Discovering Progression Stages in Trillion-Scale Behavior Logs

Kijung Shin[1], Mahdi Shafiei[2], Myunghwan Kim[2], Aastha Jain[2], Hema Raghavan[2]
[1]Carnegie Mellon University, Pittsburgh, PA, USA
[2]LinkedIn Corporation, Mountain View, CA, USA
kijungs@cs.cmu.edu,{mashafiei,mukim,asjain,hraghavan}@linkedin.com

ABSTRACT

User engagement is a key factor for the success of web services. Studying the following questions will help establishing business strategies leading to their success: *How do the behaviors of users in a web service evolve over time? To reach a certain engagement level, what are the common stages that many users go through? How can we represent the stage that each individual user lies in?*

To answer these questions, we propose a behavior model that discovers the progressions of users' behaviors from a given starting point – such as a new subscription or first experience of certain features – to a particular target stage such as a predefined engagement level of interest. Under our model, transitions over stages represent progression of users where each stage in our model is characterized by probability distributions over types of actions, frequencies of actions, and next stages to move. Each user performs actions and moves to a next stage following the probability distributions characterizing the current stage.

We also develop a fast and memory-efficient algorithm that fits our model to trillions of behavioral logs. Our algorithm scales linearly with the size of data. Especially, its distributed version implemented in the MapReduce framework successfully handles petabyte-scale data with one trillion actions.

Lastly, we show the effectiveness of our model and algorithm by applying them to real-world data from LinkedIn. We discover meaningful stages that LinkedIn users go through leading to predefined target goals. In addition, our trained models are shown to be useful for downstream tasks such as prediction of future actions.

ACM Reference Format:
Kijung Shin, Mahdi Shafiei, Myunghwan Kim, Aastha Jain, and Hema Raghavan. 2018. Discovering Progression Stages in Trillion-Scale Behavior Logs. In *WWW 2018: The 2018 Web Conference, April 23-27, 2018, Lyon, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3178876.3186182

1 INTRODUCTION

The behaviors of users of web-sites and apps change over time for reasons including temporal trends [14] and shift of personal interests [17]. When these behavioral changes are aligned with a certain direction, we observe the progression of user behavior. For example, in RateBeer, a beer review site, new users have similar tastes, but they start reviewing different types of beers as users gain experience and develop their own preferences [21, 29]. Another example is Wikipedia, where users utilize different navigation strategies throughout the information seeking process in order to reach the target information [28].

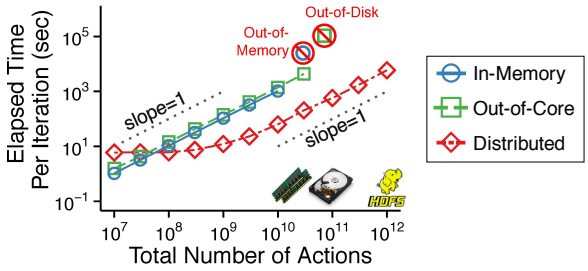


Figure 1: High scalability of our optimization algorithm. Every version of our algorithm scaled linearly with the size of the input log data. Specifically, the in-memory and out-of-core versions on a single machine handled log data with 10 billion actions and 30 billion actions, respectively. Moreover, the distributed version implemented in the MapReduce framework handled log data with one trillion actions.

Understanding such progressions is of paramount importance to providing more personalized experiences, which can lead to greater engagement and potentially increasing revenue. To revisit the RateBeer example, the site may want to advertise beers to users based on their current tastes or recommend new flavors of beers considering the next stages in the progression of the users' tastes.

While general understanding of those progression patterns is important, businesses are often interested in behavior progressions only from one state to another state due to strategic or organizational interests. For instance, a company may want to strategically focus on helping onboarding users to reach a certain level of engagement, or it may need to study users who eventually bring revenue. In another example, at LinkedIn, a professional online social network providing services including news feeds and job pages, a product manager for news feeds needs to study the progressions of users who eventually become engaged with the news feed. Another product manager for job pages may be interested in navigational patterns of users who are looking for jobs. Hence, insights about progressions of users with respect to a specific target state is very helpful for establishing practical strategies.

To summarize and better understand such progressions toward a taget state, we need to identify common stages that many users go through while interacting with a web service. For thorough summarization, these stages should capture changes in three different aspects that we describe below.

The first aspect is the change in the types of actions performed or features frequently used by users. For example, in an online social network service like LinkedIn, new users focus on making connections, while more established users with enough connections spend more time on consuming content or interacting with their

WWW 2018, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5639-8/18/04.
https://doi.org/10.1145/3178876.3186182

connections. Changes in the types of actions performed by a user will then be considered as transitions between these stages.

Another aspect to consider is how often users visit, perform an action, or use a feature in a web service. Users who visit a web service every day and those who visit it once a year may not be in the same stage even though they perform similar actions or use similar features. This distinction is particularly important because service providers typically distinguish users by their level of activity and often aim to promote user activity and engagement.

The last aspect is the direction of changes toward a given target state. New users who are getting familiar with the web service and more established users who are becoming less active may be in different stages independent of how similar the other aspects of their behavior are. That is, stages should describe not only current behavior but also transitions to future stages.

In this work, we propose a behavior model with stages characterizing all the aforementioned aspects of changes, while most existing models target only one of these aspects (e.g., only types of actions [21, 29]). Specifically, each stage in our model is characterized by a probability distribution over types of actions, a probability distribution over frequencies of actions, and transition probabilities of moving to another stage as the next stage. Progressions of users' behaviors can then be defined as transitions between these stages. That is, each user performs actions and moves to the next stage following the probability distributions describing the current stage.

Our algorithmic contribution is a fast and memory-efficient algorithm for training the parameters (i.e., probability distributions) of our model. It aims to find the parameters that best describe a given dataset. Our algorithm scales linearly with the size of the input dataset (i.e., the total number of actions) and handles extremely large datasets that do not fit in main memory. Especially, its distributed version, implemented in the MapReduce framework [10], successfully handles a petabyte-scale dataset with one trillion actions, as shown in Figure 1.

We apply our model to real-world data from LinkedIn, discovering meaningful stages that LinkedIn users go through for specific target states. For example, our model accurately captures the on-boarding stages that LinkedIn provides to new users. We also show that stage information inferred by our model is useful for downstream tasks including prediction of future actions. While the empirical evaluations of our model focus on the progressions on an online social network (i.e., LinkedIn), our model can be applied to any dataset with a series of actions by different users over time.

In summary, our main contributions are as follows:

- **Comprehensive behavior model**: we propose a probabilistic behavior model that describes progressions of users' behaviors in three different aspects (Figure 2).
- **Scalable optimization algorithm**: we propose a fast and memory-efficient algorithm that fits the parameters of our model to trillion-scale behavior logs (Figure 1).
- **Experiments with real-world data**: we show the effectiveness of our model by applying it to real-world data from LinkedIn (Tables 2 and 4)

In Section 2, we introduce our behavior model. In Section 3, we present our optimization algorithm to learn the parameters of our model. In Section 4, we discuss experimental results. After reviewing related work in Section 5, we offer conclusions in Section 6.

Table 1: Table of symbols.

Symbol	Definition
$\mathcal{U}$	set of all users
$\mathcal{A}$	set of all types of actions
Δ	set of all binned time gaps
$\mathcal{S} = \{s_1, \ldots, s_k\}$	set of all stages
n_u	number of actions done by user $u \in \mathcal{U}$
$g_u \in \{0, 1\}$	whether user $u \in \mathcal{U}$ reaches the target stage after doing n_u actions
$a_{u,j} \in \mathcal{A}$	type of the j-th action done by user $u \in \mathcal{U}$
$t_{u,j}$	timestamp of action $a_{u,j}$
$\delta_{u,j} \in \Delta$	binned time gap between $t_{u,j}$ and $t_{u,j-1}$
$\bar{a}_u$	sequence of actions performed by user $u \in \mathcal{U}$
$\bar{\delta}_u$	sequence of time gaps in $\bar{a}_u$
$\mathcal{S}_{-k}$	set of all stages excluding the target stage s_k
$z_{u,j} \in \mathcal{S}_{-k}$	stage of user u at $t_{u,j}$
$\bar{z}_u$	sequence of stages assigned to the actions in $\bar{a}_u$
$\theta_{s_i}, \phi_{s_i}, \psi_{s_i}$	probability distributions of actions, time-gaps, and transitions in stage $s_i \in \mathcal{S}_{-k}$
$\lambda_\theta, \lambda_\phi, \lambda_\psi$	hyperparameters regarding the prior distributions of $\theta_{s_i}, \phi_{s_i}, \psi_{s_i}$
ξ	hyperparameter for the initialization step

2 BEHAVIOR MODEL

In this section, we describe our behavior model for capturing progressive changes in users' behaviors. The symbols used to describe our model are listed in Table 1.

2.1 Notations and Model Description

Consider a set of users doing a sequence of actions. Let $\mathcal{U}$ be the set of users and $\mathcal{A}$ be the set of types of actions that can be done by the users. We bin the time gap between each two consecutive actions and use Δ to indicate the set of potential gap sizes.

Let $\mathcal{S} = \{s_1, s_2, ..., s_k\}$ be the set of k stages that the users may go through between the **starting stage** s_1 and the **target stage** s_k. Two stages s_1 and s_k are treated as special cases. That is, we have predefined conditions that determine whether each user has reached them (e.g., users reach the starting stage s_1 if they join LinkedIn and reach the target stage s_k as soon as they reach a certain number of connections).

We assume the monotonicity of the stages in $\mathcal{S}$ to model progression 'towards' the goal stage s_k. That is, for any i and j satisfying $1 \le i < j \le k$, users can transit from stage s_i to stage s_j but not in the opposite direction. This constraint, however, does not enforce that all users follow the same path towards the goal stage. Under our model, users are allowed to skip any intermediate stages.

Let $\mathcal{S}_{-k} = \{s_1, ..., s_{k-1}\}$ be the set of non-target stages. Each non-target stage $s_i \in \mathcal{S}_{-k}$ is characterized by probability distributions over types of actions, time gaps, and transitions from the stage, which are defined as follows:

- $\theta_{s_i} \in \mathbb{R}^{|\mathcal{A}|}$: probability distribution over types of actions performed by users in stage s_i.
- $\phi_{s_i} \in \mathbb{R}^{|\Delta|}$: probability distribution over time gaps between two consecutive actions performed by users in stage s_i.
- $\psi_{s_i} \in \mathbb{R}^{k-i+1}$: transition probability distribution over next stages moving from stage s_i before performing each action.

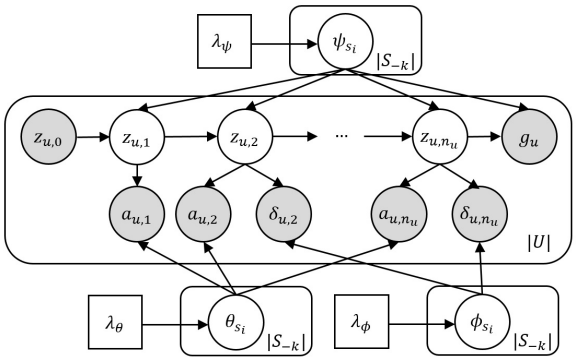


Figure 2: Plate notation [8] for our behavior model. Observed variables are colored grey, and unobserved variables are colored white. The type of each action (i.e., each $a_{u,j}$) and the time-gap between each two consecutive actions (i.e., each $\delta_{u,j}$) depend on the current stage (i.e., each $z_{u,j}$) and the probability distributions characterizing each stage (i.e., each θ_{s_i} and ϕ_{s_i}). Transitions between stages depend on the transition probability distribution in each stage (i.e., each ψ_{s_i}). The starting stage (each $z_{u,0}$) and whether the target stage is reached by each user (i.e., each g_u) are observable.

We assume a symmetric Dirichlet prior [4] over θ_{s_i}, ϕ_{s_i}, and ψ_{s_i}:

$$\theta_{s_i} \sim \text{Dirichlet}(1+\lambda_\theta), \quad \phi_{s_i} \sim \text{Dirichlet}(1+\lambda_\phi),$$
$$\text{and} \quad \psi_{s_i} \sim \text{Dirichlet}(1+\lambda_\psi),$$

where λ_θ, λ_ϕ, and λ_ψ are hyperparameters.

We only consider actions performed after reaching the starting stage and before reaching the target stage (in cases where a user reaches it). For each user $u \in \mathcal{U}$, let n_u be the number of such actions by u, and let $g_u \in \{0,1\}$ indicate whether u reaches the target stage after performing the n_u actions. Then, we use $a_{u,j} \in \mathcal{A}$ to denote the type of the j-th action of u and use $t_{u,j}$ to denote the time when that action is performed. In addition, $\delta_{u,j} \in \Delta$ indicates the binned time gap between $t_{u,j}$ and $t_{u,j-1}$; and $z_{u,j}$ indicates the stage of u at $t_{u,j}$. For simplicity, we use $\bar{a}_u = (a_{u,1}, ..., a_{u,n_u})$, $\bar{\delta}_u = (\delta_{u,2}, ..., \delta_{u,n_u})$, and $\bar{z}_u = (z_{u,1}, ..., z_{u,n_u})$ to denote the sequences of actions, time-gaps, and assigned stages for user u.

2.2 Generative Process

The generative process of our model is described in Figure 2. For user $u \in \mathcal{U}$ who is in stage $z_{u,j-1}$ after doing her j-th action, she

(1) moves to stage $z_{u,j}$ (which can be the same as $z_{u,j-1}$), where

$$z_{u,j} \sim \text{Multinomial}(\psi_{z_{u,j-1}})$$

(2) performs an action $a_{u,j}$ after a time gap $\delta_{u,j}$, where

$$a_{u,j} \sim \text{Multinomial}(\theta_{z_{u,j}}), \quad \text{and} \quad \delta_{u,j} \sim \text{Multinomial}(\phi_{z_{u,j}}).$$

In the beginning, each user $u \in \mathcal{U}$ moves from the starting stage s_1 (i.e., $z_{u,0} = s_1$), and the time-gap for her first action is ignored since there can be no previous action. Each user repeats this process until she reaches the target stage.

As seen in Figure 2, the following are observable from log data: (a) whether each user had reached the target stage before the log data were collected (i.e., $\{g_u\}_{u\in\mathcal{U}}$), (b) the number of actions performed by each user between the starting stage (inclusive) and the target stage (exclusive) before the log data were collected (i.e., $\{n_u\}_{u\in\mathcal{U}}$), (c) the type of each action (i.e., $\{\bar{a}_u\}_{u\in\mathcal{U}}$), and (d) the time gap between each two consecutive actions (i.e., $\{\bar{\delta}_u\}_{u\in\mathcal{U}}$).

3 MODEL TRAINING

In this section, we discuss training the parameters of our model. We present the objective function in Section 3.1 and our optimization algorithm in Section 3.2. We extend the algorithm to external-memory, multi-core, and distributed settings in Section 3.3 and discuss its time and space complexities in Section 3.4.

3.1 Training Objective

Given an input dataset consisting of sequences of actions $\{\bar{a}_u\}_{u\in\mathcal{U}}$, sequences of time-gaps $\{\bar{\delta}_u\}_{u\in\mathcal{U}}$ and whether users reach the target stage $\{g_u\}_{u\in\mathcal{U}}$, our objective is to find the most probable sequences of stages $\{\bar{z}_u\}_{u\in\mathcal{U}}$ as well as probability distributions of actions $\{\theta_{s_i}\}_{s_i\in\mathcal{S}_{-k}}$, time-gaps $\{\phi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}$, and transitions $\{\psi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}$. Maximizing the posterior probability of the parameters of our model given the observed states, written as

$$p(\{\theta_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, \{\phi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, \{\psi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, \{\bar{z}_u\}_{u\in\mathcal{U}} \\ |\{\bar{a}_u\}_{u\in\mathcal{U}}, \{\bar{\delta}_u\}_{u\in\mathcal{U}}, \{g_u\}_{u\in\mathcal{U}}, \lambda_\theta, \lambda_\phi, \lambda_\psi), \quad (1)$$

is equivalent to maximizing the following objective function f:

$$f(\{\bar{z}_u\}_{u\in\mathcal{U}}, \{\theta_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, \{\phi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, \{\psi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}) \\ := \prod_{u\in\mathcal{U}} f_u(\bar{z}_u, \{\theta_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, \{\phi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, \{\psi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}) \\ \times \prod_{s_i\in\mathcal{S}_{-k}} \left(p(\theta_{s_i}|\lambda_\theta) \times p(\phi_{s_i}|\lambda_\phi) \times p(\psi_{s_i}|\lambda_\psi)\right), \quad (2)$$

where $f_u(\bar{z}_u, \{\theta_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, \{\phi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, \{\psi_{s_i}\}_{s_i\in\mathcal{S}_{-k}})$

$$:= p(\bar{a}_u|\{\theta_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, \bar{z}_u) \times p(\bar{\delta}_u|\{\phi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, \bar{z}_u) \\ \times p(\bar{z}_u|\{\psi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}) \times p(g_u|\{\psi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, \bar{z}_u) \\ = \left(\prod_{j=1}^{n_u} p(a_{u,j}|z_{u,j}, \theta_{z_{u,j}})\right) \times \left(\prod_{j=2}^{n_u} p(\delta_{u,j}|z_{u,j}, \phi_{z_{u,j}})\right) \\ \times \left(\prod_{j=1}^{n_u} p(z_{u,j}|z_{u,j-1}, \psi_{z_{u,j-1}})\right) \times p(g_u|z_{u,n_u}, \psi_{z_{u,n_u}}). \quad (3)$$

Notice that our objective function is non-convex and may have multiple local optima.

3.2 Optimization Algorithm

We propose an iterative refinement algorithm for optimizing our objective function f (i.e., Eq. (2)). Different from general optimization algorithms for any graphical model (e.g., EM [11]), our algorithm fully utilizes the dependency structure of our model for fast, memory-efficient, and parallel computation.

Our algorithm consists of the following three steps:

- **Initialization step** (Section 3.2.3): We initialize the probability distributions θ_{s_i}, ϕ_{s_i}, and ψ_{s_i} in every stage $s_i \in \mathcal{S}_{-k}$.
- **Assignment step** (Section 3.2.1): Given the current probability distributions θ_{s_i}, ϕ_{s_i}, and ψ_{s_i} in every stage $s_i \in \mathcal{S}_{-k}$, we update the stage assignments $\bar{z}_u$ of every user $u \in \mathcal{U}$ so that our objective function f is maximized.

Algorithm 1 Assignment Step

Input: log data: $\{\bar{a}_u\}_{u\in\mathcal{U}}$,$\{\bar{\delta}_u\}_{u\in\mathcal{U}}$,$\{g_u\}_{u\in\mathcal{U}}$
prob. distributions: $\{\theta_{s_i}\}_{s_i\in\mathcal{S}_{-k}}$, $\{\phi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}$, $\{\psi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}$
Output: stage assignments: $\{\bar{z}_u\}_{u\in\mathcal{U}}$

1: **for** each user $u \in \mathcal{U}$ **do**
2: **for** each stage $s_i \in \mathcal{S}_{-k}$ **do**
3: **if** $g_u = 0$ **then**
4: $f_{u,n_u}(s_i) \leftarrow \theta_{s_i}(a_{u,n_u}) \cdot \phi_{s_i}(\delta_{u,n_u}) \cdot (1 - \psi_{s_i}(s_k))$
5: **else**
6: $f_{u,n_u}(s_i) \leftarrow \theta_{s_i}(a_{u,n_u}) \cdot \phi_{s_i}(\delta_{u,n_u}) \cdot \psi_{s_i}(s_k)$
7: **for** $j = n_u - 1, \ldots, 1$ **do**
8: **for** each stage $s_i \in \mathcal{S}_{-k}$ **do**
9: $s_l \leftarrow \arg\max_{s_m: i\le m<k} \left(\psi_{s_i}(s_m) \cdot f_{u,j+1}(s_m)\right)$
10: **if** $j = 1$ **then**
11: $f_{u,j}(s_i) \leftarrow \theta_{s_i}(a_{u,j}) \cdot \psi_{s_i}(s_l) \cdot f_{u,j+1}(s_l)$
12: **else**
13: $f_{u,j}(s_i) \leftarrow \theta_{s_i}(a_{u,j}) \cdot \phi_{s_i}(\delta_{u,j}) \cdot \psi_{s_i}(s_l) \cdot f_{u,j+1}(s_l)$
14: $h_{u,j}(s_i) \leftarrow s_l$
15: $z_{u,1} \leftarrow \arg\max_{s_i\in\mathcal{S}_{-k}} \left(\psi_{s_1}(s_i) \cdot f_{u,1}(s_i)\right)$
16: **for** $j = 2, \ldots, n_u$ **do**
17: $z_{u,j} \leftarrow h_{u,j-1}(z_{u,j-1})$
18: **return** $\{\bar{z}_u\}_{u\in\mathcal{U}}$

- **Update step** (Section 3.2.2): Given the current stage assignments $\bar{z}_u$ of every user $u \in \mathcal{U}$, we update the probability distributions θ_{s_i}, ϕ_{s_i}, and ψ_{s_i} in every stage $s_i \in \mathcal{S}_{-k}$ so that our objective function f is maximized.

The initialization step is performed once initially. Then, the assignment and update steps are repeated until our optimization function f converges. Each assignment step and each update step are guaranteed to improve the objective function. Therefore, our algorithm is guaranteed to find a local optima.

We describe each step in detail in the following subsections. For ease of explanation, we first present the assignment and update steps then present the initialization step.

3.2.1 Assignment Step. In this step, we maximize the objective function f by updating the stage assignments $\{\bar{z}_u\}_{u\in\mathcal{U}}$, while fixing the probability distributions $\{\theta_{s_i}\}_{s_i\in\mathcal{S}_{-k}}$, $\{\phi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}$, and $\{\psi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}$ to their current values. Once the probability distributions are fixed, the stage assignment $\bar{z}_u$ of each user $u \in \mathcal{U}$ can be optimized independently by maximizing f_u (i.e., Eq. (3)).

For each user $u \in \mathcal{U}$, we use dynamic programming [26] to update $\bar{z}_u$, as described in detail in Algorithm 1. In the algorithm, $f_{u,j}(s_i)$ denotes the maximum posterior probability of the j-th or later actions and time gaps for user u given that $z_{u,j} = s_i$. That is, $f_{u,j}(s_i)$ is defined as follows:

$$f_{u,j}(s_i) := \max_{\{z_{u,l}\}_{l>j}} \Big(p(\{a_{u,l}\}_{l\ge j}|\{\theta_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, \{z_{u,l}\}_{l>j}, z_{u,j} = s_i) \times p(\{\delta_{u,l}\}_{l\ge\max(j,2)}|\{\theta_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, \{z_{u,l}\}_{l>j}, z_{u,j} = s_i) \times p(g_u|\{\psi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, \{z_{u,l}\}_{l>j}, z_{u,j} = s_i) \times p(\{z_{u,l}\}_{l>j}|\{\psi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}, z_{u,j} = s_i)\Big). \quad (4)$$

We observe that $f_{u,j}(s_i)$ is computed easily from $\{f_{u,j+1}(s_i)\}_{s_i\in\mathcal{S}_{-k}}$ (lines 9-13). Our algorithm exploits this observation by computing $\{f_{u,j}(s_i)\}_{s_i\in\mathcal{S}_{-k}}$ in the decreasing order of j from n_u to 1 (lines 2-14). Note that the stage assignments maximizing each $f_{u,j}(s_i)$ is stored in $h_{u,j}(s_i)$ (line 14). Specifically, $h_{u,j}(s_i) = s_l$ means that the stage assignments maximizing $f_{u,j}(s_i)$ are $z_{u,j+1} = s_l$ and those maximizing $f_{u,j+1}(s_l)$.

Once we have computed $\{f_{u,1}(s_i)\}_{s_i\in\mathcal{S}_{-k}}$, which indicates the maximum posterior probabilities of everything except the initial transition, we can easily maximize f_u by finding $z_{u,1}$ maximizing $\psi_{s_1}(z_{u,1}) \cdot f_{u,1}(z_{u,1})$ (line 15). The stage assignments $\{\bar{z}_u\}_{u\in\mathcal{U}}$ maximizing f_u can be obtained from $\{\{h_{u,j}(s_i)\}_{s_i\in\mathcal{S}_{-k}}\}_{1\le j\le n_u}$ by following the path backward starting from $z_{u,1}$ (lines 16-17).

3.2.2 Update Step. In this step, we maximize f by updating the probability distributions $\{\theta_{s_i}\}_{s_i\in\mathcal{S}_{-k}}$, $\{\phi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}$, and $\{\psi_{s_i}\}_{s_i\in\mathcal{S}_{-k}}$, while fixing the stage assignments $\{\bar{z}_u\}_{u\in\mathcal{U}}$ to their current values.

To this end, we decompose our objective function based on the probability distribution that each term depends on as follows:

$$f = \prod\nolimits_{s_i\in\mathcal{S}_{-k}} \Big(f_\theta(\theta_{s_i}) \times f_\phi(\phi_{s_i}) \times f_\psi(\psi_{s_i})\Big), \text{ where}$$

$$f_\theta(\theta_{s_i}) = p(\theta_{s_i}|\lambda_\theta) \times \prod_{u\in\mathcal{U}} \prod_{j:z_{u,j}=s_i} p(a_{u,j}|z_{u,j}, \theta_{z_{u,j}}), \quad (5)$$

$$f_\phi(\phi_{s_i}) = p(\phi_{s_i}|\lambda_\phi) \times \prod_{u\in\mathcal{U}} \prod_{j\ge 2:z_{u,j}=s_i} p(\delta_{u,j}|z_{u,j}, \phi_{z_{u,j}}), \quad (6)$$

$$f_\psi(\psi_{s_i}) = p(\psi_{s_i}|\lambda_\psi) \times \prod_{u\in\mathcal{U}} \Big(p(g_u|z_{u,n_u}, \psi_{z_{u,n_u}}) \times \prod_{j:z_{u,j-1}=s_i} p(z_{u,j}|z_{u,j-1}, \psi_{z_{u,j-1}})\Big). \quad (7)$$

Then, we update each probability distribution independently so that the terms depending on it are maximized. Notice that this update has an analytical solution.

Specifically, for each stage $s_i \in \mathcal{S}_{-k}$, we update the action type distribution θ_{s_i} so that $f_\theta(\theta_{s_i})$ (i.e., Eq. (5)) is maximized. For each type of action a, the probability $\theta_{s_i}(a)$ is updated as follows:

$$\theta_{s_i}(a) \leftarrow \frac{\lambda_\theta + c_{\mathcal{A}}(s_i, a)}{|\mathcal{A}|\lambda_\theta + \sum_{a'\in\mathcal{A}} c_{\mathcal{A}}(s_i, a')}, \quad (8)$$

where $c_{\mathcal{A}}(s_i, a) := \sum_{u\in\mathcal{U}} |\{1 \le j \le n_u : a_{u,j} = a \wedge z_{u,j} = s_i\}|$.

Likewise, for each stage $s_i \in \mathcal{S}_{-k}$, we update the time gap distribution ϕ_{s_i} so that $f_\phi(\phi_{s_i})$ (i.e., Eq. (6)) is maximized. For each time gap $\delta \in \Delta$, the probability $\phi_{s_i}(\delta)$ is updated as follows:

$$\phi_{s_i}(\delta) \leftarrow \frac{\lambda_\phi + c_\Delta(s_i, \delta)}{|\Delta|\lambda_\phi + \sum_{\delta'\in\Delta} c_\Delta(s_i, \delta')}, \quad (9)$$

where $c_\Delta(s_i, \delta) := \sum_{u\in\mathcal{U}} |\{2 \le j \le n_u : \delta_{u,j} = \delta \wedge z_{u,j} = s_i\}|$.

Lastly, for each stage $s_i \in \mathcal{S}_{-k}$, we update the transition probability ψ_{s_i} so that $f_\psi(\psi_{s_i})$ (i.e., Eq. (7)) is maximized. We update the transition probability $\psi_{s_i}(s_k)$ from s_i to the target stage s_k as follows:

$$\psi_{s_i}(s_k) \leftarrow \frac{\lambda_\psi + c_g(s_i)}{(k - i + 1)\lambda_\psi + c_{\mathcal{S}}(s_i)}, \quad (10)$$

where $c_g(s_i) := \sum_{u\in\mathcal{U}} \mathbf{1}(z_{u,n_u} = s_i \wedge g_u = 1)$ and $c_{\mathcal{S}}(s_i) := \sum_{u\in\mathcal{U}} |\{0 \le j \le n_u : z_{u,j} = s_i\}|$. Then, for each non-target stages $s_i, s_l \in \mathcal{S}_{-k}$ where $i \le l$, we update the probability $\psi_{s_i}(s_l)$

of the transition from s_i to s_l as follows:

$$\psi_{s_i}(s_l) \leftarrow (1-\psi_{s_i}(s_k)) \times \frac{\lambda_\psi + c_{\mathcal{S}}(s_i, s_l)}{(k-i)\lambda_\psi + \sum_{m=i}^{k-1} c_{\mathcal{S}}(s_i, s_m)}, \quad (11)$$

where $c_{\mathcal{S}}(s_i, s_l) := \sum_{u \in \mathcal{U}} |\{1 \le j \le n_u : z_{u,j-1} = s_i \wedge z_{u,j} = s_l\}|$.

3.2.3 Initialization Step. Since our objective function f (i.e., Eq. (2)) is non-convex, the solution found by our optimization algorithm and its speed of convergence (i.e., the number of iterations required for convergence) depend on initial parameter values. In this section, we present an initialization method that works well in practice, as we experimentally show in Section 4.5 and Section 4.6.

First, we choose a subset of users and decide their stage variables in a simple way. Specifically, let $\mathcal{U}'$ be the set of users that has performed at least ξ ($\ge k-1$) actions, where ξ is a hyperparameter. For each user $u \in \mathcal{U}'$, we divide her n_u stage variables into $(k-1)$ continuous segments of equal length. Then, we assign stage s_i to the variables in each i-th segment.[1] In this process, the users with a small number of actions (i.e., $\mathcal{U} - \mathcal{U}'$) are ignored since they are less likely to have gone through all $(k-1)$ stages.

Once the stage variables of $\mathcal{U}'$ are set, the action-type probability distributions $\{\theta_{s_i}\}_{s_i \in \mathcal{S}_{-k}}$ and the time-gap probability distributions $\{\phi_{s_i}\}_{s_i \in \mathcal{S}_{-k}}$ are initialized by Eq. (8) and Eq. (9) with $\mathcal{U}'$ instead of $\mathcal{U}$. We initialize transition probability distributions $\{\psi_{s_i}\}_{s_i \in \mathcal{S}_{-k}}$ so that the transition probability decreases exponentially with the distance from the current stage.[2]

3.3 Extensions to External-memory, Multi-core, and Distributed Settings

In this section, we extend our optimization algorithm, described in Section 3.2, to the external-memory, multi-core, and distributed settings without affecting its outputs. We focus on extending the assignment and update steps. The ideas are also applicable to the initialization step due to its similarity to the other steps.

3.3.1 External-memory Settings. Our optimization algorithm does not require loading the entire input data (i.e., $\{\bar{a}_u\}_{u \in \mathcal{U}}$, $\{\bar{\delta}_u\}_{u \in \mathcal{U}}$, $\{g_u\}_{u \in \mathcal{U}}$, and $\{\bar{z}_u\}_{u \in \mathcal{U}}$) in main memory at once. Instead, it can run by loading the data for each user into main memory, while storing the data for the other users in external memory (e.g., disk). This is particularly useful when the entire input data is too large to fit in main memory.

We assume that the input data are stored in external memory sorted by user ids (by any external sorting algorithm). We sequentially read input data until we load all data for a user into main memory. Then, we assign the user's stages (lines 2-17 of Algorithm 1). Based on these stages, we add the user's contributions[3] to the counts (e.g., $c_{\mathcal{A}}(s_i, a)$ and $c_{\mathcal{S}}(s_i, s_l)$) in Eq. (8)-Eq. (11) for every action type, time gap, and stage. Then, we free the memory space allocated for the current user and move to the next user by continuing reading the input data. After processing the last user, we compute the numerators and denominators of Eq. (8)-Eq. (11) for every action type, time gap, and stage, simply by adding constants to the sums of the users' contributions. We update the probability distributions by the equations and move to the next iteration.

In Section 4.4, we experimentally show that this out-of-core processing using external memory significantly reduces the main memory requirements with a slight compromise in speed.

3.3.2 Multi-core Settings. Our optimization algorithm is easily parallelized in multi-core settings. In the assignment step, the stage assignment for one user (lines 2-17 of Algorithm 1) does not depend on that for the other users. Thus, stage assignments for different users can be run in parallel using multiple threads. Likewise, in the update step, the contributions[3] of one user to the counts (e.g., $c_{\mathcal{A}}(s_i, a)$ and $c_{\mathcal{S}}(s_i, s_l)$) in Eq. (8)-Eq. (11) do not depend on those of the other users. Thus, computing the contributions of different users also can be run in parallel using multiple threads.

In Section 4.4, we experimentally show that speed-up by this parallelization is near linear to the number of threads. Notice that this parallel processing performs the same computation as the serial processing and thus has no effect on the outputs of our algorithm.

3.3.3 Distributed Settings. We combine the extensions in the previous sections for distributed settings. We assume that the input data are distributed across machines so that (a) all data for the same user are stored in one machine and (b) the data in each machine are sorted by user id. In MapReduce [10], for example, this can be done by simply shuffling the data by user ids. Each machine sums up the contributions[3] of the assigned users to the counts (e.g., $c_{\mathcal{A}}(s_i, a)$ and $c_{\mathcal{S}}(s_i, s_l)$) in Eq. (8)-Eq. (11) by sequentially reading the assigned input data, as in Section 3.3.1. Then, the contributions are gathered and summed up in one machine, which then updates all probability distributions by Eq. (8)-Eq. (11) and broadcasts them to all other machines so that they can be used in the next iteration.

In Section 4.4, we experimentally show that the MapReduce implementation of our optimization algorithm successfully handles petabyte-scale data with one trillion actions.

3.4 Complexity Analyses

In this section, we analyze the time complexity and memory requirement of our optimization algorithm. We first show that the time complexity of our algorithm is linear in the total number of actions. Then, we show that its memory requirement is linear in the maximum number of actions among all users.

Theorem 3.1 (Time Complexity). *Let $N = \sum_{u \in \mathcal{U}} n_u$ be the total number of actions and T be the number of iterations. If $N = \Omega(|\mathcal{A}|/k + |\Delta|/k)$, then the time complexity of our optimization algorithm is $O(TNk^2)$.*

Sketch of Proof. The time complexity of the assignment step (i.e., Algorithm 1) is $O(Nk^2)$ because line 9, which takes $O(k)$, is executed $O(\sum_{u \in \mathcal{U}} n_u k) = O(Nk)$ times.

The time complexity of the update step is $O(N + |\mathcal{A}|k + |\Delta|k + k^2)$. For each of $\{(u, j) : u \in \mathcal{U}, 0 \le j \le n_u\}$, whose size is N, we need to increase a constant number of counts (e.g., $c_{\mathcal{A}}(s_i, a)$ and $c_{\mathcal{S}}(s_i, s_l)$) in Eq. (8)-Eq. (11). Updating the probability distributions from the counts by Eq. (8)-Eq. (11) takes $O(|\mathcal{A}|k + |\Delta|k + k^2)$.

Thus, one iteration (i.e., running the assignment and update steps once) takes $O(Nk^2 + |\mathcal{A}|k + |\Delta|k + k^2)$, which is $O(Nk^2)$ by our assumption. Hence, the total time complexity is $O(TNk^2)$. ■

[1] i.e., for each $1 \le j \le n_u$, $z_{u,j}$ is set to stage s_i satisfying $\frac{n_u(i-1)}{(k-1)} < j \le \frac{n_u i}{(k-1)}$.

[2] i.e., for each stage s_i, $\psi_{s_i}(s_l) \leftarrow \frac{2^{-(l-i)}}{\sum_{m=i}^{k} 2^{-(m-i)}}$.

[3] e.g., the contribution of user u to $c_{\mathcal{A}}(s_i, a)$ is $|\{1 \le j \le n_u : a_{u,j} = a \wedge z_{u,j} = s_i\}|$ and that to $c_{\Delta}(s_i, \delta)$ is $|\{2 \le j \le n_u : \delta_{u,j} = \delta \wedge z_{u,j} = s_i\}|$.

Theorem 3.2 (Memory Requirement). *If* $\max_{u \in \mathcal{U}} n_u = \Omega(|\mathcal{A}| + |\Delta| + k)$*, then the memory requirement of our optimization algorithm is* $O(k \max_{u \in \mathcal{U}} n_u)$.

Sketch of Proof. As explained in Section 3.3.1, we need to load the input data for each user u (i.e., $\bar{a}_u$, $\bar{\delta}_u$, and g_u), whose sizes are $O(n_u)$, into main memory at a time. To assign the n_u stage variables for u (by lines 2-17 of Algorithm 1), we need $O(kn_u)$ memory space for maintaining $\{\{f_{u,j}(s_i)\}_{s_i \in \mathcal{S}_{-k}}\}_{1 \le j \le n_u}$, $\{\{h_{u,j}(s_i)\}_{s_i \in \mathcal{S}_{-k}}\}_{1 \le j \le n_u}$, and the assigned stage variables $\bar{z}_u$. We also need $O(|\mathcal{A}|k + |\Delta|k + k^2)$ memory space for maintaining the probability distributions (i.e., $\{\theta_{s_i}\}_{s_i \in \mathcal{S}_{-k}}$, $\{\phi_{s_i}\}_{s_i \in \mathcal{S}_{-k}}$, $\{\psi_{s_i}\}_{s_i \in \mathcal{S}_{-k}}$) and the counts (e.g., $c_{\mathcal{A}}(s_i, a)$ and $c_{\mathcal{S}}(s_i, s_l)$) in Eq. (8)-Eq. (11). Therefore, the total memory requirement is $O(\max_{u \in \mathcal{U}}(kn_u) + |\mathcal{A}|k + |\Delta|k + k^2)$, which is $O(k \max_{u \in \mathcal{U}} n_u)$ by our assumption. ■

4 EXPERIMENTS

We review our experiments for answering the following questions:

- **Q1. Effectiveness**: Do our model and algorithm discover meaningful progression stages in real-world data?
- **Q2. Applicability**: Are our trained models useful for downstream tasks such as prediction of future actions?

We also verify that our optimization algorithm is effective by answering the following questions:

- **Q3. Scalability**: Does our algorithm scale linearly with the size of the input data? Can our algorithm handle a dataset with trillions of actions?
- **Q4. Convergence**: How rapidly does our algorithm converge?
- **Q5. Identifiability**: How accurately does our algorithm estimate ground-truth parameters?

4.1 Experimental Settings

Data: We used a dataset provided by LinkedIn. The dataset is the log of the actions performed before mid June of 2017 by a subset of LinkedIn members who joined LinkedIn after mid April of 2017. The dataset has 22 types of actions including the followings:

- visit: **visit** LinkedIn.com.
- profile-edit: **edit** one's **profile**.
- profile-view: **view** another member's **profile**.
- invite-abook: **invite** a non-member to LinkedIn on a page that lists some emails imported from one's **a**ddress **book**.
- conn-abook: send a **conn**ection request to a member on a page that lists some members imported from one's **a**ddress **book**.
- conn-ins: send a **conn**ection request to a member on a page that lists some members in the same **ins**titution.
- conn-rec: send a **conn**ection request to a member on a page that lists some members **rec**ommended by LinkedIn.
- conn-search: send a **conn**ection request to a member on a page that shows the results of one's **search**.
- conn-profile: send a **conn**ection request to a member on the member's **profile** page.
- conn-other: send a **conn**ection request to a member by means **other** than those explained above.
- conn-accept: **accept** a **conn**ection request that one received.
- job-view: **view** a **job** posting.
- message: send a **message** to a member.

The time gaps between each consecutive actions performed by the same user are binned into second (within few seconds), minute, hour, day, week, month, and over-a-month (over a month).

Starting and Target Stages: We used the following two settings:

- connected: The starting stage is defined as joining LinkedIn, and the target stage is defined as reaching 30 connections. The number of actions between the stages is about 500 millions.
- engaged: The starting stage is defined as having 30 or more connections, and the target stage is defined as visiting LinkedIn 4 days in a week for 5 (not necessarily consecutive) weeks. The number of actions between the stages is about 150 millions.

Implementations: We implemented our optimization algorithm in Java 1.7 and Hadoop 2.6.1. [1]. We used SMILE v1.3 [2] for logistic regression [22] and k-means++ [3].

4.2 Q1. Effectiveness: Descriptive Results

We present results demonstrating intuitive patterns extracted by our model for both settings of 'connected' and 'engaged'.

As shown in Table 2, our model [4] extracted the following reasonable latent stages for the connected setting:

- profile (stage 1): The first learned stage shows the behaviors of creating new profiles right after new members join the service.
- on-boarding (stages 2-4): The next three stages describe the optional on-boarding process provided to new members. During the process, LinkedIn provides new members with a list of other members to connect to and non-members to invite. New members can rapidly (notice that the most probable time gap is second) send connection requests and invitations.
- poke (stage 5): Members start poking other services and typically enjoying viewing other members' profiles.
- grow (stage 6): Members grow their networks by searching for other members and using connection recommendation services.
- explore (stage 7): In addition to growing their networks, members start consuming content. They visit LinkedIn for browsing content (notice that visit is the most probable action). They also start seeing job postings. Bursts of actions are reduced.

Notice that not every member goes through every stage. For example, many members in the profile stage skip the optional on-boarding process and jump directly to the poke or explore stage.

For the engaged setting, the grow and explore stages (i.e., the last two stages extracted for the connected setting) were subdivided into multiple stages by our model [5] as follows:

- active-grow (stages 1-4): In the first four stages, members actively grow their networks by different means including connecting from profiles (ag-1), connecting from search (ag-2), importing address books (ag-3), and others (ag-4).
- passive-grow and explore (stages 5-7): In the stage pg-1, members passively grow their networks by relying on connection recommendation services. The next stage, pg-2, captures the transition to the explore stage.

Once stages and progression of each user is learned, we can answer simple queries like "what is the stage that most members skip?" or perform post-processing to gain more detailed insights.

[4] with $k = 8$, $\lambda_\theta = \lambda_\phi = \lambda_\psi = 0.1$, and $\xi = 30$.
[5] with $k = 8$, $\lambda_\theta = \lambda_\phi = \lambda_\psi = 0.1$, and $\xi = 50$.

Table 2: Latent stages extracted by our model and algorithm from the LinkedIn dataset. The first table presents the stages after joining the service and before reaching 30 connections. Members (a) create their profiles, (b) go through on-boarding processes, (c) poke the service, (d) grow their networks, and finally (e) explore the service. The second table presents the stages after reaching 30 connections and before being engaged. Members (a) actively grow their networks by various means, (b) passively grow their networks relying on recommendation services, and finally (c) explore the service. For each stage, we list the three most probable types of actions, time gaps, and transitions, with their probabilities (∗∗∗∗: 80%-100%, ∗ ∗ ∗: 60%-80%, ∗∗: 30%-60%, ∗: 10%-30%). See Section 4.1 for descriptions of the types of actions, and see Section 4.2 for explanation of the stages.

Stages	**profile** (s_1)	**on-boarding**			**poke** (s_5)	**grow** (s_6)	**explore** (s_7)
		ob-1 (s_2)	**ob-2** (s_3)	**ob-3** (s_4)			
Actions (θ_{s_i})	profile-edit (∗∗) visit (∗∗) conn-accept	conn-ins (∗ ∗ ∗) conn-other (∗) invite-abook	invite-abook (∗∗∗∗) conn-abook (∗) conn-ins	conn-abook (∗∗∗∗) conn-ins (∗) invite-abook	profile-view (∗∗) visit (∗) profile-edit	conn-rec (∗ ∗ ∗) conn-search (∗) profile-view	visit (∗ ∗ ∗) job-view (∗) profile-view (∗)
Time-gaps (ϕ_{s_i})	hour (∗∗) minute (∗∗) second (∗)	second (∗∗∗∗) minute hour	second (∗∗∗∗) minute	second (∗∗∗∗) hour minute	minute (∗∗) hour (∗) second	minute (∗ ∗ ∗) second (∗) hour	hour (∗∗) minute (∗) day (∗)
Transitions (ψ_{s_i})	profile (∗∗∗∗) explore poke	ob-1 (∗∗∗∗) explore poke	ob-2 (∗∗∗∗) ob-3 explore	ob-3 (∗∗∗∗) poke explore	poke (∗∗∗∗) explore grow	grow (∗∗∗∗) explore target	explore (∗∗∗∗) target

Stages	**active-grow**				**passive-grow**		**explore** (s_7)
	ag-1 (s_1)	**ag-2** (s_2)	**ag-3** (s_3)	**ag-4** (s_4)	**pg-1** (s_5)	**pg-2** (s_6)	
Actions (θ_{s_i})	profile-view (∗∗) visit (∗) conn-profile	conn-search (∗∗) conn-rec (∗) message (∗)	invite-abook (∗∗) conn-abook (∗∗) conn-rec	conn-other (∗) conn-rec (∗) conn-abook (∗)	conn-rec (∗∗∗∗) profile-view visit	conn-rec (∗∗) profile-view (∗) visit (∗)	visit (∗∗) profile-view (∗) job-view
Time-gaps (ϕ_{s_i})	minute (∗ ∗ ∗) hour (∗) second (∗)	minute (∗ ∗ ∗) second (∗) hour	second (∗∗∗∗) minute	second (∗∗) minute (∗∗) hour	second (∗ ∗ ∗) minute (∗) hour	minute (∗ ∗ ∗) hour (∗) second (∗)	minute (∗∗) hour (∗∗) day (∗)
Transitions (ψ_{s_i})	ag-1 (∗∗∗∗) explore pg-2	ag-2 (∗∗∗∗) explore pg-2	ag-3 (∗∗∗∗) explore ag-4	ag-4 (∗∗∗∗) explore pg-2	pg-1 (∗∗∗∗) pg-2	pg-2 (∗∗∗∗) explore	explore (∗∗∗∗) target

For example, we can analyze the most discriminative sequence of stages for a particular cohort to achieve a given target. Here, we use a cohort as the members who pass through the explore stage (s_7). To extract such paths, for a set of stages $S' \subseteq S = \{s_2, \cdots, s_6\}$, we define the *discriminative score* as:

$$\frac{P(\text{reach target} \mid \text{pass through } s_7 \text{ and all of } S')}{P(\text{reach target} \mid \text{pass through } s_7 \text{ but not all of } S')}.$$

We find the top 3 discriminative sequences of stages for a given target and present the results in Table 3. Interestingly, they are very different depending on targets. In the 'connected' setting, passing through either the grow or poke stage is important, while progressing step by step is more crucial for the 'engaged' setting. The connected setting results are intuitive because exploring other content does not necessarily help reaching the target stage without continuous engagement with networking components. On the other hand, the results from the engaged setting imply that exposure to various networking channels is helpful for longer engagement.

Table 3: Top-3 discriminative paths for those who pass through the explore stage.

Target setting	Top-3 paths (discriminative score)
connected	grow (11.3) poke → grow (11.2) poke (8.7)
engaged	ag-3 → ag-4 → pg-1 → pg-2 (1.73) ag-2 → ag-3 → ag-4 → pg-1 → pg-2 (1.71) ag-3 → ag-4 → pg-1 → pg-2 (1.61)

4.3 Q2. Applicability to Prediction Tasks

To show that our model is useful for downstream tasks, we use our model to predict (a) the type of each user's next action, (b) the time gap between the current and next actions of each user, and (c) whether each user reaches the target stage within 100 actions.

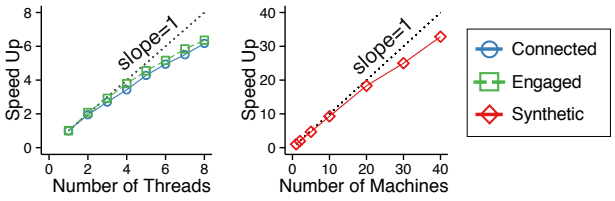
For each task, we compare the following approaches:

- Random: use a randomly chosen label.
- Frequent: use the label most frequent in the training set.
- LR: use logistic regression [22].
- Model: use the label most probable (according to our model) in the current stage of each user.
- Model + LR: divide users depending on their current stages (inferred by our model) and use logistic regression separately for each stage based on the actions in the stage.
- K-Means + LR: divide users using k-means++ [3] and use logistic regression separately for each cluster.

For logistic regression and k-means++, we used ($|\mathcal{A}| + |\Delta|$) features corresponding to the frequencies of the action types and time gaps for each user. For action-type and time-gap predictions, we used relative frequencies rather than absolute ones, which led to

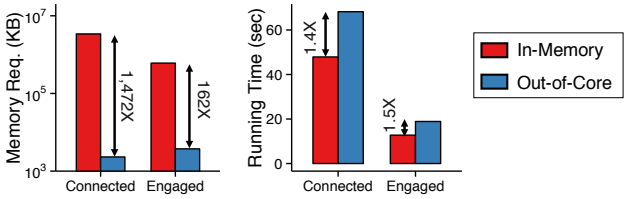
Table 4: Usefulness of our trained model for prediction tasks. Prediction solely based on our model (i.e., Model), which is unsupervised, showed similar accuracy to logistic regression (i.e., LR), which is supervised. Combining our model and logistic regression (i.e., Model + LR) was most accurate for all the tasks. See Section 4.3 for the detailed experimental settings.

Tasks	Action Prediction				Time-gap Prediction				Target-stage Reachability Prediction			
Datasets	connected		engaged		connected		engaged		connected		engaged	
Measure	Accuracy	F1	Accuracy	F1	Accuracy	F1	Accuracy	F1	Accuracy	F1	Accuracy	F1
Random	0.045	0.062	0.044	0.064	0.142	0.176	0.144	0.181	0.499	0.506	0.499	0.504
Frequent	0.273	0.117	0.327	0.161	0.348	0.180	0.154	0.041	0.615	0.469	0.598	0.447
LR	0.610	0.567	0.511	0.441	0.551	0.543	0.554	0.433	0.730	0.710	0.725	0.718
Model (Proposed)	0.597	0.508	0.482	0.443	0.577	0.537	0.564	0.494	N/A			
K-Means + LR	0.610	0.591	0.511	0.478	0.580	0.543	0.565	0.489	0.747	0.733	0.734	0.728
Model + LR (Proposed)	**0.684**	**0.675**	**0.548**	**0.523**	**0.646**	**0.633**	**0.586**	**0.536**	**0.756**	**0.751**	**0.734**	**0.731**



(a) Multi-Core Setting (b) Distributed Setting

Figure 3: Near-linear speed-ups of our optimization algorithm. Its speed increased near linearly with the number of threads in the multi-core setting and with the number of machines in the distributed setting.



(a) Main memory requirements (b) Running time

Figure 4: Memory efficiency of our out-of-core implementation in the real-world datasets. Out-of-core processing using external memory reduced the amount of required main memory space by up to $1,472\times$ at the expense of a slight decrease in speed.

higher accuracy. We trained our model and logistic regression using randomly chosen half of the users in each dataset and tested using the others. For evaluation, we used the proportion of correct predictions (Accuracy) and a weighted sum [6] of F1 scores (F1) [23].

As shown in Table 4, for all the tasks, prediction purely based on our model (Model), which is unsupervised, shows similar accuracy to logistic regression (LR), which is supervised. More importantly, their combination (Model + LR) was more accurate than combining k-means++ and logistic regression (K-Means + LR) as well as the individual methods (Model and LR) for all the tasks.

4.4 Q3. Scalability

We show the scalability of our optimization algorithm, described in Section 3. We consider the following implementations:

- In-Memory (Multi-Core): in-memory implementation where all data are loaded into memory. It can run in a parallel way in multi-core environments (see Section 3.3.2).
- Out-of-Core: out-of-core implementation using disk as the external memory (see Section 3.3.1).
- Distributed: MapReduce [10] implementation on Hadoop 2.6.1 (see Section 3.3.3).

First, we show that all implementations scale linearly with the size of the input data. To this end, we used synthetic datasets with different numbers of users while fixing $|\mathcal{A}| = |\Delta| = k = 10$ and $n_u = 1000$ for every user u. As seen in Figure 1 in Section 1, the per-iteration running times of all implementations increased linearly with the total number of actions. Especially, 'Distributed' processed a dataset with one trillion actions with per-iteration time less than 2 hours. We obtained similar results when we increased the number of actions per user, while fixing the number of users. These results are consistent with our theoretical analysis in Section 3.4.

Second, we show the near-linear speed-ups of our parallel and distributed implementations. To this end, we measured the speed-up [7] of 'Multi-Core' with different numbers of threads and that of 'Distributed' with different number of machines. For 'Multi-Core', we used the LinkedIn datasets with $k = 10$, and for 'Distributed', we used a larger synthetic dataset where $|\mathcal{A}| = |\Delta| = k = 10$, $|\mathcal{U}| = 10$ millions, and $n_u = 1000$ for every user u. As seen in Figure 3, both implementations showed near-linear speed-ups.

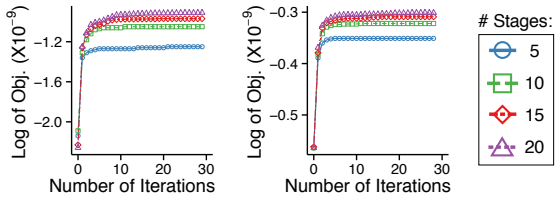
Lastly, we show significant reductions in main memory requirements by out-of-core processing using external memory. To this end, we compared the amount of main memory space required by 'In-Memory' and 'Out-of-Core' for processing the LinkedIn datasets with $k = 10$. As seen in Figure 4, 'Out-of-Core' required up to $1,472\times$ less main memory space than 'In-Memory'. However, the increase in the running time was at most 50%.

4.5 Q4. Convergence

We show that our optimization algorithm converges within a small number of iterations. Figure 5 shows the value of our objective function (i.e., Eq. (2)) in each iteration of our optimization algorithm in the LinkedIn datasets. The number of iterations required for convergence increased with the number of stages (i.e., k). However, even with 20 stages, our algorithm converged within 20 iterations.

[6] each weight is the proportion to the number of the corresponding label in the test set.

[7] relative speed compared to when a single thread (or a single machine) is used.

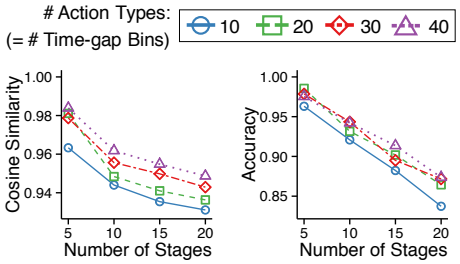


(a) Convergence (connected) (b) Convergence (engaged)

Figure 5: Fast convergence of our optimization algorithm. It converged within 20 iterations in the real-world datasets.

4.6 Q5. Identifiability

We show that our optimization algorithm learns parameters reasonably close to ground-truth parameters. We first created synthetic datasets by (a) choosing random probability distributions (i.e., $\{\theta_{s_i}\}_{s_i \in \mathcal{S}_{-k}}, \{\phi_{s_i}\}_{s_i \in \mathcal{S}_{-k}}, \{\psi_{s_i}\}_{s_i \in \mathcal{S}_{-k}}$) [8] and (b) generating action sequences of 100,000 users by following the generative process of our model (see Section 2.2). Then, we compared the probability distributions learned by our optimization algorithm with the ground-truth distributions in terms of cosine similarity [25]. We also compared the learned stage assignments with the ground-truth assignments in terms of accuracy (i.e., the proportion of correct assignments). Figure 6 shows the results averaged over 1,000 trials for each setting. As the number of stages (i.e., k) to be learned increased, similarity between trained and ground-truth parameters decreased. However, the increase in the number of action types (i.e., $|\mathcal{A}|$) and time gap bins (i.e., $|\Delta|$) increased the similarity by making different stages more likely to have distinct probability distributions. In every setting, the learned parameters were reasonably close to the true values. Specifically, the cosine similarity was higher than 0.93 and the accuracy was higher than 0.82 in every setting. These results were not sensitive to the values of the hyperparameters.



(a) Cosine similarity of probability distributions (b) Accuarcy of stage assignments

Figure 6: Accuracy of our optimization algorithm. The parameters trained by our algorithm were reasonably close to ground-truth parameters.

5 RELATED WORK

Modeling user behavior has been extensively studied to tackle various kinds of tasks. One line of work aims to predict events in the future by learning the hidden patterns from historical behavior. At a microscopic level, for predicting each event, many models and algorithms have been proposed, including frequency-based pattern mining [5, 15], mechanistic model approaches [7], generative models using latent cluster variables [13, 19], and LSTM-based approaches [16, 30]. In terms of modeling approaches, our work is closest to the generative latent variable models. However, our model is designed to infer more interpretable, coarse-grained patterns of event sequences. Such macroscopic progressions cannot be directly extracted through the use of microscopic event models.

On the other hand, another line of literature presents methods for capturing more macroscopic views of user behaviors through clustering event sequences. Those methods can typically visualize the overall view of user behavior from a certain aspect – such as navigation patterns on websites [9], diagram of user activity transitions [6], clusters of user types [27], and topics of event streams [12, 20]. Our work is related to this line of work in the sense that it provides high-level insights. However, our model presents multi-dimensional insights on each individual user's change, clusters of user actions, and transitions between these clusters.

Some recent work has focused on such multi-dimensional insights by grouping events and representing each entity's progression over coarse event groups. This approach is thus able to account for relationships between groups of events [24], regime shifts in event streams [18], and evolution of users [21].

In particular, modeling the progression of users over the latent stages has been proposed to distinguish different patterns of progression including development of various diseases [29]. Our work is closely related to this work in the sense that both assume latent stages that given event sequences progress through as well as each observed event in the sequences depends on the current latent stage. However, while the previous work differently models the step-by-step progression through each latent class, our work does not address different classes but model different patterns of progression using transition probabilities between stages. Furthermore, our model defines starting moments and goal stages to clearly illustrate the multiple paths in a particular progression region of interest, whereas the previous work does not contain a component triggering the start or the goal.

6 CONCLUSIONS

In this work, we propose a behavior model where progressions of users' behaviors from a starting state to a goal state are modeled as transitions over latent stages. These latent stages capture the progressions in three different aspects: types of actions, frequencies of actions, and directions of changes.

To fit our model to web-scale behavior logs, we propose a fast and memory-efficient optimization algorithm and extend it to multi-core, external-memory, and distributed settings. We theoretically and empirically demonstrate that our algorithm scales linearly with the size of the input data. Especially, the distributed version of our algorithm, implemented in the MapReduce framework, successfully handles a petabyte-scale dataset with one trillion actions.

We demonstrate the effectiveness of our model using a dataset from LinkedIn. Our model, however, can be applied to any dataset with a series of actions by different users over time. Our model discovers meaningful stages summarizing the progressions of LinkedIn users towards certain target states. We also show that stage information inferred by our model is useful for downstream tasks including prediction of next actions.

[8] we used $\lambda_\theta = \lambda_\phi = \lambda_\psi = 0.1$. We ignored probability distributions if there exists a short-lived stage (i.e., the probability of self-transition is less than 80%) or an isolated stage (i.e., the probability that a user reaches the stage is less than 10%).

REFERENCES

[1] 2017. Apache Hadoop. (2017). http://hadoop.apache.org/
[2] 2017. Statistical Machine Intelligence & Learning Engine. (2017). https://github.com/haifengl/smile
[3] David Arthur and Sergei Vassilvitskii. 2007. k-means++: The advantages of careful seeding. In *SODA*.
[4] Narayanaswamy Balakrishnan. 2006. *Continuous multivariate distributions*. Wiley Online Library.
[5] Iyad Batal, Dmitriy Fradkin, James Harrison, Fabian Moerchen, and Milos Hauskrecht. 2012. Mining recent temporal patterns for event detection in multivariate time series data. In *KDD*.
[6] Fabrício Benevenuto, Tiago Rodrigues, Meeyoung Cha, and VirgÃŋlio Almeida. 2009. Characterizing User Behavior in Online Social Networks. In *IMC*.
[7] Austin R. Benson, Ravi Kumar, and Andrew Tomkins. 2016. Modeling User Consumption Sequences. In *WWW*.
[8] Wray L Buntine. 1994. Operations for learning with graphical models. *Journal of Artificial Intelligence Research* (1994), 159–225.
[9] Igor V. Cadez, David Heckerman, Christopher Meek, Padhraic Smyth, and Steven White. 2000. Visualization of navigation patterns on a Web site using model-based clustering. In *KDD*.
[10] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.
[11] Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (methodological)* (1977), 1–38.
[12] Nan Du, Mehrdad Farajtabar, Amr Ahmed, Alexander J. Smola, and Le Song. 2015. Dirichlet-Hawkes Processes with Applications to Clustering Continuous-Time Document Streams. In *KDD*.
[13] Flavio Figueiredo, Bruno Ribeiro, Jussara M. Almeida, and Christos Faloutsos. 2016. TribeFlow: Mining & Predicting User Trajectories. In *WWW*.
[14] Ruining He and Julian McAuley. 2016. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In *WWW*.
[15] Srivatsan Laxman, Vikram Tankasali, and Ryen W. White. 2008. Stream prediction using a generative model based on frequent episodes in event sequences. In *KDD*.
[16] Liangyue Li, How Jing, Hanghang Tong, Jaewon Yang, Qi He, and Bee-Chung Chen. 2017. NEMO: Next Career Move Prediction with Contextual Embedding. In *WWW*.
[17] Xin Liu. 2015. Modeling Users' Dynamic Preference for Personalized Recommendation. In *IJCAI*.
[18] Yasuko Matsubara and Yasushi Sakurai. 2016. Regime Shifts in Streams: Real-time Forecasting of Co-evolving Time Sequences. In *KDD*.
[19] Yasuko Matsubara, Yasushi Sakurai, Christos Faloutsos, Tomoharu Iwata, and Masatoshi Yoshikawa. 2012. Fast Mining and Forecasting of Complex Time-Stamped Events. In *KDD*.
[20] Charalampos Mavroforakis, Isabel Valera, and Manuel Gomez-Rodriguez. 2017. Modeling the Dynamics of Learning Activity on the Web. In *WWW*.
[21] Julian John McAuley and Jure Leskovec. 2013. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *WWW*.
[22] Peter McCullagh. 1984. Generalized linear models. *European Journal of Operational Research* 16, 3 (1984), 285–292.
[23] David Martin Powers. 2011. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies* 2, 1 (2011), 37–63.
[24] Dafna Shahaf, Jaewon Yang, Caroline Suen, Jeff Jacobs, Heidi Wang, and Jure Leskovec. 2013. Information Cartography: Creating Zoomable, Large-Scale Maps of Information. In *KDD*.
[25] Amit Singhal. 2001. Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin* 24, 4 (2001), 35–43.
[26] Moshe Sniedovich. 2010. *Dynamic programming: foundations and principles*. CRC press.
[27] Gang Wang, Xinyi Zhang, Shiliang Tang, Haitao Zheng, and Ben Y. Zhao. 2016. Unsupervised Clickstream Clustering for User Behavior Analysis. In *CHI*.
[28] Robert West and Jure Leskovec. 2012. Human wayfinding in information networks. In *WWW*.
[29] Jaewon Yang, Julian McAuley, Jure Leskovec, Paea LePendu, and Nigam Shah. 2014. Finding progression stages in time-evolving event sequences. In *WWW*.
[30] Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. 2017. What to Do Next: Modeling User Behaviors by Time-LSTM. In *IJCAI*.