 shows Recall@k with $D = 10$ fixed by varying k from 100 to 400. Fig. 4 shows Recall@300 by varying D from 10 to 50. For the quality of visualization, the curve of Pop in all figures and the curve of TimeSVD in Fig 4c are omitted because their recall scores are too low.

We first study the effectiveness of BiCycle by comparing it with two static recommendation methods: MF and SVD++. Both MF and SVD++ utilize the historical feedbacks without considering the inherent life cycles in users or items. Table II indicates that on all three datasets we tested, BiCycle can achieve much better performances than the MF and SVD++ in terms of Precision@k scores and the MRR socre. And form Fig. 3 and Fig. 4, we can observe that BiCycle consistently outperforms MF and SVD++ in terms of Recall@k scores with different k. This is because our proposed method BiCycle consider the temporal dynamics of users and items by using the concept of life cycles, while the static methods MF and SVD++ do not. The results suggest that the proposed life cycle concept could correctly model the temporal dynamics, and it is a very concise but effective approach.

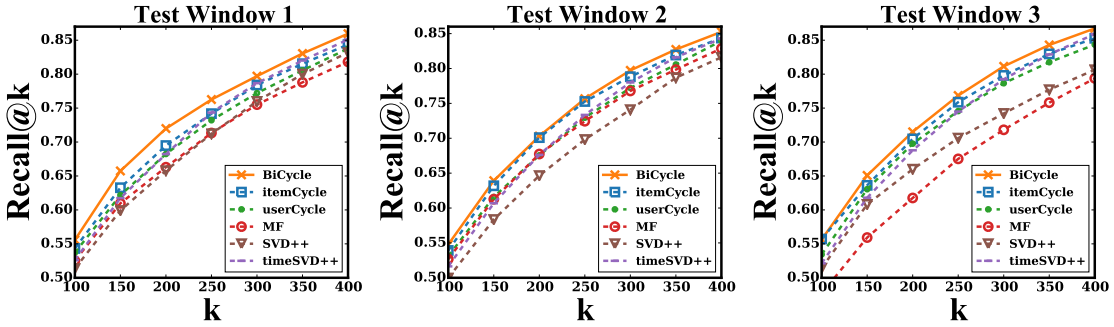
We further study the effectiveness of BiCycle by comparing it with a state-of-art time-aware recommendation method TimeSVD. Table II, Fig. 3 and Fig. 4 suggest that BiCycle achieve compariable or even better performances than TimeSVD. This is because that TimeSVD models the temporal effects using the absolute timeline, while the proposed BiCycle uses the concept of life cycles with re-aligned relative timeline. The results support our intuition of the existence of inherent life cycles in both users and items, and using such life cycles can help item recommendation methods better model the temporal dynamics. The only exception is on Epinions data set where the observed data is much more sparser than other two data sets, but their performances are close. When the sparsity is not that severe, our proposed BiCycle can perform much better than TimeSVD.

From Table II, Fig. 3 and Fig. 4, we further observe that BiCycle can usually perform better than its degenerated version UserCycle and ItemCycle. This observation indicates that modeling life cycles for both user side and item side is better than only modeling life cycles of one side. Interestingly, we also observe that ItemCycle usually outperforms UserCycle. It suggests that it is easier to capture the nature of item life cycles than to capture the nature of user life cycles. However, if we consider the user life cycles and item life cycles collectively, we can still improve the recommendation performance.

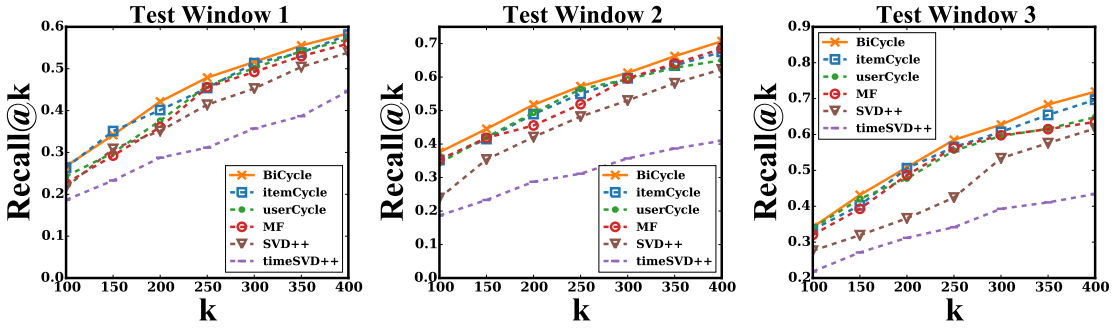
The model Pop serves as a naive baseline in our experiments, and it is consistently outperformed by other methods



(a) Recall@k at Epinions, $D = 10$. Vary k.



(b) Recall@k at DBLP-Terms, $D = 10$. Vary k.



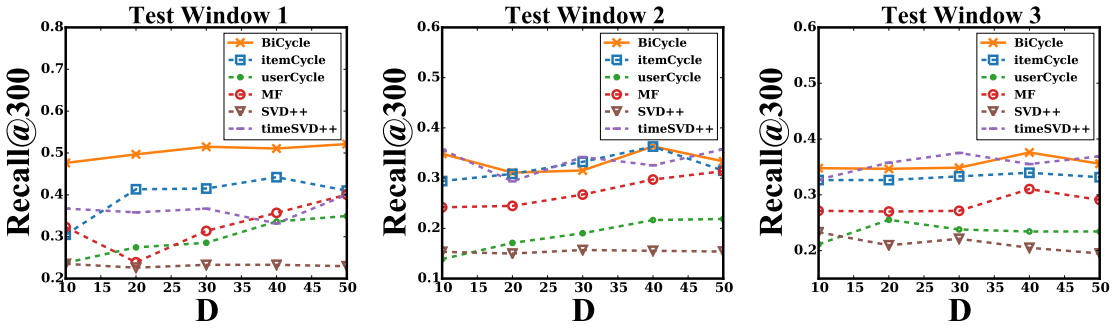
(c) Recall@k at ML100K, $D = 10$. Vary k.

Fig. 3: Recall of different models. Each row refers to a data set and each column refers to a testing window. This figure present the effect of varying k with fixed $D = 10$. Higher values are better.

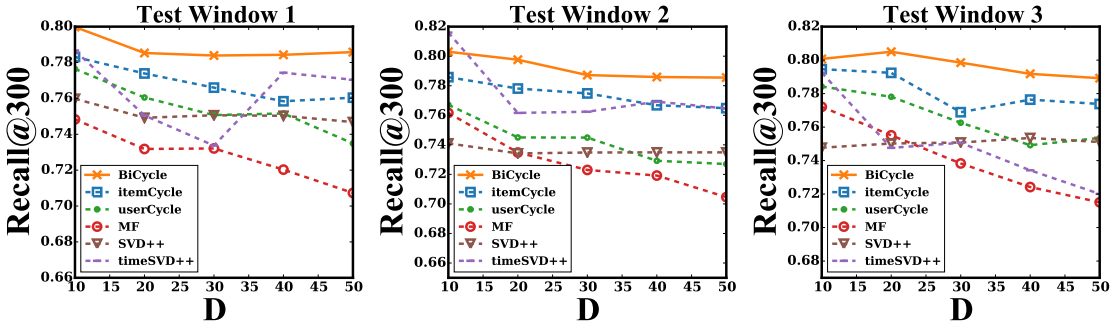
since it only considers the overall popularity of items.

F. Parameter Settings

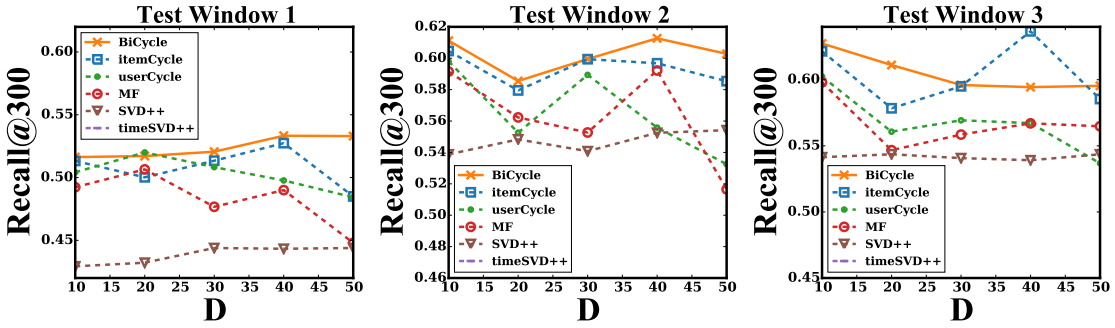
In this section, we show the effects of the parameters in BiCycle. We test α and β with values among $\{10, 1, 0.1, 0.01, 0.001, 0\}$ separately, with $\lambda = 1.0$ and $D = 10$ fixed. The average results of recall@300 are reported. As shown in Fig 5, the performance of our model using some nonzero values of α and β can be better than the case of $\alpha = 0$ or $\beta = 0$. For example, in the Epinions dataset (Fig 5a), the best parameter setting is $\alpha = 1.0$ and $\beta = 1.0$, and the corresponding recall is 49.60%. This setting is the same as our default setting in previous experiments. The situation is similar in other two datasets. These results demonstrate the effectiveness of the proposed temporal regularizations on life cycles. Besides, we also present the effects of smoothing parameters μ and π in Fig. 6a, where we set μ and π with values among $\{0, 0.1, \ldots, 0.6\}$ separately. As shown in Fig. 6a, the best setting for Epinion dataset is $\mu = 0.5$ and $\pi = 0.1$, and the corresponding recall is 42.73%. In previous experiments, we use the default setting with $\mu = \pi = 0.6$. If we try to optimize the selection of μ and π values, the performance improvement will be even larger. Finally, we show the effects of M and N values in Fig 6b, we can see that our model achieves at most 79.73% recall when considering modeling multiple life stages for users and items ($M = 2$, $N = 5$), which is much better than 75.51% achieved by static setting ($M = N = 1$). In our experiments, we choose different



(a) Recall@300 at Epinions. Vary D.



(b) Recall@300 at DBLP-Terms. Vary D.



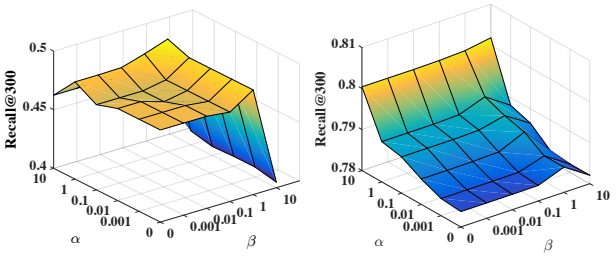
(c) Recall@300 at ML100K. Vary D.

Fig. 4: Recall of different models. Each row refers to a data set and each column refers to a testing window. This figure presents the effect of varying D with fixed $k = 300$. Higher values are better.

values of M and N for each dataset using cross-validation.

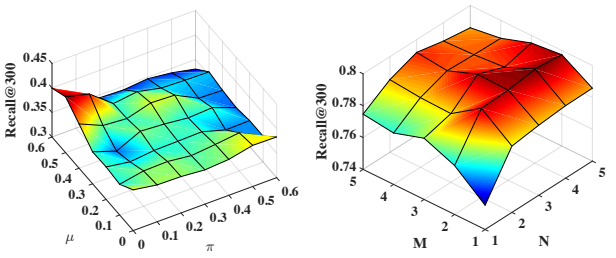
V. Related Work

The goal of collaborative filtering (CF) [1], [3], [16]–[19] is to make accurate recommendations to users based on their past reviews and feedbacks on existing products. Sarwar *et al.* proposed an item-based approach for collaborative filtering, in which they utilize the similarity among items to make recommendation [18]. Koren *et al.* proposed to use low rank matrix factorization methodf for collaborative filtering, which later become a popular approach in the literature [1]. Liu *et al.* proposed a kernelized matrix factorization approach for CF problem to improve the accuracy [16]. Koren proposed the very first attempt in this direction, which is also known as TIMESVD (or TimeSVD++) [3]. TIMESVD models a time-variant bias for each user and item at every individual time window. This model is proved to be effective on explicit feedbacks data, where users assign real-valued ratings to consumed items. So it can track the time-variant bias using these observed values of ratings. However, in implicit feedbacks data where ratings do not contain any preferences information, the performance of TIMESVD can be degenerated. Xiong *et al.* [8] proposed an tensor-based factorization model to directly incorporate absolute time as an extra dimension in order to capture the preference change over time. Li *et al.* [7] adapted Bi-LDA to model the temporal dynamics on both user side and item side. Even though their proposed method is outperformed by Bi-LDA itself, they demonstrated that their model can



(a) Epinions (b) DBLP-Terms

Fig. 5: Recall@300 of BiCycle with different α and β



(a) Epinions (on μ and π) (b) DBLP (on M and N)

Fig. 6: Recall@300 of BiCycle with different parameters

achieve better interpretation of concept drift in users. Zhang *et al.* [6] followed the generative method in [8] to propose another Bayesian factorization model. A transaction matrix is introduced in [6] to enforce the assumption of user preferences evolve gradually over time, which is an analog of the fused regularization we proposed in this paper, but in different formulations. Chang *et al.* [20] proposed a novel PU learning framework to capture the temporal dynamics in information networks or a recommender system with streaming data.

Evolution inference is first introduced by Zhang *et al.* in [21], in which they established the concept of life cycle and life stage of the temporal data in Heterogeneous Information Networks (HINs). In [21], the life cycle only exists in one type of node (*e.g.,* author in DBLP or project in Prosper.) In this paper, we extend the concepts of life cycle and life stage to both user side and item side. To the best of our knowledge, this paper is the first endeavor to incorporate the life cycle into item recommendation.

VI. Conclusion

We studied the item recommendation problem with life cycles. We proposed BiCycle to effectively incorporate the life cycles of users and items into the recommendation model. Different from previous models, BiCycle considers temporal dynamics using life cycles instead of absolute time-line. To alleviate the sparsity, BiCycle utilizes exponential smoothing to smooth the data over the life stages. Besides, BiCycle enforces two temporal regularizations to encourage the variations within each life stage to be small and to force the evolution between continuous stages not too rapid. Our experimental study on three real-world datasets demonstrate that BiCycle can improve the item recommendation quality.

VII. Acknowledgement

This work is supported in part by National Science Foundation through grant IIS-1718310.

References

[1] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[2] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *Proc. 14th ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD'08)*, 2008, pp. 426–434.

[3] ——, "Collaborative filtering with temporal dynamics," *Communications of the ACM*, vol. 53, no. 4, pp. 89–97, 2010.

[4] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *Proc. 8th IEEE Int. Conf. Data Mining (ICDM'08)*, 2008, pp. 263–272.

[5] X. Yu, X. Ren, Y. Sun, Q. Gu, B. Sturt, U. Khandelwal, B. Norick, and J. Han, "Personalized entity recommendation: A heterogeneous information network approach," in *Proc. 7th ACM Int. Conf. Web Search and Data Mining (WSDM'14)*, 2014, pp. 283–292.

[6] C. Zhang, K. Wang, H. Yu, J. Sun, and E. Lim, "Latent factor transition for dynamic collaborative filtering," in *Proc. 14th SIAM Int. Conf. Data Mining (SDM'14)*, 2014, pp. 452–460.

[7] B. Li, X. Zhu, R. Li, C. Zhang, X. Xue, and X. Wu, "Cross-domain collaborative filtering over time," in *Proc. 22nd Int. Joint Conf. Artificial Intelligence (IJCAI'11)*, 2011, pp. 2293–2298.

[8] L. Xiong, X. Chen, T. Huang, J. Schneider, and J. Carbonell, "Temporal collaborative filtering with bayesian probabilistic tensor factorization," in *Proc. 10th SIAM Int. Conf. Data Mining (SDM'10)*, 2010, pp. 211–222.

[9] A. Linden and J. Fenn, "Understanding gartners hype cycles," *Strategic Analysis Report R-20-1971. Gartner, Inc*, 2003.

[10] J. Chen, C. Wang, and J. Wang, "A personalized interest-forgetting markov model for recommendations," in *AAAI Conf. Artificial Intelligence 29th (AAAI '15)*, 2015, pp. 16–22.

[11] C. Lewis, *Industrial and business forecasting methods: A practical guide to exponential smoothing and curve fitting*. Butterworth-Heinemann, 1982.

[12] H. Yu, C. Hsieh, S. Si, and I. Dhillon, "Parallel matrix factorization for recommender systems," *Knowledge and Information Systems*, vol. 41, no. 3, pp. 793–819, 2014.

[13] M. Richardson, R. Agrawal, and P. Domingos, "Trust management for the semantic web," in *2nd Intl. Semantic Web Conf. (ISWC'03)*, 2003, pp. 351–368.

[14] M. Ji, Y. Sun, M. Danilevsky, J. Han, and J. Gao, "Graph regularized transductive classification on heterogeneous information networks," in *Proc. European Conf. Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD'10)*, 2010, pp. 570–586.

[15] F. Harper and J. Konstan, "The movielens datasets: History and context," *ACM Transactions on Interactive Intelligent Systems*, vol. 5, no. 4, p. 19, 2016.

[16] X. Liu, C. Aggarwal, Y. Li, X. Kong, X. Sun, and S. Sathe, "Kernelized matrix factorization for collaborative filtering," in *Proc. 16th SIAM Int. Conf. Data Mining (SDM'16)*, 2016, pp. 378–386.

[17] B. Yang, Y. Lei, D. Liu, and J. Liu, "Social collaborative filtering by trust," in *Proc. 24th Int. Joint Conf. Artificial Intelligence (IJCAI'13)*, 2013, pp. 2747–2753.

[18] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proc. 10th Int. World Wide Web Conf. (WWW'01)*, 2001, pp. 285–295.

[19] K. Zhou, S. Yang, and H. Zha, "Functional matrix factorizations for cold-start recommendation," in *Proc. 34th Annual ACM SIGIR Conf. (SIGIR'10)*, 2011, pp. 315–324.

[20] S. Chang, Y. Zhang, J. Tang, D. Yin, Y., M. Hasegawa-Johnson, and T. Huang, "Positive-unlabeled learning in streaming networks," in *Proc. 22nd ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD'16)*, 2016, pp. 755–764.

[21] Y. Zhang, Y. Xiong, X. Kong, and Y. Zhu, "Netcycle: Collective evolution inference in heterogeneous information networks," in *Proc. 22nd ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD'16)*, 2016, pp. 1365–1374.