LBSN2Vec++: Heterogeneous Hypergraph Embedding for Location-Based Social Networks

Dingqi Yang, Bingqing Qu, Jie Yang, and Philippe Cudré-Mauroux

Abstract—Location-Based Social Networks (LBSNs) have been widely used as a primary data source for studying the impact of mobility and social relationships on each other. Traditional approaches manually define features to characterize users' mobility homophily and social proximity, and show that mobility and social features can help friendship and location prediction tasks, respectively. However, these hand-crafted features not only require tedious human efforts, but also are difficult to generalize. Against this background, we propose in this paper LBSN2Vec++, a heterogeneous hypergraph embedding approach designed specifically for LBSN data for automatic feature learning. Specifically, LBSN data intrinsically forms a heterogeneous hypergraph including both user-user homogeneous edges (friendships) and user-time-POI-semantic heterogeneous hyperedges (check-ins). Based on this hypergraph, we first propose a random-walk-with-stay scheme to jointly sample user check-ins and social relationships, and then learn node embeddings from the sampled (hyper)edges by not only preserving the n-wise node proximity captured by the hyperedges, but also considering embedding space transformation between node domains to fully grasp the complex structural characteristics of the LBSN heterogeneous hypergraph. Using real-world LBSN datasets collected in six cities all over the world, our extensive evaluation shows that LBSN2Vec++ significantly and consistently outperforms both state-of-the-art graph embedding techniques by up to 68% and the best-performing hand-crafted features in the literature by up to 70.14% on friendship and location prediction tasks.

Index Terms—User mobility, Social relationship, Location-based social network, Heterogeneous hypergraph, Graph embedding

1 Introduction

Location Based Social Networks (LBSNs), such as Foursquare, have attracted millions of users and generated a considerable amount of digital footprints from their daily life. Specifically, in LBSNs, users can share their real-time presences with their friends by checking-in at a Point of Interest (POI), such as a restaurant or a gym. In addition to such fine-grained and semantic user mobility information, the social network of the corresponding users is also available. As such, LBSNs have become a primary data source to study urban dynamics [1], human mobility and social network analysis [2], [3].

Using LBSN data, two typical applications have been widely investigated, i.e., friendship prediction (a.k.a. link prediction) and location prediction. *Friendship prediction* aims at recommending social relationships that are likely to be established in the future [3], while *location prediction* tries to predict which POI a user will go to in a given context (e.g., at a given time) [2]. Due to the intrinsic correlation between human mobility and social relationships [2], [3], [4], [5], [6], existing work has shown that considering such correlation can improve the performance of both friendship prediction [3], [6], [7], [8], [9] and location prediction [2], [7], [10], [11], [12]. More precisely, these approaches usually select a set of hand-crafted features from both user mobility data and the corresponding social network, and then combine those features for friendship or location prediction. For example, social features usually involve different network proximity metrics including common neighbor, Adamic-Adar [13], Katz index [14], etc. For mobility features, co-location rate is a widely used metric to measure the homophily between two users in terms of mobility traces; it has different variations such as normalized/unnormalized, weighted/unweighted, spatial only/spatiotemporal [3], [15]. However, such a manual feature engineering process not only requires tedious efforts from domain experts, but also shows less generalizability [16] (see Section 4.2.2 and 4.3.2 for more detail).

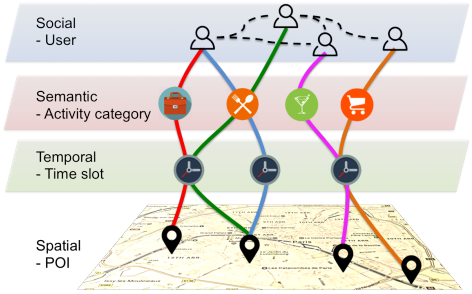


Fig. 1. A LBSN heterogeneous hypergraph containing both user mobility data (check-ins) and the corresponding social network. A friendship is represented by a homogeneous edge (a black dotted line) linking two user nodes. A check-in is represented by a *heterogeneous hyperedge* (a colored thick line) linking four nodes, i.e., a user, an activity type, a time stamp and a POI.

Against this background, automatic feature learning (a.k.a. representation learning) has been proposed to overcome the limitation of hand-crafted features [17]. When applied to networks or graphs, this paradigm is typically known as *graph (or network) embedding* [18], which represents nodes of a graph in a low-dimensional vector space while preserving key structural properties of the graph (e.g.,

Dingqi Yang is with the University of Fribourg, Switzerland and the University of Macau, SAR China. Bingqing Qu and Philippe Cudré-Mauroux are with the University of Fribourg, Switzerland, E-mail: {dingqi.yang, bingqing.qu, philippe.cudre-mauroux}@unifr.ch. Jie Yang is with Delft University of Technology, E-mail: jie@exascale.info.
Manuscript received xxx; revised xxx.

topological proximity of the nodes). Based on such embeddings, graph analysis tasks (e.g., link prediction) can then be efficiently performed. As shown in Figure 1, LBSN data in this context intrinsically form a *heterogeneous hypergraph* consisting of four different data domains, i.e., spatial, temporal, semantic and social domains. This graph contains not only classical homogeneous edges (i.e., friendships between two users in the social network), but also *heterogeneous hyperedges* (i.e., check-ins linking four nodes, one from each domain, representing a *user*'s presence at a *POI* at a specific *time* along with the *semantics* information about the user's activity there).

However, existing graph embedding techniques cannot fully grasp the complex data structure of LBSNs. First, most of the existing techniques were developed for classical graphs [19], [20], [21], [22], [23], [24], [25], where an edge links two nodes only; the node embeddings are then learnt such that the pairwise node proximity is preserved. However, preserving the pairwise node proximity cannot fully capture the information from the check-in hyperedges. Even though a hypergraph can be transformed into a classical graph by breaking each hyperedge into multiple classical edges, such an irreversible process causes a certain information loss [16], leading to degraded performance of the learnt node embeddings on different tasks (see Section 4.2 and 4.3 for more detail). Second, there are also a few techniques studying the hypergraph embedding problem, but they either focus on a n-uniform hypergraph (where all hyperedges contain a fixed number n of nodes) and thus capture only fixed-n-wise node proximity [26], [27], [28], or learn from hyperedges for heterogeneous events only (hyperedges linking nodes from different data domains) while ignoring homogeneous edges within a data domain [29]. However, as shown in Figure 1, the LBSN heterogeneous hypergraph typically contains both classical friendship edges within the user domain and check-in hyperedges across all four data domains. Subsequently, these techniques cannot be directly applied to the full LBSN heterogeneous hypergraph. Even though these techniques could be applied on check-in hyperedges only, ignoring social relationship indeed shows suboptimal performance for different tasks (see Section 4.3 for more detail). Third, the heterogeneity nature of the LBSN hypergraph implies that learning node embeddings in a unified space (as most of the existing techniques do [16], [30], [31], [32]) may not be able to fully capture the complex characteristics of the LBSN heterogeneous hypergraph. Specifically, considering a case where a popular user is connected to many friends via friendships, and this user is also connected to a POI via a check-in hyperedge while none of her friends are linked to this POI via check-ins. If we learn user and POI embeddings in a unique vector space, this user should be closely surrounded by her friends in the embedding vector space because of friendship edges, and this user should also be close to the POI because of the check-in hyperedge. However, as none of the user's friends have check-ins on this POI, the POI should be put far away from them and subsequently from the user as well. This is in conflict with fact that the user should be close to the POI because of the check-in hyperedge.

In this paper, we propose LBSN2Vec++, a heterogeneous hypergraph embedding approach designed specifically for LBSN data. Specifically, we first propose a random-walk-with-stay scheme to jointly sample friendship and check-ins from a LBSN heterogeneous hypergraph. To balance the impact of social relationships and user mobility on the learnt node embeddings, we incorporate a tunable parameter to control the proportion of social relationships and check-ins in the learning process in a probabilistic manner. Subsequently, we learn node embeddings from both sampled friendship homogeneous edges and check-in heterogeneous hyperedges by not only capturing the n-wise node proximity encoded by the hyperedges, but also considering the embedding space transformation between node domains to fully grasp the structural characteristics of the LBSN heterogeneous hypergraph. More precisely, for a social relationship connecting two users, we maximize the cosine similarity between the two corresponding user nodes in the embedding vector space. For a check-in hyperedge linking four nodes across the four data domains (as shown in Figure 1), we first transform the corresponding user node and POI node into the semantic domain via two temporal-specific projections, respectively, and then preserve the triple-wise proximity of the activity node, the transformed user and POI nodes, simultaneously. The intuition behind such a design is that, under a specific temporal context, a user has an intended activity and a POI provides a potential activity; a check-in happens when the user intended activity matches the activity provided by the POI. Analogically, in the LBSN heterogeneous hypergraph, a user node and a POI node are projected to an "intended" activity and a "potential" activity via two temporal-specific projections, respectively. Incorporating such embedding space transformations gives more flexibility to capture the complex structure of the LBSN heterogeneous hypergraph. We then learn to preserve the triple-wise proximity between the three nodes (the actual activity node and the "intended" and "potential" activity nodes). Our contributions are summarized as follows:

- We study an automatic feature learning problem for LBSN data, and propose LBSN2Vec++, a heterogeneous hypergraph embedding approach designed specifically for the LBSN heterogeneous hypergraph.
- We propose a random-walk-with-stay scheme to jointly sample friendship and check-ins from the LBSN heterogeneous hypergraph, while balancing the impact of social relationships and user mobility on the learnt node embeddings.
- We learn node embeddings from both sampled friendship edges and check-in hyperedges by not only capturing the n-wise node proximity encoded by the hyperedges, but also considering the embedding space transformation between node domains to fully grasp the complex structural characteristics of the LBSN heterogeneous hypergraph.
- We conduct a thorough evaluation using real-world LBSN datasets in six cities all over the world. Our results show that LBSN2Vec++ significantly and consistently outperforms not only state-of-the-art graph embedding techniques by up to 68%, but also the best-performing hand-crafted features in the literature by up to 70.14%, on friendship and location prediction tasks.

Moreover, using LBSN2Vec++, we discover the asymmetric impact of user mobility and social networks on predicting each other.

Compared to our previous work LBSN2Vec [16], in this paper, we further consider the heterogeneous nature of the LBSN hypergraph, and design LBSN2Vec++ leveraging embedding space transformation when learning from check-in heterogeneous hyperedges such as to more subtly capture user mobility information. Our experiment results show that compared to LBSN2Vec, LBSN2Vec++ significantly improves the performance on location prediction by 68%, while being able to maintain the same level of performance on friendship prediction. In essence, the embedding space transformation significantly helps to preserve the complex structure of the LBSN heterogeneous hypergraph.

2 Related Work

2.1 Human Mobility and Social Relationships

The interaction between human mobility and social relationships has been widely studied using different data sources over the past decade. In the earlier stage, call detail records [2], [15] and (taxi) trajectory data [33], [34] are widely used to study human mobility, social ties and link prediction problems. However, these data sourses often has a common limitation, as they do not contain actual information about social relationships between users; the corresponding social network are usually built from the users' communication activities based on certain heuristics. For example, a friendship is assumed between a pair of users when there is at least one call between them [15] (or 10 calls in [2]).

The emergence of Location-Based Social Networks provides a novel opportunity to collect both large-scale user mobility data (i.e., check-ins) and the corresponding social network [35], [36]. Using LBSN data, positive correlations between social proximity and mobility homophily have been universally found [2], [15], [16]. For example, Wang et al. [15] found positive correlations between social proximity (measured by various network proximity metrics including common neighbors, Adamic-Adar, Jaccard coefficient, and Katz index on the user social network) and mobility homophily (i.e., the similarity between mobility patterns measured by distance, spatial or spatiotemporal co-location rate under Jaccard/Cosine similarity). In this paper, we take a step forward by further investigating the impact of user check-ins and the corresponding social network on both friendship and location prediction tasks.

First, friendship prediction is a classical problem in social network analysis. It predicts the potential friendship between two users based on the existing social network [37]. Using LBSN data, friendship prediction approaches [3], [6], [7], [8], [9] often combine social proximity and mobility pattern similarity to achieve better prediction performance. For example, Scellato et al. [3] investigated a set of hand-crafted features and showed that besides social proximity, mobility pattern similarity is also a strong indicator to predict future friendship. Sadilek et al. [7] manually combined features from social proximity, mobility similarity, and textual similarity extracted from users' Tweets for friendship prediction.

Second, location prediction is a typical problem in human mobility modeling, which predicts the location of a user under a certain context based on the user's historical mobility traces. Using LBSN data, location prediction approaches incorporate social factors in mobility models, showing an improved prediction performance [2], [7], [10], [11], [12], [38]. For example, Gao et al. [11] combined check-in patterns and social ties by considering mobility similarity between friends (co-location rate measured by cosine distance). Noulas et al. [10] investigated the next place prediction problem and found that besides a user's own check-in history, a "social filtering" feature serves as an important predictor for the user's next location.

However, the manual feature engineering process adopted by the existing work not only requires a lot of human efforts, but also shows less generalizability [16], i.e., features designed for one task (e.g., friendship prediction) do not perform well in the other task (e.g., location prediction). In addition, we also note that some existing works design generative models [39], [40], [41], [42], [43] for LBSN data to perform specific tasks (most the location prediction task) without involving hand-crafted features, but those models are task-specific and cannot be applied to other tasks. More importantly, they are not able to model the impact of user mobility and social relationships on each other. In this paper, we propose LBSN2Vec++, a heterogeneous hypergraph embedding approach to automatically learn the node embeddings from a LBSN heterogeneous hypergraph, based on which both friendship and location prediction tasks can be efficiently performed. Moreover, using our proposed technique, we discovered the asymmetric impact of mobility and social relationships on predicting each other.

2.2 Graph Embeddings

Most existing graph embedding approaches focus on preserving pairwise node proximity in a classical graph, which can be further classified into two categories according to the embedding learning process. First, *factorization based approaches* [19], [20], [21] measure pairwise node proximity as a matrix using a certain network proximity metric, such as common neighbor or Adamic-Adar, and then factorize this proximity matrix using matrix factorization techniques to learn the node embeddings. However, factorization-based approaches have an intrinsic scalability limitation, due to the quadratic complexity of matrix factorization algorithms [44]. Second, *graph-sampling based approaches* [22], [23], [24], [25], [32] sample node pairs (directly or via random walks) from a graph, and then design specific models to learn node embeddings from the sampled node pairs via stochastic optimization. These graph-sampling based approaches are able to scale up to large datasets, as their complexity mainly depends on the number of the sampled node pairs.

Moreover, there are also a few approaches studying the hypergraph embedding problem, but they either focus on a homogeneous hypergraph (where all nodes with the same domain) [45], a n-uniform hypergraph (where all hyperedges contain a fixed number n of nodes) and thus capture only fixed-n-wise node proximity [26], [27], [28], or learn from hyperedges for heterogeneous event only (hyperedges linking nodes from different data domains) while ignoring

edges within a data domain [29], [46]. However, in this paper, as shown in Figure 1, our LBSN heterogeneous hypergraph contains both friendship homogeneous edges within the user domain and check-in heterogeneous hyperedges across all four data domains. Subsequently, these techniques cannot be directly applied to our full LBSN heterogeneous hypergraph. Even though they can be applied only on the check-in hyperedges for example, ignoring social relationships indeed yields suboptimal performance (see Section 4.3 for more detail). Graph embedding problems have also been studied on multiplex networks [47], where multiple networks are aligned with each other via some common nodes; it is essentially different from a hypergraph where an edge links multiple nodes.

In addition, heterogeneous graph embedding problems have also been widely studied in the current literature [16], [30], [31], [32]. These approaches mainly focus on capturing the meta-structures of a heterogeneous graph (using meta-paths, for example) and project nodes into a unified embedding space. However, as shown in our experiments 4.3, simply projecting nodes into a unified space fails to fully capture the complex structural characteristics of the LBSN heterogeneous hypergraph, resulting in unsatisfied results. Against this background, we propose LBSN2Vec++ to efficiently learn high-quality node embeddings from the LBSN heterogeneous hypergraph. It is designed to not only capture the n-wise node proximity encoded by the hyperedges, but also consider the embedding space transformation between node domains to fully grasp the heterogeneous nature of the LBSN hypergraph.

We also note that there are several works exploited embedding techniques on LBSN data, but they mostly focus on check-in data only (without using social networks) for specific tasks, such as semantic annotation of POIs [48], POI recommendation [49], [50], [51], next visitor prediction [52], user profiling [53], urban community structure discovery [54]. Only one recent work [55] used both social network and check-in data on LBSNs for learning embeddings for users and POIs, but it overlooks the temporal and semantic information of check-ins. Moreover, none of these works have investigated the correlations between human mobility and social relationships.

3 LBSN2Vec++

In this paper, we adopt a graph-sampling based embedding paradigm when designing LBSN2Vec++. Specifically, our method first uses a random-walk-with-stay scheme to jointly sample friendships and check-in hyperedges[1] from the LBSN heterogeneous hypergraph, and then learn node embeddings from these hyperedges.

3.1 Random Walk with Stay

As shown in Figure 1, the LBSN heterogeneous hypergraph consists of four data domains, i.e., spatial, temporal, semantic and social domains, and two types of edges, i.e., classical edges (friendships) linking pairs of user nodes and hyperedges (check-ins) linking four nodes, one from each domain. Formally, the social domain contains a set of users U; the semantic domain contains a set of activity category C; the temporal domain contains a set of time slots T (following our previous empirical study [16], we define in this work the time granularity as 168 hours/time slots in a full week); the spatial domain contains a set of POIs L. A friendship can then be represented as a pair of users (u_i, u_j), $u_i, u_j \in U$. A check-in hyperedge can be represented as a quadruple (u, c, t, l), $u \in U, c \in C, t \in T, l \in L$.

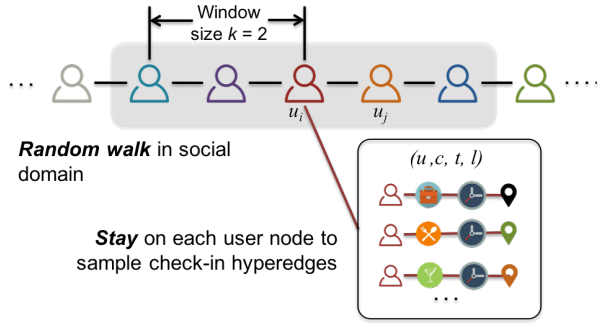
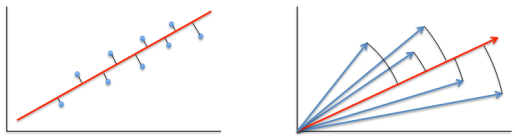


Fig. 2. Random walk with stay on the LBSN hypergraph

To sample (hyper)edges from this graph, we propose a random-walk-with-stay scheme to jointly sample friendship and check-in hyperedges. As shown in Figure 2, our random-walk-with-stay scheme performs classical random walk only on user nodes based on their friendships in social domain, while for each encountered user node, it stays on the user node to sample a set of hyperedges (check-ins) from the corresponding user. Subsequently, in the node embedding learning process, when iterating over each user node in a random walk sequence, we not only learn from two user nodes appearing with a context window of length k, but also stay on the corresponding user node to learn from its check-in hyperedges. In other words, the node embedding learning process alternates between these two types of edges (i.e., friendships and check-ins). To perform classical random walk on the user social network, we use the same strategy as used by existing works [22], [24], generating r walks of length l rooted on each user node.

Moreover, in order to balance the impact of friendships and check-ins on the learnt node embeddings, LBSN2Vec++ incorporates a tunable parameter α to control the portion of learning edges of each type. Specifically, for a given context window of length k, we iterate over $2k$ contexts for each user node u_i (k context nodes on each side), resulting in $2k$ pairs of user nodes. Figure 2 shows an example where the context window size is 2. Subsequently, we sample for each user node the same number ($2k$) of check-ins using a random sampling with replacement strategy, to ensure the exact same number of check-ins and friendships are sampled per user. Based these samples, we can then balance the impact of friendships and check-ins on the learnt node embeddings using a unique parameter α. Specifically, we learn from each check-in (u, c, t, l) with a probability α, while from each user node pair (u_i, u_j) with a probability $(1 - \alpha)$, where α and $(1 - \alpha)$ actually specify the impact of

1. A classical edge linking two user nodes (i.e., a friendship) can be regarded as a special case of a hyperedge consisting of only two nodes. In the following, besides the check-in hyperedges, we also use the term "hyperedge" to describe a pair of user nodes.



(a) Euclidean space with perpendicular distance (b) Cosine space with cosine distance

Fig. 3. Examples of the best-fit-lines in Euclidean and Cosine spaces. The given data points are represented in blue, while the corresponding best-fit-line is shown in red.

mobility and social network on the learnt node embeddings, respectively. In summary, for each node v_i in a random walk, the expected total number of learnt edges is $2k$, including an expected number $2k\alpha$ of check-ins and an expected number of $2k(1-\alpha)$ of user node pairs. A small value of α gives less importance on the check-in data and more on social network, and vice versa. In the following, we present our node embedding learning process from both sampled friendship[2] and check-in hyperedges.

3.2 Learning from Hyperedges

From the sampled friendship and check-in hyperedges, we design a customized embedding model to preserve not only the social proximity between users in the social network, but also user activity patterns from the check-in data. In the following, we first propose a general technique of learning node embeddings to preserve n-wise node proximity, which serves as the foundation of our LBSN2Vec++. Afterwards, we separately discuss our embedding learning process for both friendship and check-in hyperedges.

3.2.1 Preserving n-wise Node Proximity via Best-Fit-Line

As the foundation of our LBSN2Vec++, we design a general embedding technique to preserve the n-wise node proximity captured by a hyperedge containing n nodes, where $\{n \in \mathbb{Z}^+ | n \geq 2\}$. Specifically, to learn node embeddings from a hyperedge containing n nodes, we borrow the idea of best-fit-line (a.k.a. line of best fit) from linear regression [56]. In general, a best-fit-line is a straight line that is the best approximation of the given data points. Figure 3(a) illustrates an example of the best-fit-line in Euclidean space that minimizes the sum of perpendicular distances, which can be computed via linear least squares. In this paper, following existing graph embedding techniques [19], [20], [21], [22], [23], [24], [25], we learn node embeddings in cosine space where the proximity between nodes are measured using cosine similarity (or dot product of the normalized embedding vectors). Moreover, we show below that the computation of the best-fit-line in cosine space can be significantly simplified. Formally, the best-fit-line of a set of node embeddings is the vector that minimizes the sum of cosine distances between each node embedding and the best-fit-line, as shown in Figure 3(b), which can be efficiently computed using Proposition 1 (please refer to [16] for its proof):

2. We use the term "friendship" hereafter to refer to pairs of user nodes sampled using our random walk with stay techniques.

Proposition 1. *For a set of nodes (in a hyperedge) $\{v_i | i = 1, 2, \ldots, n\}$, the corresponding best-fit-line can be computed as*

$$\vec{v}_b = \sum_{i=1}^{n} \frac{\vec{v}_i}{\|\vec{v}_i\|} \tag{1}$$

where $\vec{v}_i$ refer to the embedding vector of node v_i.

Based on Proposition 1, we optimize the n-wise node proximity by iteratively maximizing the cosine similarity between each node embedding $\vec{v}_i$ from the hyperedge $\{v_i | i = 1, 2, \ldots, n\}$ and the best-fit-line $\vec{v}_b$.

$$\Theta = \sum_{i=1}^{n} \cos(\vec{v}_i, \vec{v}_b) \tag{2}$$

Note that here we keep using cosine similarity rather than dot product (which is widely used by existing techniques), because cosine similarity is not affected by the norm of the input vectors, in particular when the norm of the best-fit-line $\vec{v}_b$ computed by Eq. 1 has a large variance due to the variant number of the nodes n in different hyperedges.

In addition, similar to other graph-sampling based embedding techniques [22], [23], [24], [25], we also adopt a negative sampling technique to maximize the cosine distance between each negative sample node v_N and the best-fit-line vector $\vec{v}_b$. Finally, we want to maximize the following objective function for one node v_i in the hyperedge $\{v_i | i = 1, 2, \ldots, n\}$:

$$\Theta = \cos(\vec{v}_i, \vec{v}_b) + \gamma \cdot \mathbb{E}_{v_N}[1 - \cos(\vec{v}_N \cdot \vec{v}_b)] \tag{3}$$

where $\gamma \in \mathbb{Z}^+$ is the number of negative samples. Note that as the LBSN hypergraph contains four data domains, the negative samples for a node v_i from one data domain are randomly sampled from the nodes in the same data domain. Such a negative sampling strategy ensures the negative samples following the hyperedge structure. For example, for a given hyperedge (u, c, t, l), a negative sample for the POI node l should be another POI node $l_N \in L$. Otherwise, if a negative node is taken from user domain for example, resulting in (u, c, t, u_N), it breaks the intrinsic structure of check-in hyperedges.

The above objective function can be optimized using Stochastic Gradient Descent (SGD). Note that we only need to update $\vec{v}_i$ and $\vec{v}_N$ when learning from one specific hyperedge, as the best-fit-line vector $\vec{v}_b$ is fixed. The gradient of the above objective function with respect to $\vec{v}_i$ and $\vec{v}_N$ is computed as follows:

$$\frac{\partial \Theta}{\partial \vec{v}_i} = \frac{\vec{v}_b}{\|\vec{v}_i\|\|\vec{v}_b\|} - \frac{\vec{v}_i \cos(\vec{v}_i, \vec{v}_b)}{\|\vec{v}_i\|^2} \tag{4}$$

$$\frac{\partial \Theta}{\partial \vec{v}_N} = -\frac{\vec{v}_b}{\|\vec{v}_N\|\|\vec{v}_b\|} + \frac{\vec{v}_N \cos(\vec{v}_N, \vec{v}_b)}{\|\vec{v}_N\|^2} \tag{5}$$

In summary, the above technique is able to learn node embeddings preserving the n-wise node proximity captured by a hyperedge containing an arbitrary number of n nodes, $\{n \in \mathbb{Z}^+ | n \geq 2\}$. Our previous work [16] has shown the effectiveness of this technique in learning hypergraph embeddings. In the following, we separately discuss the embedding learning process of our LBSN2Vec++ for both friendship and check-in hyperedges, based on the above technique.

Algorithm 1 Learning from a sampled friendship edge

Input: a friendship edge (u_i, u_j), number of negative samples γ

1: Compute the best-fit-line using Eq. 1
2: Update $\vec{u}_i$, $\vec{u}_j$ using the gradients computed by Eq. 4
3: **loop** γ times
4: Sample a negative user node $u_N \in U, u_N \neq u_i, u_j$
5: Update $\vec{u}_N$ using the gradient computed by Eq. 5
6: **end loop**

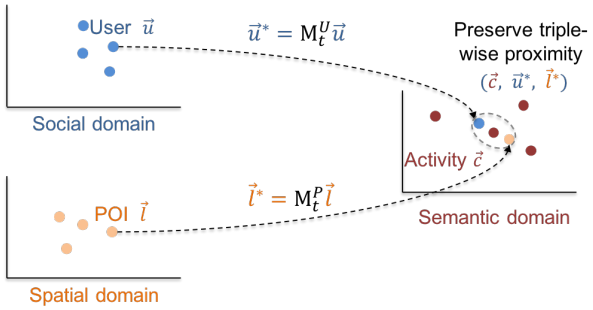


Fig. 4. Learning from a check-in hyperedge with two temporal-specific projections.

3.2.2 Learning from Friendship Edges

For a sampled friendship edge connecting two users, we directly adopt the above technique to maximize the cosine similarity between the two corresponding user nodes in the embedding vector space. Alg. 1 shows the learning process for a sampled friendship edge (u_i, u_j). We start by computing the best-fit-line of the input edge using Eq. 1 (Line 1). We then update the embedding vectors of the two user nodes $\vec{u}_i$, $\vec{u}_j$ using SGD with the gradients computed by Eq. 4 (Line 2). Meanwhile, we also sample γ negative user nodes and update their embedding vectors with the gradients computed by Eq. 5 (Line 3-6).

3.2.3 Learning from Check-in Hyperedges

For a sampled check-in hyperedge linking four nodes across the four heterogeneous data domains (as shown in Figure 1), we first transform the corresponding user node and POI node into the semantic domain via two temporal-specific projections, respectively, and then preserve the triple-wise proximity of the activity node, the transformed user and POI nodes, simultaneously. The intuition behind such a design is that, under a specific temporal context, a user has an intended activity and a POI provides a potential activity; a check-in happens when the user intended activity matches the activity provided by the POI. Analogically, in the LBSN hypergraph, a user node and a POI node are projected to an "intended" activity and a "potential" activity via two temporal-specific projections. We then learn to preserve the triple-wise proximity between three nodes (the actual activity node and the "intended" and "potential" activity nodes).

Figure 4 illustrates the learning process from a check-in hyperedge. For a check-in hyperedge (u, c, t, l), $u \in U, c \in C, t \in T, l \in L$, we first transform the user and POI node embeddings ($\vec{u}$ and $\vec{l}$, respectively), using two temporal specific projections M_t^U and M_t^L. The transformed user and POI node embeddings are denoted as $\vec{u}^* = \mathrm{M}_t^U \vec{u}$ and $\vec{l}^* = \mathrm{M}_t^L \vec{l}$, respectively. Afterward, we adopt our technique proposed in Section 3.2.1 to preserve triple-wise proximity of the activity node embedding $\vec{c}$, the transformed user node embedding $\vec{u}^*$ and the transformed POI node embedding $\vec{l}^*$. Specifically, by computing the best-fit-line of the hyperedge $\{\vec{c}, \vec{u}^*, \vec{l}^*\}$, we maximize the similarity between each (transformed) node embeddings and the best-fit-line under cosine similarity via SGD. While the objective function for the activity node $\vec{c}$ remains the same as in Eq. 4, the one for the transformed nodes needs to be derived w.r.t. the original user node embedding $\vec{u}$ and the corresponding transformation matrix M_t^U, which we formulate as follows (similar for $\vec{l}$ and M_t^L):

$$\begin{aligned}\Theta = &\cos(\vec{u}^*, \vec{v}_b) \\ &+ \gamma \cdot \mathbb{E}_{u_N}[1 - \cos(\vec{u}_N^* \cdot \vec{v}_b)] \\ &+ \gamma \cdot \mathbb{E}_{t_N}[1 - \cos(\vec{u}_{t_N}^* \cdot \vec{v}_b)]\end{aligned} \tag{6}$$

where we consider negative samples for both user node ($u_N \in U, u_N \neq u$) and time node ($t_N \in T, t_N \neq t$), resulting two transformed negative user nodes $\vec{u}_N^* = \mathrm{M}_t^U \vec{u}_N$ and $\vec{u}_{t_N}^* = \mathrm{M}_{t_N}^U \vec{u}$, respectively. Subsequently, the gradient w.r.t. the transformed nodes needs to be further derived w.r.t. the original node embedding vector $\vec{u}$ and the transformation matrix M_t^U via the chain rule. The gradient of the above objective function w.r.t. $\vec{u}$ is computed as follows:

$$\frac{\partial \Theta}{\partial \vec{u}} = \frac{\partial \Theta}{\partial \vec{u}^*} \cdot \frac{\partial \vec{u}^*}{\partial \vec{u}} + \frac{\partial \Theta}{\partial \vec{u}_{t_N}^*} \cdot \frac{\partial \vec{u}_{t_N}^*}{\partial \vec{u}} \tag{7}$$

$$\frac{\partial \Theta}{\partial \vec{u}^*} \cdot \frac{\partial \vec{u}^*}{\partial \vec{u}} = \left(\frac{\vec{v}_b}{\|\vec{u}^*\| \|\vec{v}_b\|} - \frac{\vec{u}^* \cos(\vec{u}^*, \vec{v}_b)}{\|\vec{u}^*\|^2} \right) \cdot \mathrm{M}_t^U \tag{8}$$

$$\frac{\partial \Theta}{\partial \vec{u}_{t_N}^*} \cdot \frac{\partial \vec{u}_{t_N}^*}{\partial \vec{u}} = -\left(\frac{\vec{v}_b}{\|\vec{u}_{t_N}^*\| \|\vec{v}_b\|} - \frac{\vec{u}_{t_N}^* \cos(\vec{u}_{t_N}^*, \vec{v}_b)}{\|\vec{u}_{t_N}^*\|^2} \right) \cdot \mathrm{M}_{t_N}^U \tag{9}$$

Here we separate the gradient computation of the transformed positive ($\vec{u}^*$) and negative samples ($\vec{u}_{t_N}^*$), as the latter actually depends on the negative time sample $\mathrm{M}_{t_N}^U$, which will be repeatedly sampled γ times for each check-in hyperedge in the learning process (see Alg. 2 for more detail). Similarly, the gradient w.r.t. M_t^U is computed as follows:

$$\frac{\partial \Theta}{\partial \mathrm{M}_t^U} = \frac{\partial \Theta}{\partial \vec{u}^*} \cdot \frac{\partial \vec{u}^*}{\partial \mathrm{M}_t^U} + \frac{\partial \Theta}{\partial \vec{u}_N^*} \cdot \frac{\partial \vec{u}_N^*}{\partial \mathrm{M}_t^U} \tag{10}$$

$$\frac{\partial \Theta}{\partial \vec{u}^*} \cdot \frac{\partial \vec{u}^*}{\partial \mathrm{M}_t^U} = \left(\frac{\vec{v}_b}{\|\vec{u}^*\| \|\vec{v}_b\|} - \frac{\vec{u}^* \cos(\vec{u}^*, \vec{v}_b)}{\|\vec{u}^*\|^2} \right) \otimes \vec{u} \tag{11}$$

$$\frac{\partial \Theta}{\partial \vec{u}_N^*} \cdot \frac{\partial \vec{u}_N^*}{\partial \mathrm{M}_t^U} = -\left(\frac{\vec{v}_b}{\|\vec{u}_N^*\| \|\vec{v}_b\|} - \frac{\vec{u}_N^* \cos(\vec{u}_N^*, \vec{v}_b)}{\|\vec{u}_N^*\|^2} \right) \otimes \vec{u}_N \tag{12}$$

where $\otimes$ denotes the outer product between two vectors. Similarly, the gradient w.r.t. negative samples are computed as follows:

$$\frac{\partial \Theta}{\partial \vec{u}_N} = -\left(\frac{\vec{v}_b}{\|\vec{u}_N^*\| \|\vec{v}_b\|} - \frac{\vec{u}_N^* \cos(\vec{u}_N^*, \vec{v}_b)}{\|\vec{u}_N^*\|^2} \right) \cdot \mathrm{M}_t^U \tag{13}$$

$$\frac{\partial \Theta}{\partial \mathrm{M}_{t_N}^U} = -\left(\frac{\vec{v}_b}{\|\vec{u}_{t_N}^*\| \|\vec{v}_b\|} - \frac{\vec{u}_{t_N}^* \cos(\vec{u}_{t_N}^*, \vec{v}_b)}{\|\vec{u}_{t_N}^*\|^2} \right) \otimes \vec{u} \tag{14}$$

Algorithm 2 Learning from a sampled check-in hyperedge

Input: a check-in hyperedge (u, c, t, l), number of negative samples γ

1: Get the transformed user node $\vec{u}^* = \mathrm{M}_t^U \vec{u}$
2: Get the transformed POI node $\vec{l}^* = \mathrm{M}_t^L \vec{l}$
3: Compute the best-fit-line $\vec{v}_b$ for $\{\vec{c}, \vec{u}^*, \vec{l}^*\}$ via Eq.1
4: Update $\vec{c}$ using the gradient in Eq.4
5: Update $\vec{u}$ (and $\vec{l}$) using the gradients in Eq.8
6: Update M_t^U (and M_t^L) using the gradients in Eq.11
7: **loop** γ times
8: Sample a negative activity node $c_N \in C$, $c_N \neq c$
9: Update $\vec{c}_N$ using the gradient in Eq. 5
10: Sample a negative user node $u_N \in U$, $u_N \neq u$
11: Update $\vec{u}_N$ using the gradient in Eq. 13
12: Update M_t^U using the gradient in Eq. 12
13: Sample a negative POI node $l_N \in L$, $l_N \neq p$
14: Update $\vec{l}_N$ using the gradient in Eq. 13
15: Update M_t^L using the gradient in Eq. 12
16: Sample a negative time node $t_N \in T$, $t_N \neq t$
17: Update $\mathrm{M}_{t_N}^U$ (and $\mathrm{M}_{t_N}^L$) using the gradients in Eq. 14
18: Update $\vec{u}$ (and $\vec{l}$) using the gradients in Eq. 9
19: **end loop**

The objective function and gradient are formulated in the same way for $\vec{l}$ and M_t^L, which are omitted for the sake of brevity. Hereafter, we also use the above equations for the gradients w.r.t. $\vec{l}$ and M_t^L (See the algorithm below).

Alg. 2 shows the learning process for a sampled check-in hyperedge (u, c, t, l). We start by transforming the user node and POI node into the semantic domain via two temporal-specific projections, and obtain the transformed user and POI node embeddings $\vec{u}^*$ and $\vec{l}^*$, respectively (Line 1-2). We then update the embeddings to preserve the triple-wise proximity of the activity node embedding $\vec{c}$, and the transformed user and POI node embeddings $\vec{u}^*$ and $\vec{l}^*$, using our technique proposed in Section 3.2.1. Specifically, we start by computing the best-fit-line of the hyperedge $\{\vec{c}, \vec{u}^*, \vec{l}^*\}$ using Eq. 1 (Line 3). Subsequently, we update the activity node embedding $\vec{c}$, the user and POI node embeddings ($\vec{u}$ and $\vec{l}$, respectively), and the corresponding temporal-specific project matrices (M_t^U and M_t^L, respectively) (Line 4-6). Meanwhile, we repeat the negative sampling process γ times. For each round, we perform negative sampling and update the corresponding embedding vectors for an activity node c_N (Line 8-9), a user node u_N (Line 10-12), a POI node l_N (Line 13-15), and a time node t_N (Line 16-18). Note that for the negative user node u_N, we update not only u_N, but also M_t^U, since the gradient of the objective function also depends on M_t^U as shown in Eq. 12. Similarly, we also update M_t^L for the negative POI node l_N. In addition, for the negative time node t_N, we update not only $\mathrm{M}_{t_N}^U$ (and $\mathrm{M}_{t_N}^L$), but also $\vec{u}$ (and $\vec{l}$), since the gradient of the objective function also depends on $\vec{u}$ (and $\vec{l}$) as shown in Eq. 9.

In summary, to learn node embeddings from the LBSN heterogeneous hypergraph, we first sample a set of friendship edges and check-in hyperedges using our proposed random-walk-with-stay scheme, and then alternatively learn from each friendship edge and check-in hyperedge (using algorithm 1 and 2, respectively).

TABLE 1
Statistics of the selected datasets

Dataset	NYC	TKY	IST	JK	KL	SP
#User	4,024	7,232	10,367	6,395	6,432	3,954
#POI	3,628	10,856	12,693	8,826	10,817	6,286
#Check-ins	105,961	699,324	908,162	378,559	526,405	249,839
#Friendships (before)	8,723	37,480	21,354	11,207	16,161	9,655
#Friendships (after)	10,545	51,704	36,007	16,950	31,178	14,402

4 EXPERIMENTS

In this section, we evaluate LBSN2Vec++ on both friendship and location/activity prediction tasks. In the following, we first present our experimental setup, and then discuss the results on individual tasks, followed by a study on the trade-off between social relationship and mobility on predicting each other.

4.1 Experimental Setup

4.1.1 Dataset

We use a large-scale and long-term LBSN dataset collected by [16]. It contains not only check-ins of a set of users over a two-year period (from Apr. 2012 to Jan. 2014), but also two snapshots of the corresponding user social network before and after the check-in data collection period, respectively. The chronological order of the two social networks provides a unique opportunity to study the impact of user mobility and the corresponding social network on predicting each other (see Section 4.2 and 4.3 below). More precisely, we investigate the friendship and location prediction at an urban scale. Without loss of generality, we select six cities with a large number of check-ins, while also considering the cultural diversity of the selected cities: New York City (**NYC**), Tokyo (**TKY**), Istanbul (**IST**), Jakarta (**JK**), Kuala Lumpur (**KL**), Sao Paulo (**SP**). We select users in the largest connected components of the old social network. Table 1 summarizes the statistics of the selected datasets.

4.1.2 Baselines

We compare our method to the following state-of-the-art graph embedding methods from three categories:

- *Classical graph embedding techniques for homogeneous graphs*. **DeepWalk** [22] feeds node sequences from random walks to a SkipGram model to output node embeddings. We set the walk length $l = 80$, the number of walks per node $r = 10$, and the context window size $k = 10$. **Node2Vec** [24] extends DeepWalk by introducing a parameterized random walk method to balance the breadth-first search (return parameter p) and depth-first search (in-out parameter q) strategies, to capture richer graph structures. We tune p and q with a grid search over $p, q \in \{0.25, 0.05, 1, 2, 4\}$, and keep the other parameters same as for DeepWalk. **LINE** [23] separately learns from 1st and 2nd-order node proximity in a graph, and then concatenates them together. **NetMF** [21] derives the closed form of DeepWalk's implicit matrix, and factorizes this matrix to output node embeddings. We tune the implicit window size T within $\{1, 10\}$. **VERSE** [25] directly samples node pairs to learn graph embeddings to preserve

the node proximity measured by personalized PageRank. We tune the damping factor α of personalized PageRank using the method suggested by the authors.

- *Heterogeneous graph embedding techniques*. **Metapath2Vec** [30] first generates meta-path guided random walks based on a given meta-path, then feeds them to a SkipGram model. For our LBSN heterogeneous hypergraph, we empirically use a meta-path "user-activity-time-POI-time-activity-user". **Hin2Vec** [31] automatically combines all meta-paths shorter than a predefined length, in order to jointly learn both node embeddings and meta-path embeddings. We empirically set the maximum length of meta-paths to 7, which is the length of the meta-path used in Metapath2Vec, for a fair comparison.
- *Heterogeneous hypergraph embedding techniques*. **HEBE** [29] learns node embeddings for *heterogeneous event data* using tensor modeling, where a heterogeneous hyperedge links nodes across different domains only and is modeled as an entry in an high-order tensor. **DHNE** [28] uses a deep autoencoder to learning node embeddings from a *n-uniform heterogeneous hypergraph*. It preserves both fixed-n-wise node proximity encoded by the hyperedges, and also the high-order node proximity defined as the similarity between nodes' neighborhood structures. **LBSN2Vec** [16] is our previously proposed technique. It uses the same random-walk-with-stay scheme as LBSN2Vec++ to sample friendship and check-in hyperedges from a LBSN hypergraph, but learn node embeddings *without considering the heterogeneous nature of the LBSN hypergraph*. Specifically, it learns to preserve the n-wise node proximity using the general technique described in Section 3.2.1 only, without involving the embedding space transformation in learning from check-in hyperedges (see Section 3.2.3).

For our method LBSN2Vec/LBSN2Vec++, we keep the same random walk parameters as for DeepWalk and Node2vec ($l = 80$, $r = 10$, $k = 10$), and tune the parameter α within $[0, 1]$ with a step of 0.1 to balance the impact of social and mobility on the learnt embeddings (more discussion on this point in Section 4.5). The dimension of the node embeddings d is set to 128 and the number of negative samples γ is set to 10 for all methods in all experiments, if not specified otherwise. The implementation of LBSN2Vec/LBSN2Vec++ and the used datasets are available online[3].

4.1.3 Dataset Configuration

Except **LBSN2Vec/LBSN2Vec++**, other baseline methods cannot directly take the LBSN heterogeneous hypergraph as an input graph. Therefore, we propose three settings to adapt our LBSN heterogeneous hypergraph:

- **(S)**: Only the Social network is considered by keeping nodes in user domain and their friendship edges. This setting can be applied to classical graph embedding techniques for homogeneous graphs, i.e., **DeepWalk**, **Node2vec**, **LINE**, **NetMF** and **VERSE**.
- **(M)**: Only the user Mobility is considered by keeping check-in hyperedges only. This setting can be directly applied to **HEBE** and **DHNE** which can take heterogeneous check-in hyperedges as input. For **Metapath2Vec** and **Hin2Vec**, we break each heterogeneous hyperedge user-activity-time-POI into three edges user-activity, activity-time and time-POI, resulting in a classical heterogeneous graph, which can be used as inputs to these techniques. In addition, we also apply this setting to **DeepWalk**, **Node2vec**, **LINE**, **NetMF** and **VERSE** by breaking each check-in hyperedge into multiple classical edges linking each pair of nodes in the hyperedge (e.g., user-time, time-POI, etc.).
- **(S&M)**: The full LBSN heterogeneous hypergraph is considered, and converted to a classical graph by 1) keeping friendship edges and 2) also breaking each check-in hyperedge into multiple classical edges linking each pair of nodes in the hyperedge (same as for the M setting). This setting can be applied to **DeepWalk**, **Node2vec**, **LINE**, **NetMF**, **VERSE**, **Metapath2Vec** and **Hin2Vec**.

3. https://github.com/eXascaleInfolab/LBSN2Vec

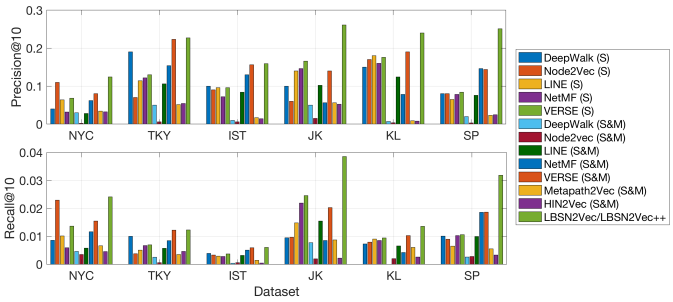


Fig. 5. Friendship prediction performance comparison with other graph embedding methods under setting setting S and S&M.

4.2 Friendship (link) Prediction

Friendship prediction recommends friendships that will probably be established in the future. In this study, we advocate an unsupervised friendship prediction approach [37], [57] which generates a ranking list of potential links between pairs of users. Specifically, after learning the node embeddings based on the *old* social network and check-ins, we rank pairs of user nodes (not being friends in the old social network) according to the cosine similarity between their embeddings. We then evaluate the obtained ranking list against the new friendships (appearing in the new but not in the old social network), reporting precision and recall on the top 10 predicted friendships [37]. As the number of candidate pairs of nodes is too large, we randomly sample 1% pairs of nodes for the evaluation, and report the average results from 10 independent trials.

4.2.1 Comparison with other graph embedding methods

We report only the methods including social networks (setting S and S&M), since the social proximity in the old social network is indeed the primary predictor of new friendships [16] and since we found out that methods using setting M perform very poorly on this task. Figure 5 shows the results. Note that we report the results together for LBSN2Vec/LBSN2Vec++, since there is no visible difference between their performance. This is reasonable because both of the methods optimize the same objective function when learning user node embeddings from user social relationships. We clearly observe that LBSN2Vec/LBSN2Vec++ achieve the highest precision and recall in general. For

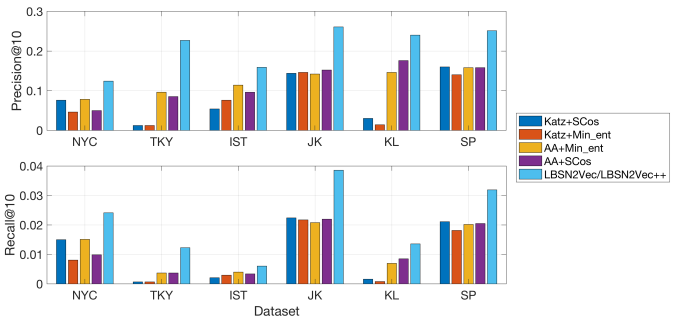


Fig. 6. Friendship prediction performance comparison with hand-crafted features.

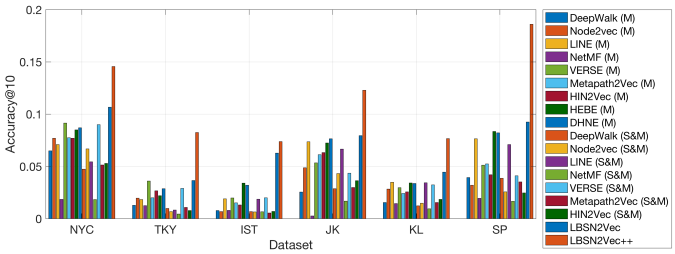


Fig. 7. Location prediction performance comparison with other graph embedding methods on all test check-ins.

example, LBSN2Vec/LBSN2Vec++ outperform all baselines with an average improvement of 62.56% on precision@10 over the best baselines across different datasets.

One interesting observation is that for all the baseline methods, considering both the social network and mobility (setting S&M) leads to worse performance than considering only the social network (setting S) in most cases. In other words, for all baseline methods, adding user mobility information decreases the friendship prediction performance. Despite the counter-intuitive nature of this observation, we notice that these methods fail to balance the impact of social network and user mobility on the learnt node embeddings. More precisely, as the number of check-ins is usually much larger than the number of friendships, the S&M graph is dominated by the edges representing check-ins. When the baseline methods uniformly sample edges from the S&M graph to learn the node embeddings, the sampled edges are also dominated by the edges representing check-ins, which naturally imposes a strong impact of mobility on the learnt node embeddings. In fact, as reported by previous studies [16], the social proximity in the old social network is indeed the primary predictor of new friendships (our parameter sensitivity study in Section 4.5 also verifies this point below). Therefore, the learnt node embeddings from those baseline methods (with setting S&M) result in the worst performance in friendship prediction.

4.2.2 Comparison with hand-crafted features

We also compare our method with the following hand-crafted features suggested specifically for the friendship prediction task by previous work:

- *Katz Index (**Katz**)* [14] is the best performing social feature suggested by [15]. It is defined as the weighted sum of all paths between two users on the social networks, where the weight of a path decays exponentially with its length. $Katz(u,v) = \sum_{l=1}^{\infty} \beta^l \cdot |path_{u,v}^l|$, where $path_{u,v}^l$ is the set of all paths with length l from u to v. We set $\beta = 0.05$ as suggested by [15].
- *Adamic-Adar (**AA**)* [13] is the best performing social feature suggested by [3]. It is the weighted sum of the common neighbors between two users. The weight for each common neighbor is the inverse logarithm of its degree. $AA(u,v) = \sum_{z \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log |\Gamma(z)|}$, where $\Gamma(u)$ is the set of neighbors (friends) of user u.
- *Spatial Cosine Similarity (**SCos**)* is the best performing mobility feature suggested by [15]. It is defined as the cosine similarity between two users' check-in vectors C_u and C_v, i.e., $\cos(C_u, C_v)$, where each element $C_u(i)$ is u's check-in count on POI i.
- *Minimum place entropy (**Min_ent**)* is the best performing mobility feature suggested by [3]. It is defined as the minimum POI entropy (i.e., the entropy of the empirical distribution of check-ins at that POI over users) that two users share.

As combining social and mobility features is suggested by both [15] and [3], we consider all the combinations of the above social and mobility features, including **Katz+SCos**, **Katz+Min_ent**, **AA+Min_ent** and **AA+SCos**. Specifically, as these features have different value ranges, we first generate one ranking list of predicted friendships using each feature, and then merge two ranking lists using reciprocal rank fusion [58], which is a popular rank fusion method widely used in information retrieval research. Figure 6 shows the results. We clearly observe that LBSN2Vec/LBSN2Vec++ outperforms the hand-crafted feature based methods in general, and we find an average improvement of 70.14% on precision@10 over the best baselines across different datasets. Moreover, We observe that none of the four combinations can consistently outperform others over all datasets, which shows the generalization limitations of the hand crafted features across different datasets.

4.3 Location Prediction

Location prediction tries to predict the POI a user will be located in at a given time slot. To implement this task, we chronologically split our check-in data into two parts, i.e., the first 80% for training and the last 20% for testing. We then learn node embeddings from the old social network and the training check-in data. Based on the learnt node embeddings, for each test check-in (u, c, t, l_{true}), we rank all POIs $l \in L$ according to the cosine similarity between the transformed user node embeddings $\vec{u}^* = \mathrm{M}_t^U \vec{u}$ and the transformed POI node embedding $\vec{l}^* = \mathrm{M}_t^L \vec{l}$. Subsequently, we evaluate the obtained POI ranking list against the actual POI in the test check-in. We report the average accuracy@10 over all test check-ins.

4.3.1 Comparison with other graph embedding methods

Figure 7 shows the results. Note that only settings involving check-in data (setting M and S&M) are eligible for this task. First, we observe that our LBSN2Vec++ consistently and significantly outperforms all baselines. In particular, LBSN2Vec++ shows an average improvement of 68% on accuracy@10 over the best-performing baseline LBSN2Vec (our previous work without considering the embedding

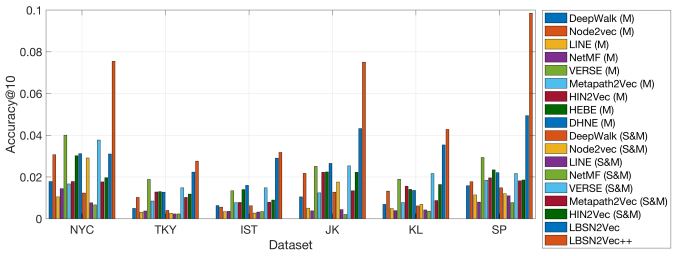


Fig. 8. Location prediction performance comparison with other graph embedding methods on new POIs.

space transformation when learning from check-in hyperedges) across different datasets. Second, we see that using setting M, hypergraph embedding techniques (HEBE and DHNE) achieve better performance than other baselines in general, as the irreversible process of transforming a hypergraph into a classical graph causes a certain information loss, leading to degraded performance for the location prediction task. Finally, similar to friendship prediction, we also find that for all baselines considering both the social network and mobility (setting S&M) results in worse performance than considering mobility (M) only in general, as the baseline methods fail to balance the impact of the social network and user mobility on the learnt node embeddings. However, the performance drop is relatively smaller than that of the friendship prediction task, as the S&M graph is actually dominated by the edges from the check-ins.

In addition, we also investigate the prediction performance on new POIs only, i.e., the POIs appearing in the test check-ins but not in the training check-ins. As historical (training) check-ins of a user are represented as hyperedges in the LBSN heterogeneous hypergraph, check-ins with new POIs actually refer to the non-observed hyperedges that will be established in the future. Therefore, this task is more difficult as it requires to explore the the hypergraph structure for predicting potential hyperedges, rather than preserving the observed hypergraph structure. Figure 8 shows the accuracy@10 for only new POIs. We observe that our LBSN2Vec++ consistently and significantly outperforms all baselines with an average improvement of 52.6% on accuracy@10 over the best-performing baseline LBSN2Vec across different datasets.

4.3.2 Comparison with hand-crafted features

We also compare our method with the following hand-crafted features suggested specifically for the location prediction task by previous work:

- *Most Frequent POI (**MFP**)* is the best performing mobility feature suggested by [10]. For each user, it ranks a POI according to the number of the user's check-ins at that POI in the training dataset.
- *Most Frequent Time (**MFT**)* is suggested by [59]. For each user and each time slot, it ranks a POI according to the number of user's check-ins at that POI and at that time slot in the training dataset.
- *Social Filtering (**SF**)* is suggested by [10]. For a user u and his friends $\Gamma(u)$, it ranks a POI according to the total number of check-ins that any friend v ($v \in \Gamma(u)$) of the user has performed at the POI.
- *Weighted Social Filtering (**WSF**)* is suggested by [11]. It is a weighted version of SF, where each friend's check-in count is weighed by the mobility similarity between the user and that friend. It ranks a POI according to the weighted sum of the number of check-ins that any her friend v, $v \in \Gamma(u)$, has performed at that POI.

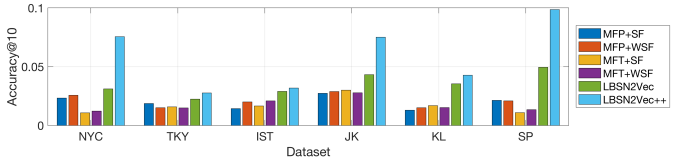


Fig. 9. Location prediction performance comparison with hand-craft features on new POIs.

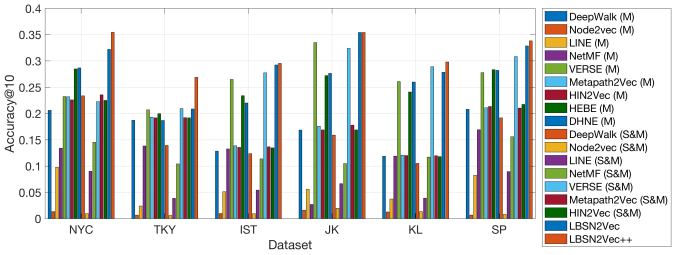


Fig. 10. Activity prediction performance comparison with other graph embedding methods.

Similar to the friendship prediction task, we combine the results from mobility and social features using reciprocal rank fusion [58], resulting in four combinations (i.e., **MFP+SF**, **MFP+WSF**, **MFT+SF**, **MFT+WSF**). Figure 9 shows the results on new POIs. We observe that our method consistently outperforms the hand-crafted feature. In addition, similar to the case of friendship prediction, none of the four combinations consistently achieve higher accuracy than others, showing the generalization limitations of hand-crafted features.

4.4 Activity Prediction

Activity prediction tries to predict activities a user is interested in at a given time slot [59]. Similar to the location prediction task, we learn node embeddings from the old social network and the training check-in data. Based on the learnt node embeddings, for each test check-in (u, c_{true}, t, l), we rank all activities $c \in C$ according to the cosine similarity between the transformed user node embeddings $\vec{u}^* = \mathrm{M}_t^U \vec{u}$ and the activity node embedding $\vec{c}$. We report the average accuracy@10 over all test check-ins. Figure 10 shows the results. We observe that our LBSN2Vec++ consistently outperforms all baselines, showing an average improvement of 7.63% and 12.85% on accuracy@10 over LBSN2Vec and other best-performing baselines, respectively. In addition, we also find that in general the performance on activity prediction is higher than locations prediction, as the number of activity categories is indeed much smaller than the number of POIs.

4.5 Balance the trade-off between social and mobility

We investigate the trade-off between the social network and mobility patterns on both the friendship and location prediction tasks by varying α within $[0, 1]$ with a step of 0.1. A small value of α gives more importance on the social network and less on check-in data when learning node embeddings, and vice versa.

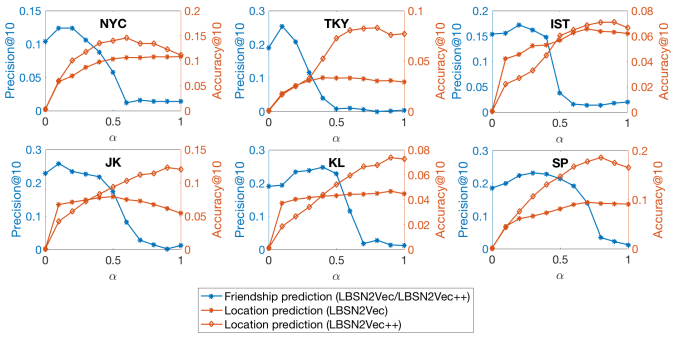


Fig. 11. Impact of α on friendship and location prediction tasks

Figure 11 shows the results for both LBSN2Vec and LBSN2Vec++. Note that we report the friendship prediction results for LBSN2Vec/LBSN2Vec++ together, since there is no visible difference between their performance. In general, we see a clear trade-off between the social network and mobility on all datasets. On one hand, when increasing α from 0, the friendship prediction performance slightly increases and reaches its peak around $\alpha = 0.2$, as considering mobility factors indeed helps the friendship prediction task. When further increasing α, the performance start to decrease, and we observe a sharp drop around $\alpha = 0.5$. On the other hand, when decreasing α from 1, the location prediction performance slightly increases and reaches its peak around $\alpha = 0.6$ for LBSN2Vec ($\alpha = 0.8$ for LBSN2Vec++) in most cases, meaning that considering social factors can also help the location prediction task. When further decreasing α, the performance start to decrease, but we observe a sharp drop around $\alpha = 0.2$ for LBSN2Vec ($\alpha = 0.4$ for LBSN2Vec++). In the extreme case where $\alpha = 0$, LBSN2Vec++ is equivalent to LBSN2Vec, showing the same location prediction performance, because both techniques learn only from homogeneous friendship edges (in the same way by optimizing the same objective function) while completely ignoring check-in hyperedges.

We also find the asymmetric impact of social and mobility factor on each other, which seems to be universal across different datasets for both LBSN2Vec and LBSN2Vec++. More precisely, combining 80% social with 20% mobility data results in the best performance for friendship prediction, while combining 60% mobility with 40% social data gives the best performance for location prediction using LBSN2Vec. For LBSN2Vec++, the best performance for location prediction is achieved with more mobility data (i.e., 80% mobility with 60% social data). This implies that LBSN2Vec++ considering the heterogeneous nature of the LBSN hypergraph is able to learn more than LBSN2Vec from check-in hyperedges, and thus models user mobility better.

4.6 Runtime Performance

We investigate the efficiency of both node embedding learning and link/location/activity prediction tasks using the learnt node embeddings. All the experiments are conducted on a commodity PC (Intel Core i7-6820HQ@2.70GHz, 16GB RAM, Mac OS X) on NYC dataset.

First, Table 2 shows the node embedding learning time. Compared to classical graph embedding techniques for homogeneous graphs (DeepWalk, Node2Vec, LINE, VERSE and NetMF), LBSN2Vec++ requires more learning time than these techniques in general, as it requires additional computation to capture heterogeneous hyper-relations in the LBSN hypergraph. We note that as the runtime complexity of LINE depends mostly on the number of sampled node pairs for learning (set to 1 billion), we observe a similar learning time over different settings. Compared to heterogeneous (hypergraph) embedding techniques (Metapath2Vec, Hin2Vec, HEBE and DHNE), LBSN2Vec++ is faster in embedding learning in general, as it is specifically designed to efficiently learn node embeddings from the LBSN hypergraph using embedding space transformation and n-wise node proximity preservation. Finally, compared to LBSN2Vec, incorporating embedding space transformations in LBSN2Vec++ indeed incurs a certain amount of learning time, but it significantly improves the performance on location prediction by 68%.

TABLE 2
Node embedding learning time (in seconds) with different dataset configuration settings if applicable. *We run DHNE without using GPUs, for a fair comparison with all other techniques (which use CPUs only).

Methods	(S)	(M)	(S&M)
DeepWalk	1313	2658	2660
Node2Vec	419	845	844
LINE	2199	2159	2234
VERSE	427	863	873
NetMF	191	584	984
Metapath2Vec	-	3269	3254
Hin2Vec	-	9708	9698
HEBE	-	3426	-
DHNE*	-	5623	-
LBSN2Vec	-	-	762
LBSN2Vec++	-	-	2254

Second, all evaluation tasks can be conducted very efficiently using the learnt node embeddings, as they mostly involve basic vector operations and sorting. Specifically, for 1000 predictions, the link, location and activity prediction tasks take about 2ms, 460ms and 370ms, respectively, for all embedding techniques.

5 Conclusion

In this paper, we introduce LBSN2Vec++, a heterogeneous hypergraph embedding approach designed specifically for LBSN data. It performs random-walk-with-stay to jointly sample user mobility patterns and social relationships from the LBSN heterogeneous hypergraph, and then learns node embeddings from the sampled hyperedges by not only preserving the n-wise node proximity captured by hyperedges, but also considering the embedding space transformation between node domains to fully grasp the complex structural characteristics of the LBSN heterogeneous hypergraph. Using real-world LBSN datasets collected in six cities all over the world, our extensive evaluation shows that LBSN2Vec++ significantly and consistently outperforms not only state-of-the-art graph embedding techniques by up to 68%, but also the best-performing hand-crafted features in the literature by up to 70.14%, on friendship and location prediction tasks. Moreover, using LBSN2Vec++, we discover the asymmetric impact of user mobility and social networks on predicting

each other, which can serve as guidelines for future research on friendship and location prediction in LBSNs.

In the future, we plan to explore the idea of graph embedding on analyzing other types of user mobility data such as transportation trajectories and fine-grained mobile phone traces, etc.

ACKNOWLEDGEMENTSS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 683253/GraphInt).

REFERENCES

[1] Y. Zheng, "Location-based social networks: Users," in *Computing with spatial trajectories*. Springer, 2011, pp. 243–276.

[2] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: user movement in location-based social networks," in *KDD*. ACM, 2011, pp. 1082–1090.

[3] S. Scellato, A. Noulas, and C. Mascolo, "Exploiting place features in link prediction on location-based social networks," in *KDD*. ACM, 2011, pp. 1046–1054.

[4] S. Scellato, C. Mascolo, M. Musolesi, and V. Latora, "Distance matters: Geo-social metrics for online social networks." in *WOSN*, 2010.

[5] S. Scellato, A. Noulas, R. Lambiotte, and C. Mascolo, "Socio-spatial properties of online location-based social networks." *ICWSM*, vol. 11, pp. 329–336, 2011.

[6] Y. Zhang and J. Pang, "Distance and friendship: A distance-based model for link prediction in social networks," in *APWeb*. Springer, 2015, pp. 55–66.

[7] A. Sadilek, H. Kautz, and J. P. Bigham, "Finding your friends and following them to where you are," in *WSDM*. ACM, 2012, pp. 723–732.

[8] J. Valverde-Rebaza, M. Roche, P. Poncelet, and A. de Andrade Lopes, "Exploiting social and mobility patterns for friendship prediction in location-based social networks," in *ICPR*. IEEE, 2016, pp. 2526–2531.

[9] R. Cheng, J. Pang, and Y. Zhang, "Inferring friendship from check-in data of location-based social networks," in *ASONAM*. ACM, 2015, pp. 1284–1291.

[10] A. Noulas, S. Scellato, N. Lathia, and C. Mascolo, "Mining user mobility features for next place prediction in location-based services," in *ICDM*. IEEE, 2012, pp. 1038–1043.

[11] H. Gao, J. Tang, and H. Liu, "Exploring social-historical ties on location-based social networks." in *ICWSM*, 2012.

[12] D. Lian, X. Xie, V. W. Zheng, N. J. Yuan, F. Zhang, and E. Chen, "Cepr: A collaborative exploration and periodically returning model for location prediction," *TIST*, vol. 6, no. 1, p. 8, 2015.

[13] L. A. Adamic and E. Adar, "Friends and neighbors on the web," *Social networks*, vol. 25, no. 3, pp. 211–230, 2003.

[14] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, no. 1, pp. 39–43, 1953.

[15] D. Wang, D. Pedreschi, C. Song, F. Giannotti, and A.-L. Barabasi, "Human mobility, social ties, and link prediction," in *KDD*. Acm, 2011, pp. 1100–1108.

[16] D. Yang, B. Qu, J. Yang, and P. Cudre-Mauroux, "Revisiting user mobility and social relationships in lbsns: A hypergraph embedding approach," in *WWW*. ACM, 2019, pp. 2147–2157.

[17] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[18] H. Cai, V. W. Zheng, and K. Chang, "A comprehensive survey of graph embedding: problems, techniques and applications," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[19] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *CIKM*. ACM, 2015, pp. 891–900.

[20] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *KDD*. ACM, 2016, pp. 1105–1114.

[21] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *WSDM*. ACM, 2018, pp. 459–467.

[22] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*. ACM, 2014, pp. 701–710.

[23] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW*. ACM, 2015, pp. 1067–1077.

[24] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*. ACM, 2016, pp. 855–864.

[25] A. Tsitsulin, D. Mottin, P. Karras, and E. Müller, "Verse: Versatile graph embeddings from similarity measures," in *WWW*. International World Wide Web Conferences Steering Committee, 2018, pp. 539–548.

[26] Y. Zhu, Z. Guan, S. Tan, H. Liu, D. Cai, and X. He, "Heterogeneous hypergraph embedding for document recommendation," *Neurocomputing*, vol. 216, pp. 150–162, 2016.

[27] A. Bahmanian and M. Newman, "Embedding factorizations for 3-uniform hypergraphs ii: r-factorizations into s-factorizations," *The Electronic Journal of Combinatorics*, vol. 23, no. 2, pp. 2–42, 2016.

[28] K. Tu, P. Cui, X. Wang, F. Wang, and W. Zhu, "Structural deep embedding for hyper-networks," in *AAAI*, 2017.

[29] H. Gui, J. Liu, F. Tao, M. Jiang, B. Norick, and J. Han, "Large-scale embedding learning in heterogeneous event data," in *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 2016, pp. 907–912.

[30] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *KDD*. ACM, 2017, pp. 135–144.

[31] T.-y. Fu, W.-C. Lee, and Z. Lei, "Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning," in *CIKM*. ACM, 2017, pp. 1797–1806.

[32] R. Hussein, D. Yang, and P. Cudré-Mauroux, "Are meta-paths necessary?: Revisiting heterogeneous graph embeddings," in *CIKM*. ACM, 2018, pp. 437–446.

[33] J. Ang, T. Fu, J. Paul, S. Zhang, B. He, T. S. D. Wenceslao, and S. Y. Tan, "Trav: An interactive exploration system for massive trajectory data," in *BigMM 2019*. IEEE, 2019, pp. 309–313.

[34] P. Wang, Y. Fu, G. Liu, W. Hu, and C. Aggarwal, "Human mobility synchronization and trip purpose detection with mixture of hawkes processes," in *KDD 2017*, 2017, pp. 495–503.

[35] D. Yang, D. Zhang, L. Chen, and B. Qu, "Nationtelescope: Monitoring and visualizing large-scale collective behavior in lbsns," *Journal of Network and Computer Applications*, vol. 55, pp. 170–180, 2015.

[36] D. Yang, D. Zhang, and B. Qu, "Participatory cultural mapping based on collective behavior data in location-based social networks," *TIST*, vol. 7, no. 3, p. 30, 2016.

[37] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *journal of the Association for Information Science and Technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[38] D. Yang, D. Zhang, Z. Yu, and Z. Wang, "A sentiment-enhanced personalized location recommendation system," in *HT*. ACM, 2013, pp. 119–128.

[39] W. Mathew, R. Raposo, and B. Martins, "Predicting future locations with hidden markov models," in *UbiComp*. ACM, 2012, pp. 911–918.

[40] T. Kurashima, T. Iwata, T. Hoshide, N. Takaya, and K. Fujimura, "Geo topic model: joint modeling of user's activity area and interests for location recommendation," in *WSDM*. ACM, 2013, pp. 375–384.

[41] H. Yin, Y. Sun, B. Cui, Z. Hu, and L. Chen, "Lcars: a location-content-aware recommender system," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 221–229.

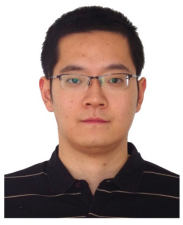
[42] Y. Wang, N. J. Yuan, D. Lian, L. Xu, X. Xie, E. Chen, and Y. Rui, "Regularity and conformity: Location prediction using heterogeneous mobility data," in *KDD*. ACM, 2015, pp. 1275–1284.

[43] J. Ye, Z. Zhu, and H. Cheng, "What's your next move: User activity prediction in location-based social networks," in *SDM*. SIAM, 2013, pp. 171–179.

[44] M. Brand, "Fast online svd revisions for lightweight recommender systems," in *SDM*. SIAM, 2003, pp. 37–46.

[45] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," *AAAI 2019*, 2018.

[46] I. M. Baytas, C. Xiao, F. Wang, A. K. Jain, and J. Zhou, "Heterogeneous hyper-network embedding," in *ICDM*. IEEE, 2018, pp. 875–880.
[47] X. Chu, X. Fan, D. Yao, C.-L. Zhang, J. Huang, and J. Bi, "Noise-aware network embedding for multiplex network," in *IJCNN 2019*. IEEE, 2019, pp. 1–8.
[48] Y. Wang, Z. Qin, J. Pang, Y. Zhang, and J. Xin, "Semantic annotation for places in lbsn through graph embedding," in *CIKM*. ACM, 2017, pp. 2343–2346.
[49] M. Xie, H. Yin, H. Wang, F. Xu, W. Chen, and S. Wang, "Learning graph-based poi embedding for location-based recommendation," in *CIKM*. ACM, 2016, pp. 15–24.
[50] R. Ding and Z. Chen, "Recnet: a deep neural network for personalized poi recommendation in location-based social networks," *International Journal of Geographical Information Science*, pp. 1–18, 2018.
[51] T. Qian, B. Liu, Q. V. H. Nguyen, and H. Yin, "Spatiotemporal representation learning for translation-based poi recommendation," *ACM Transactions on Information Systems (TOIS)*, vol. 37, no. 2, p. 18, 2019.
[52] S. Feng, G. Cong, B. An, and Y. M. Chee, "Poi2vec: Geographical latent representation for predicting future visitors." in *AAAI*, 2017, pp. 102–108.
[53] P. Wang, Y. Fu, H. Xiong, and X. Li, "Adversarial substructured representation learning for mobile user profiling," in *KDD 2019*, 2019, pp. 130–138.
[54] P. Wang, Y. Fu, J. Zhang, X. Li, and D. Lin, "Learning urban community structures: A collective embedding perspective with periodic spatial-temporal mobility graphs," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 9, no. 6, pp. 1–28, 2018.
[55] W. X. Zhao, F. Fan, J.-R. Wen, and E. Y. Chang, "Joint representation learning for location-based social networks with multi-grained sequential contexts," *ACM Transactions on Knowledge Discovery from Data*, vol. 12, no. 2, p. 22, 2018.
[56] J. Hopcroft and R. Kannan, *Foundations of data science*, 2014.
[57] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Physica A: statistical mechanics and its applications*, vol. 390, no. 6, pp. 1150–1170, 2011.
[58] G. V. Cormack, C. L. Clarke, and S. Buettcher, "Reciprocal rank fusion outperforms condorcet and individual rank learning methods," in *SIGIR*. ACM, 2009, pp. 758–759.
[59] D. Yang, D. Zhang, V. W. Zheng, and Z. Yu, "Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns," *TSMC*, vol. 45, no. 1, pp. 129–142, 2015.



Dingqi Yang is a senior researcher at the University of Fribourg in Switzerland. He received the Ph.D. degree in computer science from Pierre and Marie Curie University and Institut Mines-TELECOM/TELECOM SudParis, where he won both the CNRS SAMOVAR Doctorate Award and the Institut Mines-TELECOM Press Mention in 2015. His research interests include big social media data analytics, ubiquitous computing, and smart city applications.

Bingqing Qu is a researcher at the University of Fribourg in Switzerland. She received her Ph.D. in Computer Science in University of Rennes 1 in 2016. Her research interests include historical document analysis, multimedia content analysis, social media data mining and computer vision.

Jie Yang is an Assistant Professor at the Web Information Systems Group of the Faculty of Engineering, Mathematics and Computer Science (EEMCS/EWI), Delft University of Technology (TU Delft). Before joining TU Delft, he was a machine learning scientist at Amazon and a senior researcher at the eXascale Infolab, University of Fribourg. His research focuses on human-centered machine learning for Web-scale information systems, aiming at leveraging the joint power of human and machine intelligence for understanding and making use of data in large-scale information systems.

Philippe Cudre-Mauroux is a Full Professor and the director of the eXascale Infolab at the University of Fribourg in Switzerland. He received his Ph.D. from the Swiss Federal Institute of Technology EPFL, where he won both the Doctorate Award and the EPFL Press Mention. Before joining the University of Fribourg he worked on information management infrastructures for IBM Watson Research, Microsoft Research Asia, and MIT. His research interests are in next-generation, Big Data management infrastructures for non-relational data. Webpage: http://exascale.info/phil