

TribeFlow: Mining & Predicting User Trajectories

Flavio Figueiredo^{1,2}, Bruno Ribeiro^{4,5}, Jussara Almeida³, Christos Faloutsos⁵

¹UFCEG - Brazil, ²IBM Research - Brazil, ³UFMG - Brazil, ⁴Purdue University, ⁵Carnegie Mellon University
{flaviiov,jussara}@dcc.ufmg.br, ribeiro@cs.purdue.edu, christos@cs.cmu.edu

ABSTRACT

Which song will Smith listen to next? Which restaurant will Alice go to tomorrow? Which product will John click next? These applications have in common the prediction of user trajectories that are in a constant state of flux over a hidden network (e.g. website links, geographic location). Moreover, what users are doing now may be unrelated to what they will be doing in an hour from now. Mindful of these challenges we propose TribeFlow, a method designed to cope with the complex challenges of learning personalized predictive models of non-stationary, transient, and time-heterogeneous user trajectories. TribeFlow is a general method that can perform next product recommendation, next song recommendation, next location prediction, and general arbitrary-length user trajectory prediction without domain-specific knowledge. TribeFlow is more accurate and up to $413\times$ faster than top competitors.

Keywords

User Trajectory Recommendation; Latent Environments;

1. INTRODUCTION

Web users are in a constant state of flux in their interactions with products, places, and services. User preferences and the environment that they navigate determine the sequence of items that users visit (links they click, songs they listen, businesses they visit). In this work we refer to the sequence of items visited by a user as the user's trajectory. Both the environment and user preferences affect such trajectories. The underlying navigation environment may change or vary over time: a website updates its design, a suburban user spends a weekend in the city. Similarly, user preferences may also vary or change over time: a user has different music preferences at work and at home, a user prefers ethnic food on weekdays but will hit all pizza places while in Chicago for the weekend.

The above facts result in user trajectories that over multiple time scales can be non-stationary (depend on wall clock times), transient (some visits are never repeated), and time-heterogeneous (user behavior changes over time); please refer to Section 5 for examples. Unfortunately, mining non-stationary, transient, and time-heterogeneous stochastic processes is a challenging task. It would

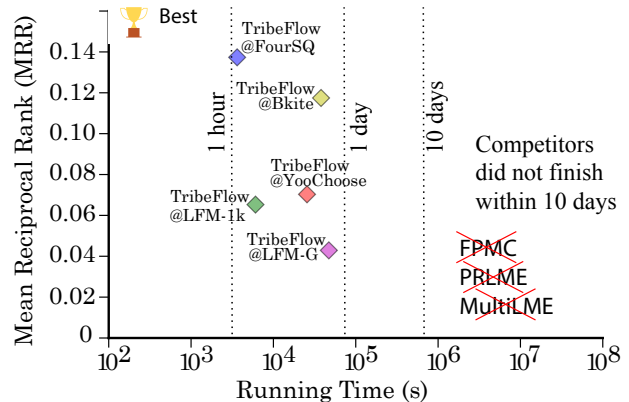


Figure 1: TribeFlow is at least an order of magnitude faster than state-of-the-art methods for next-item predictions.

be easier if trajectories were stationary (behavior is independent of wall clock times), ergodic (visits are infinitely repeated), and time-homogeneous (behavior does not change over time).

In this work we propose TribeFlow to tackle the problem of mining and predicting user trajectories. TribeFlow takes as input a set of users and a sequence items they visit (user trajectories), including the timestamps of these visits if available, and outputs a model for personalized next-item prediction (or next $n > 1$ items). TribeFlow can be readily applied to personalized trajectories from next check-in recommendations, to next song recommendations, to product recommendations. TribeFlow is highly parallel and nearly two orders of magnitude faster than the top state-of-the-art competitors. In order to be application-agnostic we ignore application-specific user and item features, including time-of-day effects, but these can be trivially incorporated into TribeFlow.

To illustrate the performance of TribeFlow consider Figure 1, where we seek to compare the Mean Reciprocal Rank (MRR) of TribeFlow over datasets with up to 1.6 million items and 86 million item visits (further details about this dataset is given in Section 4) against that of state-of-the-art methods such as Multi-core Latent Markov Embedding (MultiLME) [40], personalized ranking LME (PRLME) [13], and Context-aware Ranking with Factorizing Personalized Markov Chains [45] (FPMC). Unfortunately, MultiLME, PRLME, and FPMC cannot finish any of these tasks in less than 10 days while for TribeFlow it takes between one and thirteen hours. In significantly sub-sampled versions of the same datasets we find that TribeFlow is at least 23% more accurate than its competitors.

TribeFlow works by decomposing potentially non-stationary, transient, time-heterogeneous user trajectories into *very* short sequences of random walks on latent environments that are stationary, ergodic, and time-homogeneous. An intuitive way to understand TribeFlow

is as follows. Random walks have been widely used for ranking items on *observed graph* topologies (e.g. PageRank-inspired approaches [5, 24, 25, 28, 42, 46, 56]); meanwhile, overlapping community detection algorithms [1, 34, 44, 65] also use *observed graphs* to infer latent weighted subgraphs. But *what if we were not given the environments (weighted graphs & time scales) but could see the output of a random surfer over them?* TribeFlow sees user trajectories and infers a set of latent environments (weighted item-item graphs and their time scales) that best describe user trajectories through short random walks over these environments; after the short walk users perform a weighted jump between environments; the jump allows TribeFlow to infer user preference of latent environments. Once the TribeFlow model infers the relationships between short trajectories and latent environments, we can give it any user history and current trajectory to infer a posterior over the latent environment that the user is currently surfing and, this way, perform accurate personalized next-item prediction using a random walk. Our main contributions can be summarized as:

- **(Accuracy).** In our datasets TribeFlow predictions are always more accurate than state-of-the-art methods. The state-of-the-art methods include Latent Markov Embedding (LME) of Chen et al. [10], Multi-LME of Moore et al. [40], PRLME of Feng et al. [13], and FPMC of Rendle et al. [45]. TribeFlow is also more accurate than an application of the time-varying latent factorization method (TM-LDA) of Wang et al. [61]. We also see why TribeFlow can better capture the latent space of user trajectories than state-of-the-art tensor decomposition methods [39].
- **(Parameter-free).** In all our results TribeFlow is used without parameter tuning. Because TribeFlow is a nonparametric hierarchical Bayesian method, it has a few parameters that are set as small constants and do not seem significantly affect the performance if there is enough data. The only parameter that affects performance ($B \in \mathbb{Z}^+$ explained in the model description) is safely set to a small constant ($B = 1$).
- **(Scalability).** TribeFlow uses a scalable parallel algorithm to infer model parameters from the data. TribeFlow is between $48\times$ to $413\times$ faster than our top competitors, including LME, PLME, PRLME, and FPMC. When available, we evaluated the performance of TribeFlow over the datasets originally used to develop the competing methods.
- **(Novelty).** TribeFlow provides a general framework (*random surfer over infinite latent environments*) to build upon for application-specific recommendation systems.

Reproducibility. Of separate interest is the reproducibility of our work. Datasets, the TribeFlow source code and extra source code of competing methods that were not publicly available (implemented by us) can be found on our website¹.

We now present the outline of this work. Section 2 reviews the related work. Section 3 describes the TribeFlow model. Section 4 presents our results on both small and reasonably-sized datasets of real-world user trajectories. Section 4 also compares TribeFlow both in terms of accuracy and speed against state-of-the-art and naive methods. Section 5 shows that TribeFlow has excellent sense-making capabilities. Finally, Section 6 presents our conclusions.

2. RELATED WORK

How do we find a compact representation of the state space of network trajectories that allows easy-to-infer and accurate models?

¹<http://flaviovdf.github.io/tribeflow>

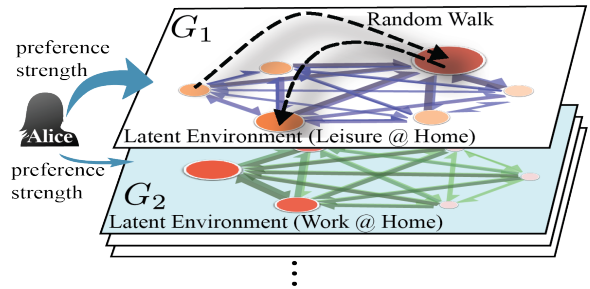


Figure 2: (TribeFlow) Alice randomly chooses a (latent) environment (weighted item-item graph G_1 and associated time scales) according to her preferences and surfs for a short time (two steps) before randomly jumping environments.

In this section, we present an overview of previous efforts related to TribeFlow that were also motivated by this question.

Random Walks over Observed Networks: The naive solution to the above question would be to simplify the state space of trajectories by merging nodes into communities via community detection algorithms [1, 44, 48, 49, 65]. However, we do not have the underlying network, only the user trajectories. TribeFlow is able to infer latent environments (weighted graphs and inter-event times) from user trajectories without knowledge of the underlying network.

Latent Markov Embedding: Latent Markov Embedding (LME) [10, 11, 13, 58, 63] was recently proposed to tractably learn trajectories through a Markov chain whose states are projected user trajectories into Euclidean space. However, even with parallel optimizations [11] the method does not yet scaled beyond hundreds of thousands of users and items (as shown in Section 4.3). The LME method can be seen as one of the first practical approaches in the literature to jointly learn user memory & item preferences in trajectories. Wu et al. [63] present a factorization embedding adding user personalization to the LME predictions, called PLME, which is not parallel and suffers from a quadratic runtime. Very recently Feng et al. [13] relaxed some of the assumptions of PLME in order to focus only on rankings (thus, PRLME), making the quadratic learning algorithm become linear but not multi-core as Multi-LME.

Factorizing Personalized Markov Chains: Rendle et al. [45] proposes Factorizing Personalized Markov Chains (FPMC) to learn the stochastic process of user purchasing trajectories. FPMC predicts which item a customer would add next in his or her basket by reducing the state space of trajectories of each user into an unordered set (i.e., trajectory sequence is neglected). The resulting Markov model of each user transition in the space of items forms the slice of a tri-mode tensor which then is embedded into a lower dimensional space via a tensor decomposition. Similarly, Aizenberg et al. [2] performs a matrix factorization for embedding. Wang et al. [60] and Feng et al. [13] also make personalized factorization-style projections. FPMC seems to be the most widely used method of this class that predicts next-items from unordered sets.

Collaborative Filtering Methods: Collaborative filtering methods can be broadly classified into two general approaches: memory-based (e.g. [52]) and item-based (e.g. [37, 50]). In memory-based models next item predictions are derived from the trajectory each user independently of other users. In item-based models, next item predictions are based on the unordered trajectories of all users, which disregards the sequence of events in the trajectory. More general unordered set predictions use Collective Matrix Factorization [54]. Recently, Hierarchical Poisson Factorization of Gopalan et al. [21] deal with a different problem: item recommendation based on user ratings rather than user trajectories. Chaney et al. [9]

Table 1: Comparison of Properties of State-of-art Methods.

	MC MLE [35]	Gravity Model [53]	LDA/TM-LDA [22, 61]	LME/MultiLME [10, 40]	P(R)LME [13, 63]	FPMC [45]	Temporal Tensors	TribeFlow (our method)
General Approach	✓		✓	✓	✓	✓	✓	✓
Trajectory Model	✓			✓	✓	✓		✓
Personalized			✓		✓	✓	✓	✓
Multiple Time Scales					✓	✓		✓
Trajectory Memory	✓							✓
Sense Making	✓	✓	✓				✓	✓
Sub-Quadratic	✓	✓	✓		✓	✓	✓	✓
Scalable	✓	✓	✓				✓	✓

extends the Gopalan et al. model for recommendations when network side information exists but also does not consider trajectories. The work of Wang et al. [61] uses a Latent Dirichlet Allocation-type embedding to capture latent topic transitions which we adapt to model trajectories in our evaluation (TM-LDA).

Naive Methods: Naive methods such as Gravity Model (GM) [53] are used to measure the average flow of users between items. Recently, Smith et al. [55] employs GMs to understand the flow of humans within a city. Galivanes et al. [18] employs GMs to understand Twitter user interactions. Note that GMs are application-dependent as they rely on a pre-defined distance function between items. Section 4.4 shows that TribeFlow is significantly more accurate than GM while retaining fast running times.

Other Markov Chain Approaches: A naive Markov Chain (MC) of the item-item transitions can be inferred via Maximum Likelihood Estimation (MLE) [35] but it does not provide enough flexibility to build good predictive models. Fox et al. [15, 16] and Matsubara et al. [38] propose Bayesian generalizations of Markov Switching Models [17] (MSMs) for a different problem: to segment video and audio. Liebman et al. [36] uses reinforcement learning (RL) to predict song playlists but the computational complexity of the method is prohibitive even for our smallest datasets.

Hidden Markov Models (HMMs) can also be used to model trajectories. However, even in recent approaches [20, 57, 59], fitting HMMs requires quadratic time in the number of latent spaces. The TribeFlow fitting algorithm is, in contrast, linear. HMMs are nevertheless interesting since inference is conditioned on the full sequence of a trajectory. TribeFlow can mimic this behavior with the B parameter. We also considered novel HMM based model (called Stages) that has been proposed by Yang et al. [64]. Stages has sub-quadratic runtimes in the number of visited items (transitions) but the author-supplied source code did not converge to usable parameter values in our larger datasets. It is unclear why Stages is unable to converge over large datasets. In smaller datasets, where convergence did occur, Stages is less accurate than TribeFlow. The lower accuracy is likely because Stages is more focused on explicit trajectory commonalities and does not model personalized user transitions explicitly as TribeFlow does.

Table 1 compares TribeFlow with the strongest competitors in the literature for trajectory prediction and sense-making. TribeFlow is the only method that meets all criteria: general, personalized, multiple time scales, and scalable. Our inference algorithm is sub-quadratic in asymptotic runtime, as well as fully parallel. TribeFlow is the only approach that is accurate, general, and scalable.

3. THE TRIBEFLOW MODEL

TribeFlow models each user as a random surfer over latent environments. User trajectories are the outcome of a combination of latent user preferences and the latent environment that users are exposed to in their navigation. We use a nonparametric model of short user trajectory sequences as steps of semi-Markov random

walks over latent environments composed of random graphs and associated inter-event time distributions. Inter-event times are defined as the time difference between two consecutive items visited by a user. The model is learned via Bayesian inference. Our model is illustrated in Figure 2; in our illustration user Alice jumps to a latent environment ($\mathcal{M} = 1$) according to her environment preferences and performs two random walk steps on the graph $G_{\mathcal{M}}$ with inter-event times associated to environment \mathcal{M} . After the two steps Alice randomly jumps to another latent environment of her preference.

Random walks on our latent environments are not to be confused with random walks on dynamic graphs. In the former the underlying graph topology and associated environment characteristics do not change once they are drawn from an unknown probability distribution while in the latter the graph structure is redrawn at each time step. In our applications probabilistic static environments seem like a better framework: a user with a given latent intent in mind (listen to heavy metal, eat spicy Indian food) at a given location (a webpage, a neighborhood) has item preferences (edge weights to songs, restaurants) similar to other like-minded users in the same network location. A random graph framework would be more accurate if webpages and restaurants randomly changed every time users made a choice. Our probabilistic environments are different from random graphs as the environment distribution is unknown and defines other characteristics such as the inter-event time distribution (simpler unidimensional examples of Markovian random walks on random environments with known distributions are given in Alexander et al. [3] and Hughes [27, Chapter 6]).

By construction, TribeFlow’s semi-Markov random walk generates ergodic, stationary, and time-homogeneous (E-S-Ho) trajectories. TribeFlow models potentially non-ergodic, transient, and time-heterogeneous user trajectories (Ne-T-He) as short sequences of E-S-Ho trajectories. By its nature E-S-Ho processes are generally easier to predict than Ne-T-He processes. And while a single user may not spend much time performing E-S-Ho transitions, other users will use the same E-S-Ho process which should allow us to infer well its characteristics.

3.1 Detailed TribeFlow Description

The set of users (\mathcal{U}) in TribeFlow can be agents such as people, bots, and cars. The set of items (Ω) can be anything: products, places, services, or websites. The latent environment $\mathcal{M} = 1, 2, \dots$ is a latent weighted clique $G_{\mathcal{M}} = (\Omega, E_{\mathcal{M}})$ over the set of items Ω . Edge weights in $E_{\mathcal{M}}$ have gamma distribution priors, $w_{(\cdot, v)} = w_v \sim \text{Gamma}(\beta, 1), \forall v \in \Omega$. In what follows we define the operator $|\cdot|$ to be the size of a set and \otimes denotes the outer-product.

Each user $u \in \mathcal{U}$ generates a “sequence trajectory” of length $B + 1$ at the t -th visit, $t \geq 1$,

$$(x_{u,t}, \dots, x_{u,t+B}) \in \Omega^{B+1}, B \geq 1,$$

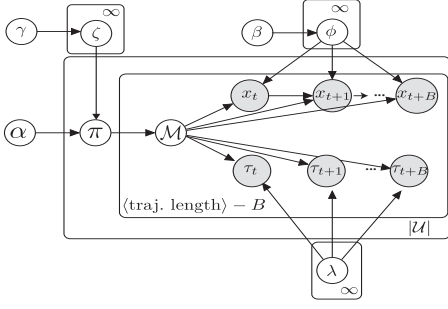


Figure 3: The TribeFlow model of the user and the semi-Markov transition probability matrix mixture.

before jumping to another environment \mathcal{M}' according to user preference distribution $\pi_{\mathcal{M}'|u}$. The entire trajectory $x_{u,1}, x_{u,2}, \dots$ of user $u \in U$ is the concatenation of such sequences.

The time between observations $x_{u,t+k}$ and $x_{u,t+k+1}$ is the k -th inter-event time $\tau_{u,t+k}$, $k = 0, \dots, B$. Special care must be taken with the last event of a user, which does not have an inter-event time. The random walk over $G_{\mathcal{M}}$ is modeled as a semi-Markov process with inter-event time $\tau_{u,t} \sim \lambda(\mathcal{M})$ (a.k.a. holding or residence times). Note again that inter-event times depend on the current latent environment.

We now define the transition probability matrix of our random walk over the random graph $G_{\mathcal{M}}$.

THEOREM 3.1. *The random walk over graph $G_{\mathcal{M}}$ of environment \mathcal{M} is a semi-Markov chain with a random $|\Omega| \times |\Omega|$ transition probability matrix distributed as*

$$\mathbf{P}_{\mathcal{M}} \sim (I - \text{diag}(\phi_{\mathcal{M}}))^{-1} (\mathbf{1} \otimes \phi_{\mathcal{M}} - \text{diag}(\phi_{\mathcal{M}})), \quad (1)$$

where \otimes denotes the outer product, $\mathbf{1}$ is a vector of ones, $\text{diag}(\cdot)$ is a diagonal matrix and $\phi_{\mathcal{M}} \sim \text{Dirichlet}(\cdot | \beta)$. Semi-Markov chain \mathcal{M} is stationary, ergodic, and time-homogeneous with high probability if $|\Omega| > 2$.

PROOF. Without loss of generality we assume that the walker starts at $o \in \Omega$. The semi-Markov random walk with transition probability matrix $\mathbf{P}_{\mathcal{M}}$ over $G_{\mathcal{M}}$, $\mathcal{M} \geq 1$, sees edge weights $w_v \sim \text{Gamma}(\beta, 1)$, $\forall v \in \Omega \setminus \{o\}$. The probability that the walk moves to $v \neq o$ is $w_v / S_{\neq o}$, where $S_{\neq o} = \sum_{j \in \Omega \setminus \{o\}} w_j$. Let $\mathbf{P}_{\mathcal{M}}(o, \Omega \setminus \{o\})$ denote the off-diagonal elements of the random walk transition probability matrix $\mathbf{P}_{\mathcal{M}}$. Because $\{w_j\}_{j \in \Omega}$ are independent and Gamma distributed then

$$\mathbf{P}_{\mathcal{M}}(o, \Omega \setminus \{o\}) = (w_v / S_{\neq o})_{v \in \Omega \setminus \{o\}}$$

follows a Dirichlet distribution $\text{Dirichlet}(\beta, \dots, \beta)$ [32, pp. 91]. Note that $\mathbf{P}_{\mathcal{M}}(o, o) = 0$. A little algebra gives Eq. (1). The chain is trivially stationary and time-homogeneous as transition probabilities do not change over time. We now show the chain is ergodic. Any state $j \in \Omega$ is reachable from any state $i \in \Omega$ as $(\mathbf{P}_{\mathcal{M}})^n(i, j) > 0$ for some $n \geq 1$ implying that the chain is recurrent [31, Theorem 5.1, pp. 66]. As $|\Omega| < \infty$ the chain is positive recurrent. By construction G_1 is connected and thus $\mathbf{P}_{\mathcal{M}}$ is irreducible. For $|\Omega| > 2$ the graph G_1 is not bipartite and making the chain aperiodic. If a chain is irreducible, aperiodic, and positive recurrent, then it is ergodic [31, pp. 85]. \square

Note that the random walk does not have **revisits**, $x_t \neq x_{t+1}, \forall t > 0$. In the datasets we remove all revisits because re-consumption (repeated accesses to the same item) tends to be easy to predict [14], highly application-specific, and can be *decoupled entirely* from our

problem via stochastic complementation using phase-type Markov chains with a single entry state such as the ones in Neuts [41], Robert and Le Boudec [47] and Kleinberg [33].

Gathering all elements together we obtain the model illustrated in Figure 3, which can be seen as a random surfer taking B steps over a latent graph $G_{\mathcal{M}}$ and then randomly moving to a new environment according to the following generative model:

1. Draw $\zeta \sim \text{GEM}(\gamma)$ according to a stick-breaking process.
2. For each user $u \in \mathcal{U}$ sample $\pi_{\mathcal{M}|u} \sim \text{Dirichlet}(\cdot | \alpha\zeta)$.
3. Draw a semi-Markov random walk transition probability matrix $\mathbf{P}_{\mathcal{M}} \sim (I - \text{diag}(\phi_{\mathcal{M}}))^{-1} (\phi_{\mathcal{M}} \otimes \phi_{\mathcal{M}} - \text{diag}(\phi_{\mathcal{M}}^2))$, where $\text{diag}(\cdot)$ is a diagonal matrix and $\phi_{\mathcal{M}} \sim \text{Dirichlet}(\cdot | \beta)$, $\mathcal{M} = 1, 2, \dots$
4. For a given user u each sequence burst $(x_t, \dots, x_{t+B})_u$ with inter-event times $(\tau_t, \dots, \tau_{t+B-1})$ is generated as follows:
 - (a) Draw a latent semi-Markov chain $\mathcal{M} \sim \text{Multinomial}(\pi_{\mathcal{M}|u})$.
 - (b) For $k = 0, \dots, (B-1)$ select item x_{t+k} according to probability $\mathbf{P}_{\mathcal{M}}(x_{t+k}, x_{t+k+1})$ and inter-event time $\tau_{t+k'} \sim \lambda(\mathcal{M})$, $k' = 0, \dots, B$, where $\lambda(\mathcal{M})$ is the inter-event time distribution of environment \mathcal{M} . Item x_1 is drawn uniformly from Ω .

3.2 Inferring TribeFlow Model from Data

In what follows we describe how we learn TribeFlow from data. Given a set of user trajectories $\{(x_{u,1}, x_{u,2}, \dots) : \forall u \in \mathcal{U}\}$ from a set of items $x_{u,t} \in \Omega$, $t \geq 1$, we infer:

- The number of environments $K > 1$ from the data.
- K semi-Markov transition probability matrices $\{\mathbf{P}_{\mathcal{M}} : \mathcal{M} = 1, \dots, K\}$ corresponding random walks over a *finite* set of graphs $\{G_{\mathcal{M}} : \mathcal{M} = 1, \dots, K\}$.
- A distribution of user environment preferences $\{\pi_{\mathcal{M}|u} : u \in \mathcal{U}\}$.

If the inter-event time distribution $\lambda(\mathcal{M})$ comes of a known family we can also get a distribution of inter-event times for each environment. The probability that a user sees a sequence $x_{u,t}, \dots, x_{u,t+B}$ with inter-arrival times $\tau_{u,t}, \dots, \tau_{u,t+B}$ at environment \mathcal{M} is

$$\begin{aligned} P[x_{u,t}, \dots, x_{u,t+B}, \tau_{u,t}, \dots, \tau_{u,t+B} | \mathcal{M}] = \\ \prod_{k=0}^{B-1} \mathbf{P}_{\mathcal{M}}(x_{u,t+k}, x_{u,t+k+1}) P[\tau_{u,t+k} | \mathcal{M}] P[\tau_{u,t+B} | \mathcal{M}], \end{aligned} \quad (2)$$

with $\mathbf{P}_{\mathcal{M}}$ as given in Eq. (1). The probability we observe such burst for user $u \in \mathcal{U}$ is then

$$\begin{aligned} P[x_{u,t}, \dots, x_{u,t+B}, \tau_{u,t}, \dots, \tau_{u,t+B} | u] = \\ \sum_{K=1}^{\infty} P[\zeta] P[\tau_{u,t+B} | \mathcal{M}] \pi_{\mathcal{M}|\alpha\zeta, u} \\ \times \prod_{k=0}^{B-1} \mathbf{P}_{\mathcal{M}}(x_{u,t+k}, x_{u,t+k+1}) P[\tau_{u,t+k} | \mathcal{M}], \end{aligned} \quad (2)$$

where $P[\zeta]$ is the stick-breaking prior over \mathcal{M} . Unrolling Eq. (1) into Eq. (2) we obtain the equation that describes the trajectory:

$$\begin{aligned} P[x_{u,t}, \dots, x_{u,t+B}, \tau_{u,t}, \dots, \tau_{u,t+B-1} | u] \propto \\ \sum_{K=1}^{\infty} P[\zeta] P[\tau_{u,t+B} | \mathcal{M}] \pi_{\mathcal{M}|\alpha\zeta, u} \\ \times \prod_{k=0}^{B-1} \frac{\phi_{\mathcal{M}}(x_{u,t+k+1})}{1 - \phi_{\mathcal{M}}(x_{u,t+k})} P[\tau_{u,t+k} | \mathcal{M}]. \end{aligned} \quad (3)$$

We use collapsed Gibbs sampling to estimate the model parameters. Initially, given a sequence of size $B + 1$, we transform user trajectories into a set \mathcal{D} of tuples using a sliding window over the trajectories of each user. To exemplify, for $B = 2$ each entry is:

$$(u, x_{u,t}, x_{u,t+1}, x_{u,t+2}, \tau_{u,t}, \tau_{u,t+1}) \in \mathcal{D}, t \geq 1.$$

This tuple represents the user u , the trajectory $x_{u,t}, \dots, x_{u,t+B}$ and the inter-event times $\tau_{u,t}, \dots, \tau_{u,t+B}$ for every time $t \geq 1$. The use of sliding windows is inspired by Wang et al.’s [59] collapsed Gibbs sampling (equation in Figure 1 of Wang et al.) for approximate inference in Hidden Markov Models (HMMs). While our procedure is not similarly theoretically justified by our model, it tremendously simplify the inference problem by not forcing us to decide how to segment the data into blocks of B events or forcing us to make B a random variable. Adding random jumps would entail a costly forward-backward inference procedure needed to decide when users jump between environments. To infer TribeFlow, our heuristic starts with an initial estimate of the number of environments K and randomly assign each tuple in \mathcal{D} to one environment.

After the initial assignment we count the number of tuples of each user u : $n_u = \sum_{(u', \dots) \in \mathcal{D}} \mathbf{1}(u' = u)$, where $\mathbf{1}$ is the indicator function. We also count the number of times environment \mathcal{M} is assigned to a tuple from user u : $e_{\mathcal{M},u}$, as well as the joint count of items, at any position, and environments: $c_{i,\mathcal{M}}$, and count the number of tuples assigned to an environment \mathcal{M} : $a_{\mathcal{M}}$. Assuming, for now, that ζ is given, we can infer [22]

$$\pi_{\mathcal{M}|\alpha\zeta,u} = \frac{e_{\mathcal{M},u} + \alpha\zeta(\mathcal{M})}{n_u + K\alpha\zeta(\mathcal{M})}, \quad \phi_{\mathcal{M}}(i) = \frac{c_{i,\mathcal{M}} + \beta}{a_{\mathcal{M}} + |\Omega|\beta}. \quad (4)$$

We then employ ECME [19] inference where: (1) the e-step consists of one pass over the entire dataset performing a Gibbs sampling update (in other terms, one iteration of the collapsed Gibbs sampler); (2) an m-step where the algorithm considers the probability of inter-event times according to the following procedure.

If the inter-event times of environments $\mathcal{M} = 1, 2, \dots$ have a known probability law $\lambda(\mathcal{M})$ we can include this law in our model with the appropriate priors (e.g. a Normal distribution with fixed variance can have a Normal prior), updating the distribution parameters in each m-step. If the law of $\lambda(\mathcal{M})$ is unknown but inter-event times are observed, we use the empirical complementary cumulative distribution function (ECCDF) in the the following heuristic: We estimate the ECCDF of each \mathcal{M} based on the entries assigned to \mathcal{M} on the last e-step. If $T_{\mathcal{M}}$ is the random variable that defines the inter-arrival at environment $\mathcal{M} = 1, \dots$. For inter-event time $\tau_{u,t}$ the probability $P[T_{\mathcal{M}} > \tau_{u,t}]$ given the current ECCDF is the number of entries whose observed inter-event times in \mathcal{M} are greater than $\tau_{u,t}$. Thus, $\mathbf{1}(T_{\mathcal{M}} > \tau_{u,t})$ is a Bernoulli random variable [62] with parameter $p = P[T_{\mathcal{M}} > \tau_{u,t}]$. Adding a conjugate prior $Beta(1, K - 1)$ to the Bernoulli gives the following predictive posterior [19]:

$$P[\tau_{u,t}|\mathcal{M}] \propto \frac{b_{>\tau_{u,t},\mathcal{M}} + 1}{n_{\mathcal{M}} + K}, \quad (5)$$

where $b_{>\tau_{u,t},\mathcal{M}}$ is the number of tuples currently assigned to \mathcal{M} that have inter-arrival times greater than $\tau_{u,t}$. It is easy to see from Eq. (5) that transitions with large inter-event times w.r.t. other inter-event times in \mathcal{M} are less likely to have been generated by \mathcal{M} . This captures the intuition that inter-event times in an environment should be similar and is an integral part of how we learn ζ as detailed below. In our experiments the ECCDF heuristic consistently provides better results than assuming Normal inter-event time distributions. Testing other application-specific inter-event time posteriors is of interest in more application-specialized future work.

We now infer K , the number of environments, by adapting the partial expectation (partial-e) and partial maximization (partial-m) heuristics of Bryant and Sudderth [7] for hierarchical Dirichlet processes (HDPs) as follows: (a) merge pairs of redundant environments when there is gain in the joint posterior probability of model & data; and (b) split environments based on their inter-event times by separating the 5% entries with highest inter-event time if there is a marginal gain in the posterior. The merge step (partial-e) is equivalent to the one described in [7]. Our split step (partial-m) splits a environment with high variance inter-event times into two environments with lower variance if the splitting improves the joint posterior probability of model & data.

Parallel Learning: Finally, the model can be fully trained in parallel using the approach described in Asuncion et al. [4]. Each processor learns the model on a subset of the dataset. After a processor performs an ‘‘E’’ and an ‘‘M’’ steps, another processor is chosen at random to synchronize the model state (merge count matrices of the Gibbs sampler and update ECCDF estimates), leaving them with the same count matrices and ECCDF estimates. After a fixed number of iterations, which we fix as 200, every processor meets a barrier and the partial-e and partial-m steps (merges and splits) are performed on a single master processor. After these partial-e and partial-m steps, the master-processor updates all slave processors and the parallel learning continues as previously described. The learning ends after a fixed number of iterations, which we set as 2,000 for the results in Section 4.

Inference without Timestamps: In a few of our datasets timestamps are not available. In these cases we only need to infer $\mathbf{P}_{\mathcal{M}}$ and $\pi_{\mathcal{M}|\alpha\zeta,u}$ without the need to take into account $P[\tau_{t+k}|\mathcal{M}]$. In our results we also take the opportunity to assume that the number of environments is fixed ($\zeta(\mathcal{M}) = 1$) and infer the model posteriors using collapsed Gibbs sampling [22, 23, 29, 30, 38, 61, 66] employing Eqs. (3) and (4), updating the counts at each iteration (updating the posterior probabilities). We denote the latter simpler approach **TribeFlow-NT** (TribeFlow-NoTimestamps).

Prior Parameters: For all of our experiments, we fix prior parameters $\alpha = 50/K$ and $\beta = 0.001$. Constant γ does not need to be stated explicitly. In the presence of large amounts of data these priors are expected to have little impact on the outcome of Bayesian nonparametric models [19].

3.3 TribeFlow Predictions

In this work TribeFlow has two prediction tasks: **next-item** likelihood predictions and **ranking**. The personalized predictive likelihood of user $u \in \mathcal{U}$ for candidate next-item $\tilde{x}_{u,t+1} \in \Omega$ is based on the last B items and inter-event times of the user. Observing that user u has chosen items $x_{u,t-1}, \dots, x_{u,t-B}$ with inter-event times $\tau_{u,t-1}, \dots, \tau_{u,t-B}$ gives a posterior probability that u is performing a random walk at latent environment $\mathcal{M} \in \{1, \dots, K\}$, where K is the learned number of latent environments, typically $K < 10^3$ in our experiments. More precisely, the posterior probability that $u \in \mathcal{U}$ is in environment \mathcal{M} after choosing items $x_{u,t-1}, \dots, x_{u,t-B}$ with inter-event times $\tau_{u,t-1}, \dots, \tau_{u,t-B}$ is

$$P[\mathcal{M}|u, x_{u,t-1}, \dots, x_{u,t-B}, \tau_{u,t-1}, \dots, \tau_{u,t-B}],$$

which yields the full likelihood

$$P[\tilde{x}_{u,t+1}|u, x_{u,t-1}, \dots, x_{u,t-B}, \tau_{u,t-1}, \dots, \tau_{u,t-B}] = \quad (6)$$

$$\left(\prod_{k=1}^B \mathbf{P}_{\mathcal{M}}(x_{u,t-k}, x_{u,t-k+1}) P[\tau_{u,t-k}|\mathcal{M}] \pi_{\mathcal{M}|\alpha\zeta,u} \right)$$

$$\times \frac{\sum_{\mathcal{M}=1}^K \mathbf{P}_{\mathcal{M}}(x_{u,t}, \tilde{x}_{u,t+1})}{\sum_{\mathcal{M}=1}^K \prod_{h=1}^B \mathbf{P}_{\mathcal{M}}(x_{u,t-h}, x_{u,t-h+1}) P[\tau_{u,t-h}|\mathcal{M}] \pi_{\mathcal{M}|\alpha\zeta,u}}.$$

Table 2: Summary of our datasets.

	Users	Items	# Transitions	# Users	# Items	Inter-event Times	Timestamp Span
Last.FM-1k	User	Artist	10,132,959	992	348,156	Yes	Feb. 2005 to May 2009
Last.FM-Groups	User	Artist	86,798,741	15,235	1,672,735	Yes	Feb. 2005 to Aug. 2014
BrightKite	User	Venue	2,034,085	37,357	1,514,460	Yes	Apr. 2008 to Oct. 2010
FourSQ	User	Venue	453,429	191,061	87,345	Yes	Dec. 2012 to April 2014
YooChoose	Session	Product	19,721,515	6,756,575	96,094	Yes	Apr. 2014 to Sept. 2014
Yes	Playlist	Song	1,542,372	11,139	75,262	No	(No timestamps)

The task of ranking the next items is easier and faster than predicting their likelihood. This is because we can speed up the predictions by not computing the denominator in eq. (6), as the denominator is the same for all values of $\tilde{x}_{u,t+1} \in \Omega$. Note that our rankings are personalized and consider the inter-event times.

4. RESULTS

Previous sections introduce TribeFlow and explain how to infer its posteriors from data. We now turn our attention to compare TribeFlow against state-of-the-art approaches, solving the very same problems over some of the very same datasets (if publicly available) as their original papers. We also use larger publicly available dataset and one ever larger dataset that we collected for this study (available for download at our website). In what follows Section 4.2 contrasts TribeFlow against state-of-the-art methods for next-item ranking. Section 4.3 compares TribeFlow against methods that learn latent Markov chains and predict the likelihood of next items. Finally, Section 4.4 contrasts TribeFlow ability to predict average mean-field user flows against that of Gravity Model.

Although TribeFlow is able to predict not just the next-item but also the next $n \geq 1$ items, our evaluations are based at next-item predictions because previous efforts mostly focused their evaluations on this task. We also consider the reconsumption problem (consecutive visits to the same item) treated in Figueiredo et al. [14] as a separate, often easier, problem that can be dealt with via stochastic complementation as discussed in Section 3.1.

4.1 Datasets and Evaluation Setup

Our datasets encompass three broad range of applications: (a) location-based social networks (check-in datasets), (b) music streaming applications, and (c) user clicks on e-commerce websites. It is important to point out that recommendation engines and user interfaces influence user navigation and their trajectories. Such effects are considered to be an integral part of our predictive task. But TribeFlow can as easily learn pure user preferences if given a dataset with no environment bias (when that is possible).

Table 2 summarizes our datasets showing the number of users and items, the total number of $x_{u,t}, x_{u,t+1}$ pairs visited all users (or transitions/trajectories when $B = 1$) as well as the time span covered by the dataset. The set of items, Ω , can be songs or artists on music datasets, venues on check-in data, and products on e-commerce data. The set of users \mathcal{U} are individuals, a “playlist”, or a “browser session” as described next.

Last.FM-Groups. Last.FM is a music streaming service that aggregates data from various forms of digital music consumption, ranging from desktop/mobile media players to other streaming services. This dataset was crawled in August 2014, using the user groups² feature from Last.FM. We manually selected 15 groups of

pop artists and two general interest groups³. For each group, we crawled the listening history of a subset of the users (the first users listed in the group).

Last.FM-1k. The second Last.FM dataset was collected in 2009 using snowball sampling by Celma et al. [8].

BrightKite. Brightkite is a location based social network (LBSN) where users share their current locations by check-ins. In this publicly available dataset, each items is a location where users are checks-in. Collected by Cho et al. [12].

FourSQ Our second LBSN dataset was gathered from FourSquare by Sarwt et al. [51] in 2014.

YooChoose. This dataset is comprised of user clicks on a large e-commerce business. Each users is captured by a session and the trajectories capture clicks on different products within the session. As “users” are actually browser sessions, there is an upper-limit of 12 hours on recorded inter-event times of a single “user” (session).

Yes. Finally, the Yes dataset consists of song transitions (playlists) of popular broadcast (offline) radios in the United States. *This dataset does not provide explicit user information or timestamps.* However, we use TribeFlow-NT by defining the playlists as users and each song as an item. The Yes dataset was collected by Chen et al. [10, 11] to develop LME.

No filtering or trimming is done over the original data for the results shown in Figure 1. In our evaluation of trajectory predictions we divide the datasets into “past” ($\mathcal{D}_{\text{past}}$) and “future” ($\mathcal{D}_{\text{future}}$) by selecting a timestamp that splits the dataset into “the first 70% transitions” for “past” (training set) and the remaining 30% transitions in the data in “future” (test set), with the exception of Yes that has no timestamps. This training and testing scenarios best represent real-life situations where the training is performed in batches over existing data. For instance, in this realistic training and test setting we may need to predict the next transitions of a user that belongs to the test set but not to the training set. Note that some users will have trajectories confined in the “past” dataset while the trajectory of “new users” may be entirely placed in the “future” dataset. For the Yes data the training and testing sets are the ones pre-defined by Chen et al. [10, 11]. Due to limitations in scalability of state-of-the-art methods we also exploit subsamples of our larger datasets when necessary. These subsamples have the first 1000, 2000, 5000, 10,000, 20,000 and 100,000 transitions ordered by timestamps. We test TribeFlow’s robustness through these dataset subsamples.

Setup: Our tests run on a server with 2×10 -core Intel Xeon-E5 processors and 256 GB of RAM. We tested and present results of TribeFlow with $B = 1, 2, 3, 4, 5$, fixing other hyper-parameters

³Active Users, Music Statistics, Britney Spears, The Strokes, Arctic Monkeys, Miley Cyrus, LMFAO, Katy Perry, Jay-Z, Kanye West, Lana Del Rey, Snoop Dogg, Madonna, Rihanna, Taylor Swift, Adelle, and The Beatles

²Pages in which the user discusses musical artists

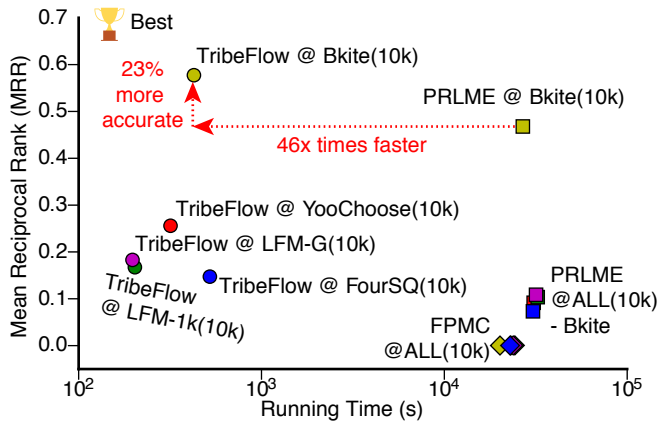


Figure 4: TribeFlow again outperforms in ranking task (subsampled datasets with only 10^4 transitions).

as discussed in Section 3. Our results show that regardless of the choice of B , TribeFlow outperforms competitors.

4.2 TribeFlow for Next-item Ranking

Before we compare TribeFlow against competing approaches it is important to note **only TribeFlow can handle our larger datasets** as evidenced by Figure 1. All of our comparison against competing state-of-the-art methods are **performed over small or sub sampled datasets due to the large execution times**.

We start our discussion on ranking methods by focusing on the evaluation metric: the mean reciprocal rank. The reciprocal rank, RR , is the inverse of the position of the destination $x_{u,t+1}$ on the ranking of *all* potential candidates in decreasing order. That is, if candidate, $\tilde{x}_{u,t+1}$, destinations *Big Brewery*, *Pizza Place*, and *Sandwich Shop* are ranked with probabilities 0.4, 0.5, and 0.1 respectively, and the *true* destination was *Big Brewery*, the reciprocal rank has value $1/2$. Using TribeFlow, the RR can be computed using Eq. (6) as follows (with $B = 1$):

$$RR(x_{u,t+1}, x_{u,t}, \tau_t, u) = \frac{1}{\text{rank}(P[x_{u,t+1} | x_{u,t}, \tau_t, u])}. \quad (7)$$

Based on the reciprocal rank we define a single metric measured over the entire test set. This metric is simply the mean of reciprocal rank values over every transition in \mathcal{D}_{test} (MRR).

TribeFlow with HDP heuristics: Before comparing TribeFlow with competing approaches, we test if our heuristic of HDP expansion and contraction of latent environments & inter-event time ECCDF inference improves the quality of the predictions. That is, we run TribeFlow with and without the partial e- and m-steps described in Section 3. We train TribeFlow with an initial guess of $K = 100$ environments and TribeFlow-NT (without partial e- and m-steps) with 100 environments. We evaluate the MRR at our largest datasets: BrightKite, LastFM-Groups, LastFM-1k, and YooChoose, and one small dataset (FourSQ). At FourSQ and YooChoose both TribeFlow and TribeFlow-NT have the same accuracy, possibly because of lower quality timestamps: FourSQ user data is sub-sampled *in time* and YooChoose browser sessions timeout at 12 hours. For the remaining datasets the MRR gains of TribeFlow over TribeFlow-NT are 7%, 45%, and 53%, for LastFM-1k, LastFM-Groups, and BrightKite, respectively. This shows that our partial e- and m-step heuristics can significantly improve the results.

TribeFlow v.s. state-of-the-art: We now turn to our comparison of TribeFlow against state-of-the-art competing ranking approaches. Our first competitor is Factorizing Personalized Markov Chain (FPMC) of Rendle et al. [45]. FPMC was initially proposed

to predict the next object a user will insert into an online shopping basket. If we consider each source x_t as a size one shopping basket, FPMC can be used to rank the candidate items, or destinations \tilde{x}_{t+1} , that a user will consume next. Our second competitor is the best-performing Latent Markov Embedding (LME) method in our datasets: Personalized Ranking by Latent Markov Embedding (PRLME) of Feng et al. [13]. Inspired by Personalized Latent Markov Embedding [63] (PLME), the PRLME approach focuses on rankings and not on extracting Markov chains.

Figure 4 presents the results of TribeFlow against FPMC and PRLME over datasets Bkite, FourSQ, LFM-Groups, LFM-1k, YooChoose subsampled to the first **10,000** transitions only due to scalability issues of FPMC and PRLME (even when limiting inference with 1000 stochastic gradient descent iterations). The figure shows the MRR scores (y-axis) against running times in log-scale (x-axis). Each point in the figure represents one method-dataset pair, while different datasets are represented by different colors. We label each point in order to help readability. The top-left corner of the figure indicates the best accuracy and shorter runtime. In these tests, as with all our tests, we use 70% of the initial user transitions to perform inference and the last 30% to test accuracy.

In Figure 4 we see that *TribeFlow is 23% more accurate and 46x times faster than the best result of PRLME*. In all cases TribeFlow is more accurate and faster than PRLME and FPMC, oftentimes TribeFlow is two orders of magnitude faster. This is surprising as TribeFlow runs parameter-free while PRLME and FPMC both optimize over a large parameter space to obtain their best results⁴. Our evaluation also considers a range of subsampled transitions in the datasets, from 10^3 to 10^5 transitions. The accuracy results are similar in all cases, with TribeFlow showing consistently more accurate results. Interestingly, using the results from the LastFM-Groups subsamples (1000, 5000, 10000, 20000, 10^5) using a simple linear regression reveals that it would take *over six years* to run PRLME and FPMC in the full LastFM-Groups dataset using our server. The minimum expected running time of PRLME/FPMC on a complete dataset is 14 days for the FourSQ dataset (the smallest dataset). The parameter search, as well as the lack of parallelism, greatly impact the runtime of PRLME and FPMC. Nevertheless, both methods are sub-quadratic as is TribeFlow. In the next sub-section we shall compare TribeFlow with another fully parallelized baseline.

Finally, we point that we also compare TribeFlow with the Stages method proposed by Yang et al. [64]. Our simulations used an author-supplied source code⁵ that, unfortunately, did not converge to usable parameter values except over a few of the sub-sampled datasets. In the cases where the model converged, best results of Stages over TribeFlow were observed in the YooChoose data (sub-sampled to 10k transitions) where Stage’s MRR value is 0.15 while TribeFlow’s MRR is 0.25. That is, in its best-performing dataset Stages is 40% less accurate than TribeFlow. Moreover, Stages achieved MRR values of 0.05 for Brightkite (against 0.57 on TribeFlow) and of 0.10 on FourSQ (against 0.14 for TribeFlow).

The results discussed so far consider only average measures of the effectiveness of the rankings produced by the methods. Going a step further, we also performed a Kolmogorov-Smirnov test between the distributions of reciprocal rank values obtained by TribeFlow and the competing methods. The results again clearly indicate that TribeFlow obtains larger RR values over all datasets ($p < 0.001$), which is consistent the MRR results. Another aspect is the impact of the walker sequence length B on TribeFlow performance. We test TribeFlow with $B = 1, 2, 3, 4, 5$ finding that larger B improves

⁴We perform a grid-search over parameters, testing 3 to 5 different values for each parameter of each method.

⁵<http://infolab.stanford.edu/~crucis>

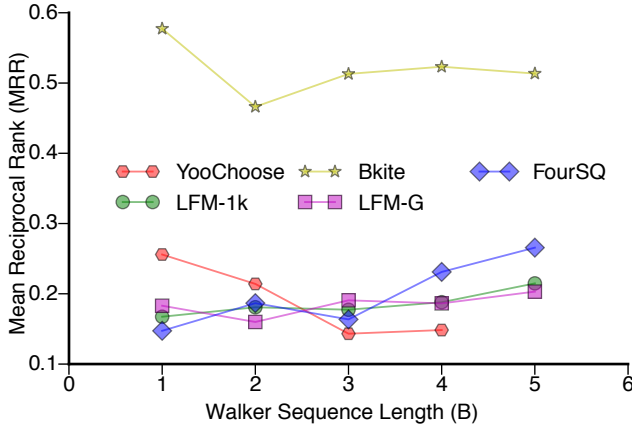


Figure 5: Impact of walker sequence length B .

accuracy at FourSQ, LFM-1k, LFM-Groups and reduces accuracy at Bkite and YooChoose as shown in Figure 5. As the best choice of B is application dependent, we recommend fixing $B = 1$.

4.3 TribeFlow’s Execution Time

In this section we compare the performance of TribeFlow against Multi-LME [10, 11] in our smallest datasets, namely Yes and FourSQ. **These experiments validate the performance (in training time) of TribeFlow against another fully parallelized method.** For the sake of fairness, we also compare TribeFlow with MultiLME using the next-item likelihood measure: $P[x_{u,t+1} | x_{u,t}]$ as in Chen et al. [11] and other naive approaches (see below). The predictive likelihood using TribeFlow can be computed as:

$$P[x_{u,t+1} | x_{u,t}] = \sum_{\mathcal{M}} \mathbf{P}_{\mathcal{M}}(x_{u,t+1}) \mathbf{P}_{\mathcal{M}}(x_{u,t}) P[\mathcal{M}] \quad (8)$$

where $P[\mathcal{M}] \propto$ the number of tuples assigned to \mathcal{M} in the inference. Accuracy on the test set is measure using the log likelihood:

$$\text{PredLL}(\mathcal{D}_{\text{future}}) = \sum_{x_{u,t+1}, x_{u,t} \in \mathcal{D}_{\text{future}}} \log(P[x_{u,t+1} | x_{u,t}]). \quad (9)$$

Naive Approaches: To compute $P[x_{u,t+1} | x_{u,t}]$ one can also trivially adapt both Latent Dirichlet Allocation (LDA) [6] and Transition Matrix LDA (TM-LDA) [61] for the same task. In LDA each we define users as “documents”, environments are “topics”, and $x_{u,t}$ is the t -th word. TM-LDA follows the same definitions. LDA is trained with K latent factors and TM-LDA creates a (K, K) matrix capturing the probability of transitioning between a latent “topics”. Note that in these models $P[x_{u,t+1} | x_{u,t}] = P[x_{u,t+1}]$. We refer to these adaptations as LDA’ and TM-LDA’ for simplicity as they do not reflect the original applications of LDA and TM-LDA.

Inference Procedure: In our evaluations we use the fully parallelized Multi-LME implementation [11]⁶. The LDA’ and TM-LDA’ adaptations to our problem are trained using `scikit-learn` [43] which includes a fast/online [26] and parallelized implementation of these methods. While faster LDA’ training methods do exist [67], we preferred to make use of a mature software package.

The Yes dataset of Chen et al. [10, 11] has no timestamps, thus we use TribeFlow-NT for this comparison instead of the more accurate general TribeFlow method. Each method was trained using different values of K , the number of latent environments ($K \in \{10, 50, 25, 100\}$) on the Yes dataset and with $K = 10$ on FourSQ because the original Multi-LME code has difficulties scaling to more factors on FourSQ.

⁶<http://www.cs.cornell.edu/People/tj/playlists/>

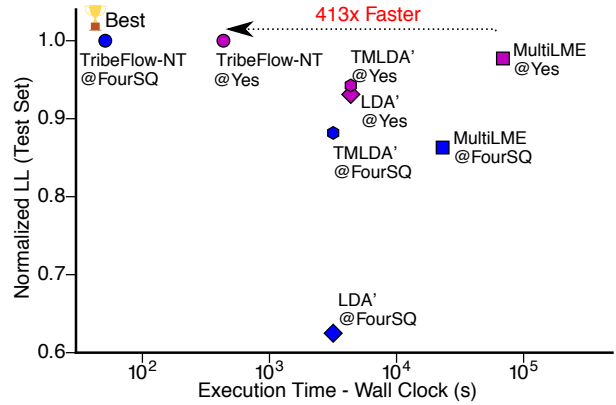


Figure 6: TribeFlow outperforms competition in next-item predictive log-likelihood task (small dataset due to competitors).

Results: Figure 6 presents our results in the next-item predictive log-likelihood task using $K = 100$ and $K = 10$ on the Yes and FourSQ datasets. We get similar results for Yes with $K \in \{10, 25, 50\}$. For the sake of interpretability we normalize the predictive log likelihood of each method by the best result (always TribeFlow). Each point in the figure represents one method-dataset pair, while different datasets are represented by different colors. We labelled each point in order to help the interpretation of the results. With these settings, it is expected that the method that performs the best embedding is placed in the top-left corner of the figure (higher accuracy, lower runtime). The x-axis represents the execution time, whereas the y-axis represents the normalized likelihood.

In Figure 6 we clearly see that TribeFlow-NT is the best approach on both datasets. Compared to MultiLME TribeFlow-NT achieves a higher log-likelihood at a fraction of the runtime (speedups are up to 413 \times). As expected, TribeFlow-NT and MultiLME usually outperforms the LDA-based baselines since TribeFlow-NT and MultiLME were built to explicitly capture user trajectories. Interestingly, comparing the speed of TribeFlow and TribeFlow-NT in Figures 1 and 6 we see that for FourSQ TribeFlow-NT is one order of magnitude faster than TribeFlow but less accurate than TribeFlow. This occurs because TribeFlow-NT does not infer inter-event time distributions using the ECCDF heuristic.

4.4 Flows of Users Between Locations

TribeFlow outperforms the state-of-the-art methods in sophisticated tasks such as ranking and predictive next-item likelihoods. But what about a simpler task? Can TribeFlow outperform simple application-specific methods? In this section we compare TribeFlow against the Gravity Model (GM) for uncovering the average flows of users between two locations. The widely popular Gravity Model (GM) uses GPS coordinates and requires a pre-defined distance function, $dist$, capturing the proximity of locations around the globe. As in Smith et al. [55] and Garcia-Gavilanes et al. [18] we employ the distance on a sphere from the latitude and longitude coordinates of the venues in our LBSN datasets. The three parameters, θ_1 , θ_2 and θ_3 of the distance are fitted using a Poisson regression that is known to lead to better results [53].

Our goal is to estimate f_{ds} , the flow of users going from location $s \in \Omega$ to location $d \in \Omega$. Let $n_d = \sum_{x_{u,t+1}, x_{u,t} \in \mathcal{D}_{\text{present}}} \mathbf{1}(x_{u,t+1} = d)$ denote the number of visits of all users to a destination d and r_s be the equivalent number of visits of all users to a source location s . GM captures the flows of users between the two locations f_{ds} as

$$\hat{f}_{ds}^{(\text{GM})} = \frac{r_s^{\theta_1} n_d^{\theta_2}}{dist(d, s)^{\theta_3}}.$$



Figure 7: Examples of popular items in latent TribeFlow environments in a dataset dominated by pop music fans (LFM-G).



Figure 8: Latent flow environments inferred by TribeFlow without GPS information.

TribeFlow can trivially estimate the flow $\hat{f}_{ds}^{(TF)}$ from Eq. (8). Since gravity models are limited to geolocated datasets (need GPS coordinates), we compare TribeFlow with GM on our FourSQ and Brightkite. Note that unlike GM, TribeFlow is application-agnostic and **does not** use GPS coordinates, albeit it would be straightforward to incorporate such application-specific features in the latent environments. Further, we opt to use TribeFlow-NT (instead of the more accurate TribeFlow method) because of its slightly faster running time, trading-off accuracy for speed. We infer the posteriors from TribeFlow-NT with $K = 10$ environments. In this setting, training both models takes less than 5 minutes. Methods are evaluated using the mean absolute error (MAE).

TribeFlow-NT significantly outperforms GM for geolocation flows with just a few ($K = 10$) random environments and, unlike GM, without GPS coordinates. Specifically, GM achieves MAE results of 10.48 and 9.81 on the Bkite and FourSQ datasets, respectively. In contrast, TribeFlow-NT achieves **1.606** and **1.41** on Bkite and FourSQ, respectively. *The improvements of TribeFlow range from roughly 800% to 900% in mean absolute error.* Again, validating our results using the Kolmogorv-Smirnov test showed that TribeFlow-NT is statistically more accurate than GM ($p < 0.001$).

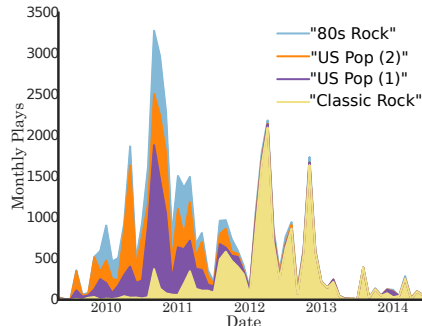
5. EXPLORATORY ANALYSIS

In this section we consider how TribeFlow can be used for sense-making in our datasets. Section 5.1 discusses the semantics of latent environments inferred by TribeFlow in our LastFM-Groups dataset, specially in the presence of non-stationary, transient, and time-heterogeneous user trajectories. Section 5.2 introduces latent environments inferred in the FourSQ dataset without GPS data. Finally, Section 5.3 discusses how TribeFlow compares with tensor decomposition approaches when uncovering meaningful patterns of user behavior from stationary user trajectory data.

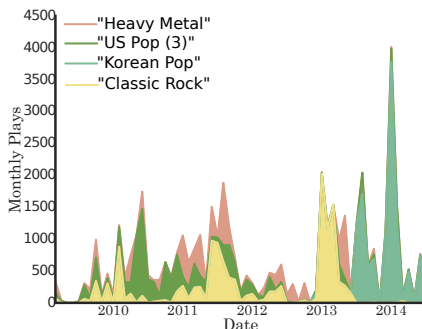
5.1 Artist-to-Artist Transitions

Figure 7 shows five latent environments inferred by TribeFlow from the LastFM-Groups dataset. Each latent environment is represented as a word-cloud of the names of the top 15 artists in the environment ranked by the random walk steady-state probability at each environment. We cross-reference the top artists at each latent environment in Figures 7(a-e) with the AllMusic guide⁷, finding that the environments discovered are semantically meaningful. It is worth

⁷<http://www.allmusic.com/>



(a) User 1



(b) User 2

Figure 9: Example of Two Users From Last.FM-Groups

noting that users in the LastFM-Groups dataset are overwhelmingly declared fans of pop music (as described in Section 4.1), but even then, TribeFlow is able to extract the user interests of very diverse musical themes such as those depicted in Figure 7.

Taking a more in-depth look at the latent environment represented in Figure 7(a) shows a sequential user trajectory preference for songs related to “Classical Music”. Some of the main artists/composers in this environment are: Chopin, Bach, Beethoven and Mozart. The latent environment in Figure 7(b) shows composers of motion picture sound tracks. John Williams, for instance, compose soundtracks for popular movies such as Star Wars, Jaws, ET and Superman. The environment in Figure 7(c) represents popular rock bands such as Nirvana and Green Day, whereas the environment in Figure 7(d) represents heavy metal bands such as Iron Maiden and Slayer. Finally, the environment in Figure 7(e) represents *electro house* music being composed of groups such as Skrillex, Pendulum, deadmau5, and Nero.

5.2 Flow Semantics in Check-in Data

We now turn our attention to the Foursquare dataset. Environments in this dataset capture user trajectories between businesses. We want see whether TribeFlow infers semantically meaningful latent environments for check-in trajectory data. Interestingly, because TribeFlow **does not use GPS** features, TribeFlow can identify latent environments of “nearby” locations in any geometry.

Figure 8 shows the top-20 locations for two different latent environments inferred by TribeFlow. Each point in the figure is a latitude-longitude coordinate plotted on the world map from its GPS coordinate. Two latent environments best exemplify the sense-making abilities of TribeFlow. The first latent environment is represented by U.S. airports and seems to capture flows of U.S. domestic flights (including Hawaii). The second latent environment captures check-in trajectories of connections to/from Pacific-Asia-based airports. Also in this environment are major U.S./U.K. major hubs (JFK in New York City (NYC), LAX in Los Angeles and Heathrow in London) that connect to Pacific-Asia. Although omitted from the figure, TribeFlow also extracted environments based on user trajectories within cities including NYC, Miami and Atlanta. In these settings, check-ins are related to different places in these cities. This illustrates that TribeFlow latent environments can be a powerful tool for sense-making in user trajectory datasets.

5.3 Transient User Trajectories

In this section we illustrate how TribeFlow can help us identify transient user trajectories (also showing that TribeFlow can cope well with the transience). To illustrate this, Figures 9(a,b) present the number of song plays (y-axis) over multiple years (x-axis) of two users from Last.FM-Groups broken down into the user’s four preferred latent environments. More precisely, the y-axis shows the cumulative sum of the song plays at a given month color-coded by the song’s most likely latent environment for that user, $P[M|u, x_t]$.

As shown in the Figure 9(a), from 2010 until mid 2011 the user goes through a strong Pop phase – most representative (top) artists in the environment labeled “U.S. Pop (1)” are *Madonna*, *Nelly Furtado*, and *Alicia Keys*, and top artists in the “U.S. Pop (2)” environment are *Britney Spears*, *Leona Lewis*, and *Kelly Clarkson*. We also note some interest in 70-80’s Rock overtones – top artists being *Queen*, *Michael Jackson*, and *The Beatles*. After mid 2011 the user moves away from Pop artists towards a “Classic Rock” environment, with *The Beatles*, *Pink Floyd*, and *Nirvana* as top artists.

The user represented in Figure 9(b) also changes interest over time, most markedly from “Classic Rock” & “Heavy Metal” to “Korean Pop”. The three major take-aways are: (a) user trajectories are indeed transient (Figure 9); (b) users can show interests in the same latent environment at different points in time: User 1 Figure 9(a) shows strong “Classic Rock” environment preference between 2011-2012 while User 2 Figure 9(b) shows strong preference for the same environment between 2013-2014; and (c) TribeFlow can cope well with transient trajectories.

To provide further evidence in support TribeFlow’s ability to extract stationary, ergodic, and time-homogeneous behavior from user trajectories consider the following synthetic dataset. For comparison, we contrast TribeFlow with a state-of-the-art temporal tensor approach in the same scenario. This comparison sheds light into the reasons why tensor-factorization-based methods should have difficulty in extracting stationary, ergodic, and time-homogeneous behavior from user trajectory data.

Our synthetic dataset has 50 users event assigned to one of five Markov chains. The Markov chains are used to create user trajectories. We model the popularity of items at each chain as a Lognormal distribution. Also, each environment has exponentially distributed inter-event times but timestamps are not recorded. We simulate a total of 5 days where and each user selects in average 100 items per day. We simulate users joining the system 1 or 2 days apart.

Applying TribeFlow-NT to this synthetic data almost perfectly recovers user preferences of transition matrices as presented in Figure 10. TribeFlow should work even better if timestamps were available, but that would be an unfair comparison because of the ex-

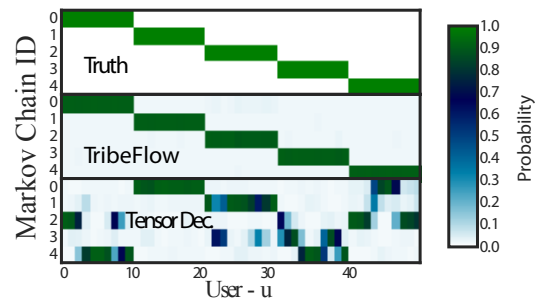


Figure 10: Comparison with Temporal Decomposition.

tra feature. A state-of-the-art temporal decomposition method [39] (tensor with time mode) on the same data has trouble finding the ground truth (i.e., user ids 0 – 9 using Markov chain 0, user ids 10 – 19 using Markov chain 1, etc). Note that only TribeFlow is able to recover the different user trajectory preferences despite the asynchronous behavior of users. Clearly this type of tensor decomposition is meant to uncover only synchronized behavior as originally proposed by Matsubara et al. [39].

6. CONCLUSIONS

In this work we introduced TribeFlow, a general method to mine and predict user trajectories. TribeFlow decomposes non-stationary, transient, time-heterogeneous user trajectories into a small number of short random walks on latent random environments. The decomposed trajectories are stationary, erratic, and time-homogeneous in short time scales. User activity (e.g., listening to music, shopping for products online or checking-in different places in a city) is then captured by different latent environments inferred solely from observed user navigation patterns (e.g., listening to “classical music”, listening to “Brazilian Pop”, shopping for shoes, shopping for electronics, checking-in into airport fast-food venues, etc.). We summarize our major contributions as follows:

- **Accurate:** TribeFlow outperforms various state-of-the-art baseline methods in three different tasks: extracting Markov embedding of user behavior, next-item prediction and capturing the flows of users between locations. Gains are up-to 900% depending on the dataset and task analyzed.
- **Scalable:** TribeFlow is at least tens and up to hundreds of times faster than state-of-the-art competitors even in relatively small datasets. If we consider the only other fully parallel competitor, MultiLME [11], TribeFlow is 413x faster and still more accurate.
- **Novelty:** TribeFlow provides a general framework to build upon (random surfer over infinite latent random environments) and make application-specific personalized recommendation systems.

Acknowledgments

Research was funded by Brazil’s National Institute of Science and Technology for Web Research (MCT/CNPq/INCT Web 573871/2008-6). Research was also sponsored by the Defense Threat Reduction Agency and was accomplished under contract No. HDTRA1-10-1-0120, as well as by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

7. REFERENCES

- [1] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing. Mixed Membership Stochastic Blockmodels. In *Proc. NIPS*, 2009.
- [2] N. Aizenberg, Y. Koren, and O. Somekh. Build your own music recommender by modeling internet radio streams. In *Proc. WWW*. ACM, 2012.
- [3] S. Alexander, J. Bernasconi, W. Schneider, and R. Orbach. Excitation dynamics in random one-dimensional systems. *Reviews of Modern Physics*, 53(2):175, 1981.
- [4] U. Asuncion, P. Smyth, and M. Welling. Asynchronous distributed estimation of topic models for document analysis. *Statistical Methodology*, 8, 2011.
- [5] L. Backstrom and J. Leskovec. Supervised random walks: Predicting and Recommending Links in Social Networks. In *Proc. WSDM*, 2011.
- [6] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3(4-5):993–1022, 2003.
- [7] M. Bryant and E. B. Sudderth. Truly Nonparametric Online Variational Inference for Hierarchical Dirichlet Processes. In *Proc. NIPS*, 2012.
- [8] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 1 edition, 2010.
- [9] A. J. Chaney, D. M. Blei, and T. Eliassi-Rad. A probabilistic model for using social networks in personalized item recommendation. In *Proc. RecSys*, 2015.
- [10] S. Chen, J. L. Moore, D. Turnbull, and T. Joachims. Playlist prediction via metric embedding. In *Proc. KDD*, 2012.
- [11] S. Chen, J. Xu, and T. Joachims. Multi-space probabilistic sequence modeling. In *Proc. KDD*, 2013.
- [12] E. Cho, S. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proc. KDD*, 2011.
- [13] S. Feng, X. Li, Y. Zeng, G. Cong, Y. M. Chee, and Q. Yuan. Personalized ranking metric embedding for next new POI recommendation. In *Proc. IJCAI*, 2015.
- [14] F. Figueiredo, J. M. Almeida, Y. Matsubara, B. Ribeiro, and C. Faloutsos. Revisit Behavior in Social Media: The Phoenix-R Model and Discoveries. In *Proc. PKDD/ECML*, 2014.
- [15] E. Fox, E. Sudderth, M. Jordan, and A. Willsky. Bayesian Nonparametric Methods for Learning Markov Switching Processes. *IEEE Signal Processing Magazine*, 27(6):43–54, Nov. 2010.
- [16] E. B. Fox, E. B. Sudderth, M. I. Jordan, and A. S. Willsky. Joint Modeling of Multiple Time Series via the Beta Process with Application to Motion Capture Segmentation. *Annals of Applied Statistics*, page 33, Nov. 2014.
- [17] S. Fröhlich-Schnatter and S. Kaufmann. Model-Based Clustering of Multiple Time Series. *Journal of Business & Economic Statistics*, 26(1):78–89, Jan. 2008.
- [18] R. García-Gavilanes, Y. Mejova, and D. Quercia. Twitter ain't without frontiers: economic, social, and cultural boundaries in international communication. In *Proc. CSCW*, 2014.
- [19] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis, Third Edition (Texts in Statistical Science)*. CRC Press, 2013.
- [20] S. Goldwater and T. L. Griffiths. A Fully Bayesian Approach to Unsupervised Part-Of-Speech Tagging. In *Proc. ACL*, 2007.
- [21] P. Gopalan, J. M. Hofman, and D. M. Blei. Scalable Recommendation with Poisson Factorization. In *Proc. UAI*, 2015.
- [22] T. Griffiths. Gibbs sampling in the generative model of Latent Dirichlet Allocation. Technical report, Stanford University, 2002.
- [23] M. Harvey, I. Ruthven, and M. J. Carman. Improving social bookmark search using personalised latent variable language models. In *Proc. WSDM*, 2011.
- [24] T. H. Haveliwala. Topic-sensitive PageRank. In *Proc. WWW*, 2002.
- [25] T. H. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):784–796, 2003.
- [26] M. D. Hoffman, D. M. Blei, and F. Bach. Online learning for latent dirichlet allocation. In *In Proc. NIPS*, 2010.
- [27] B. D. Hughes. *Random Walks and Random Environments*, volume 2. Clarendon; Oxford University Press, 1995.
- [28] G. Jeh and J. Widom. Scaling personalized web search. In *Proc. WWW*, 2003.
- [29] J.-H. Kang and K. Lerman. LA-CTR: A Limited Attention Collaborative Topic Regression for Social Media. In *Proc. AAAI*, 2013.
- [30] J.-H. Kang, K. Lerman, and L. Getoor. LA-LDA: A Limited Attention Model for Social Recommendation. In *Social Computing, Behavioral-Cultural Modeling and Prediction*. Springer, 2013.
- [31] S. Karlin and H. M. Taylor. *Stochastic processes*, 1975.
- [32] J. F. C. Kingman. *Poisson processes*, volume 3. Oxford university press, 1992.
- [33] J. Kleinberg. Bursty and Hierarchical Structure in Streams. In *Proc. ECML PKDD*, 2003.
- [34] P. N. Krivitsky, M. S. Handcock, A. E. Raftery, and P. D. Hoff. Representing degree distributions, clustering, and homophily in social networks with latent cluster random effects models. *Social Networks*, 31(3):204–213, July 2009.
- [35] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov Chains and Mixing Times*. AMS, 2008.
- [36] E. Liebman, M. Saar-Tsechansky, and P. Stone. DJ-MC: A Reinforcement-Learning Agent for Music Playlist Recommendation. In *Proc. AAMAS*, 2015.
- [37] H. Ma, H. Yang, M. R. Lyu, and I. King. SoRec : Social Recommendation Using Probabilistic Matrix Factorization. In *Proc. CIKM*, 2008.
- [38] Y. Matsubara, Y. Sakurai, and C. Faloutsos. AutoPlait: automatic mining of co-evolving time sequences. In *Proc. SIGMOD*, 2014.
- [39] Y. Matsubara, Y. Sakurai, C. Faloutsos, T. Iwata, and M. Yoshikawa. Fast mining and forecasting of complex time-stamped events. In *Proc. KDD*, 2012.
- [40] J. L. Moore, S. Chen, T. Joachims, and D. Turnbull. Taste Over Time: The Temporal Dynamics of User Preferences. In *Proc. ISMIR*, 2013.
- [41] M. F. Neuts. A versatile Markovian point process. *Journal of Applied Probability*, pages 764–779, 1979.
- [42] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. Technical report, Stanford InfoLab, 1999.
- [43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss,

- V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [44] S. Pool, F. Bonchi, and M. van Leeuwen. Description-Driven Community Detection. *ACM Transactions on Intelligent Systems and Technology*, 5(2):1–28, Apr. 2014.
- [45] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Factorizing personalized Markov chains for next-basket recommendation. In *Proc. WWW*, 2010.
- [46] M. Richardson and P. Domingos. The intelligent surfer: Probabilistic combination of link and content information in pagerank. In *Proc. NIPS*, 2001.
- [47] S. Robert and J.-Y. L. Boudec. On a Markov modulated chain exhibiting self-similarities over finite timescale. *Performance Evaluation*, 27-28:159–173, 1996.
- [48] M. G. Rodriguez, J. Leskovec, D. Balduzzi, and B. Schölkopf. Uncovering the structure and temporal dynamics of information propagation. *Netw. Sci.*, 2(01):26–65, 2014.
- [49] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *PNAS*, 105(4):1118–1123, 2008.
- [50] R. Salakhutdinov and a. Mnih. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *Proc. ICML*, 2008.
- [51] M. Sarwat, J. J. Levandoski, A. Eldawy, and M. F. Mokbel. LARS*: An Efficient and Scalable Location-Aware Recommender System. *IEEE Transactions on Knowledge and Data Engineering*, 26(6):1384–1399, 2014.
- [52] Y. Shi, M. Larson, and A. Hanjalic. Collaborative Filtering beyond the User-Item Matrix. *ACM Computing Surveys*, 47(1):1–45, May 2014.
- [53] J. M. C. S. Silva and S. Tenreiro. The Log of Gravity. *Review of Economics and Statistics*, 88(4):641–658, 2006.
- [54] A. P. Singh and G. J. Gordon. Relational learning via collective matrix factorization. In *Proc. KDD*, 2008.
- [55] C. Smith, D. Quercia, and L. Capra. Finger on the pulse: Identifying deprivation using transit flow analysis. In *Proc. CSCW*, 2013.
- [56] H. Tong, C. Faloutsos, and J. Y. Pan. Fast random walk with restart and its applications. In *Proc. ICDM*, 2006.
- [57] K. Toutanova and M. Johnson. A Bayesian LDA-Based Model for Semi-Supervised Part-Of-Speech Tagging. In *Proc. NIPS*, 2008.
- [58] D. R. Turnbull, J. A. Zupnick, K. B. Stensland, A. R. Horwitz, A. J. Wolf, A. E. Spigel, S. P. Meyerhofer, and T. Joachims. Using Personalized Radio to Enhance Local Music Discovery. In *Proc. CHI*, 2014.
- [59] P. Wang and P. Blunsom. Collapsed Variational Bayesian Inference for Hidden Markov Models. In *Proc. AISTATS*, 2013.
- [60] P. Wang, J. Guo, and Y. Lan. Modeling Retail Transaction Data for Personalized Shopping Recommendation. In *Proc. CIKM*, 2014.
- [61] Y. Wang, E. Agichtein, and M. Benzi. TM-LDA: Efficient Online Modeling of Latent Topic Transitions in Social Media. In *Proc. KDD*, 2012.
- [62] L. Wasserman. *All of Statistics : A Concise Course in Statistical Inference*. Springer Texts in Statistics, 2004.
- [63] X. Wu, Q. Liu, E. Chen, L. He, J. Lv, C. Cao, and G. Hu. Personalized next-song recommendation in online karaokes. In *Proc. RecSys*, 2013.
- [64] J. Yang and J. Leskovec. Overlapping Communities Explain Core-Periphery Organization of Networks. *Proceedings of the IEEE*, 102(12):1892–1902, Dec. 2014.
- [65] J. Yang, J. McAuley, and J. Leskovec. Community Detection in Networks with Node Attributes. In *Proc. ICDM*, 2013.
- [66] H. Yin, B. Cui, L. Chen, Z. Hu, and C. Zhang. Modeling Location-based User Rating Profiles for Personalized Recommendation. *ACM Transactions on Knowledge Discovery from Data*, To Appear, 2015.
- [67] H.-F. Yu, C.-J. Hsieh, H. Yun, S. Vishwanathan, and I. S. Dhillon. A Scalable Asynchronous Distributed Algorithm for Topic Modeling. In *Proc. WWW*, 2015.