SPEX: A Generic Framework for Enhancing Neural Social Recommendation

HUI LI, School of Informatics, Xiamen University, China
LIANYUN LI, School of Informatics, Xiamen University, China
GUIPENG XV, School of Informatics, Xiamen University, China
CHEN LIN*, School of Informatics, Xiamen University, China
KE LI, PLA Strategic Support Force Information Engineering University, China
BINGCHUAN JIANG, PLA Strategic Support Force Information Engineering University, China

Social Recommender Systems (SRS) have attracted considerable attention since its accompanying service, social networks, helps increase user satisfaction and provides auxiliary information to improve recommendations. However, most existing SRS focus on social influence and ignore another essential social phenomenon, i.e., social homophily. Social homophily, which is the premise of social influence, indicates that people tend to build social relations with similar people and form influence propagation paths. In this paper, we propose a generic framework Social PathExplorer (SPEX for short) to enhance neural SRS. SPEX treats the neural recommendation model as a black box and improves the quality of recommendations by modeling the social recommendation task, the formation of social homophily, and their mutual effect in the manner of multi-task learning. We design a Graph Neural Network based component for influence propagation path prediction to help SPEX capture the rich information conveyed by the formation of social homophily. We further propose an uncertainty based task balancing method to set appropriate task weights for the recommendation task and the path prediction task during the joint optimization. Extensive experiments have validated that SPEX can be easily plugged into various state-of-the-art neural recommendation models and help improve their performance. The source code of our work is available at: https://github.com/XMUDM/SPEX.

CCS Concepts: • **Information systems** → **Recommender systems**; **Social recommendation**; **Social networks**.

Additional Key Words and Phrases: social recommender, social homophily, social influence, multi-task learning

ACM Reference Format:
Hui Li, Lianyun Li, Guipeng Xv, Chen Lin, Ke Li, and Bingchuan Jiang. 2021. SPEX: A Generic Framework for Enhancing Neural Social Recommendation. *ACM Transactions on Information Systems* 1, 1 (July 2021), 32 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*Chen Lin is the corresponding author.

Authors' addresses: Hui Li, School of Informatics, Xiamen University, Xiamen, China, hui@xmu.edu.cn; Lianyun Li, School of Informatics, Xiamen University, Xiamen, China, lilianyun@stu.xmu.edu.cn; Guipeng Xv, School of Informatics, Xiamen University, Xiamen, China, xuguipeng@stu.xmu.edu.cn; Chen Lin, School of Informatics, Xiamen University, Xiamen, China, chenlin@xmu.edu.cn; Ke Li, PLA Strategic Support Force Information Engineering University, Zhengzhou, China, like19771223@163.com; Bingchuan Jiang, PLA Strategic Support Force Information Engineering University, Zhengzhou, China, jbc021@163.com.

1046-8188/2021/7-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Recommender systems (RS), which suggest desirable items to users, have become *de facto* tools for alleviating the information overload problem. Not only can users utilize RS to search for desirable targets efficiently, but also RS help e-commerce platforms promote their products and boost sales. Therefore, RS have been widely deployed in practice. Real applications include music recommendation (e.g., Yahoo! Music), video recommendation (e.g., Netflix), product recommendation (e.g., Amazon), to name a few.

In addition to user-item interactions (e.g., user-item ratings or user clicks), RS can harness auxiliary information to enrich the training data and provide better recommendations [1]. A promising direction is to leverage social relations since social networking is deployed as the accompanying service in many RS to increase user satisfaction. For example, Yelp[1] provides recommendations for local business. At the same time, Yelp users can follow each other and check other people's ratings and reviews. A great number of social recommender systems (SRS) are proposed in the literature, and they deliver superior performance than general RS without additional social features [68, 86]. Furthermore, they are shown to be effective to handle the cold-start problem (i.e., new user/item does not have historical interactions) and data sparsity problem (i.e., many existing users/items only have few interactions) [1].

Early SRS focus on exploiting social relations to enhance matrix factorization [45, 46] or nearest neighbor based recommendation approaches [86]. The recent blossom of deep learning techniques has dramatically changed the study of social recommendation [5, 10, 65]. Particularly, Graph Neural Network (GNN), which aims at graph based learning, has become prevalent for designing SRS [11, 12, 63, 73, 81, 85] since it is a natural fit for modeling information diffusion in the graph and it has shown promising results for the social recommendation task. One reason for the improvements of above SRS over traditional RS is that they model the *social influence* [47], i.e., the phenomenon that people who influence each other via influence propagation paths ultimately become more similar [47, 79].

Despite the success of SRS, we find that existing SRS fail to distinguish social influence from another crucial social phenomenon, namely *social homophily* [48], and identify the mutual effect between social homophily and social influence. As sociologists postulate, social homophily is the phenomenon that people tend to relate to other people with similar preferences and form influence propagation paths [48]. On one hand, as social influence diffuses along influence propagation paths, the formation of social homophily can be viewed as the premise of social influence. On the other hand, as their definitions reveal, social homophily and social influence are mutually reinforced. We now illustrate the mutual effect between social homophily and social influence in the context of SRS. As depicted in Figure 1, Alice is looking at the web page of the game *Grand Theft Auto V* on Epinions[2], an online recommender system. She finds that Swaminathan (user name: mailme_swami) appears in the section "Highest Rated Consumer Review by the Community" because Swaminathan recently gave a high rating on this game. After browsing Swaminathan's Epinions homepage[3], Alice feels that Swaminathan has similar tastes as her (they both like action games) and therefore starts to follow Swaminathan (i.e., establish the *trust* relation in Epinions). In this example, social homophily is formed and a new influence propagation path (i.e., the social relation $\langle$*Swaminathan, Alice*$\rangle$) appears due to the new user-item interaction $\langle$*Swaminathan, Grand Theft Auto V* $\rangle$. After

[1]https://www.yelp.com
[2]https://bit.ly/3aEED2p
[3]https://bit.ly/3xqhEll

Fig. 1. An overview of the mutual effect between social homophily and social influence in the context of social recommendation systems. Recommendation results may cause the formation of new social homophily, and social influence propagated via the new social homophily can be utilized to help the system provide a better recommendation.

the formation of social homophily, we can recommend *Tomb Raider*[4], which is highly rated by Swaminathan, to Alice due to the possible propagation of social influence.

Figure 1 reveals that, social recommendation models can be improved by distinguishing social homophily and social influence, and modeling the mutual effect between them. Completely relying on social influence while ignoring the formation of social homophily, as most SRS [11, 38, 64] do, will weaken the ability of social recommendation models. The reason is that essentially they assume that social network is *invariable*. But in fact, social network is *dynamic*, i.e., new social relations that users build with others bring the formation of influence propagation paths and the new social homophily, which in turn causes the diffusion of social influence along new influence propagation paths and finally affects the recommendations as we have seen in the example of Figure 1. Modeling and capturing the rich information conveyed by the formation of social homophily, will help SRS understand how social influence may propagate in the future.

Based on the above rationale, in this paper, we propose a generic framework, Social PathEXplorer (*SPEX* for short), to enhance neural social recommendation by modeling the recommendation process with social influence, the formation of social homophily, and their mutual effect in the manner of multi-task learning. SPEX is orthogonal to existing neural recommendation models that can be applied in the social recommendation task. The goal of this work is not designing a new recommendation method which can exceed state-of-the-art methods for the social recommendation task, but providing some insights into how to better leverage and model both social homophily and

[4]https://bit.ly/3tOVI1i

social influence to improve existing SRS. Thus, we treat neural recommendation models as a *black box* so that SPEX can be easily plugged into existing neural SRS or general neural RS when social data is available.

The contributions of this paper are summarized as follows:

- We point out that existing SRS have failed to distinguish social homophily and social influence as well as identify their mutual effect, which limits their performance of social recommendation.
- We revisit the concept of *temporal influence* in the literature of social network analysis [34] and design a Graph Neural Network [84] based module to model the core in the establishment of social homophily via predicting influence propagation paths.
- We propose a general multi-task learning framework called Social PathExplorer (SPEX) to model the recommendation process with social influence, the formation of social homophily and their mutual effect. In the joint optimization, SPEX is able to automatically balance the weights of different tasks using uncertainty.
- SPEX can be plugged into existing neural SRS or general RS, when social information is available, to enhance the recommendation without much effort. We conduct extensive experiments on several state-of-the-art neural SRS and general RS over public social recommendation data sets. The experimental results demonstrate that the performance of these recommendation models can be improved with SPEX.

The remainder of this paper is organized as follows: Section 2 introduces the background knowledge. Section 3 provides an overview of SPEX. Section 4 illustrates how SPEX models the formation of social homophily. Section 5 describes how SPEX can be plugged into existing social influence driven neural recommendation models in the manner of multi-task learning. We present our experiments in Section 6. Section 7 reviews the related work. Finally, Section 8 concludes our work.

2 PROBLEM DEFINITION

We first review the definition of the social recommendation task. Recommendation models typically rely on user-item historical interactions to provide recommendations. Assume that there are m users $\{u_1, \cdots u_m\}$ and n items $\{v_1, \cdots v_n\}$. The user-item interactions form an $m \times n$ interaction matrix $R = [r_{i,j}]$ where $r_{i,j} = 1$ if user u_i has interacted with item v_j (e.g., rate, click or view), otherwise $r_{i,j} = 0$. We additionally have a social network where each node represents a user and each edge represents the social relationships between users (e.g., friends or followers). The social network can also be modeled by an $m \times m$ social adjacency matrix $S = [s_{i,j}]$, where $s_{i,j} = 1$ indicates the existence of social relation between users u_i and u_j, otherwise $s_{i,j} = 0$. The task of a social recommender is as follows:

Definition 1 (Social Recommendation). *Given the user-item interaction matrix $R = [r_{i,j}]$ and the social adjacency matrix $S = [s_{i,j}]$, provide a list of k items with the highest probabilities that a target user will interact with in the future.*

Note that there is another social recommendation task aiming at predicting ratings of users on items (i.e., explicit feedback) in SRS. Since top-k recommendation is acknowledged to be more important in practice [1], we focus on the ranking task (i.e., implicit feedback) in this paper. However, the idea of SPEX can be generalized to the setting of explicit feedback.

3 OVERVIEW OF SPEX

Figure 2 provides an overview of SPEX using direct feature sharing mechanism (More details about feature sharing are included in Section 5.2). SPEX treats the neural recommendation approach as

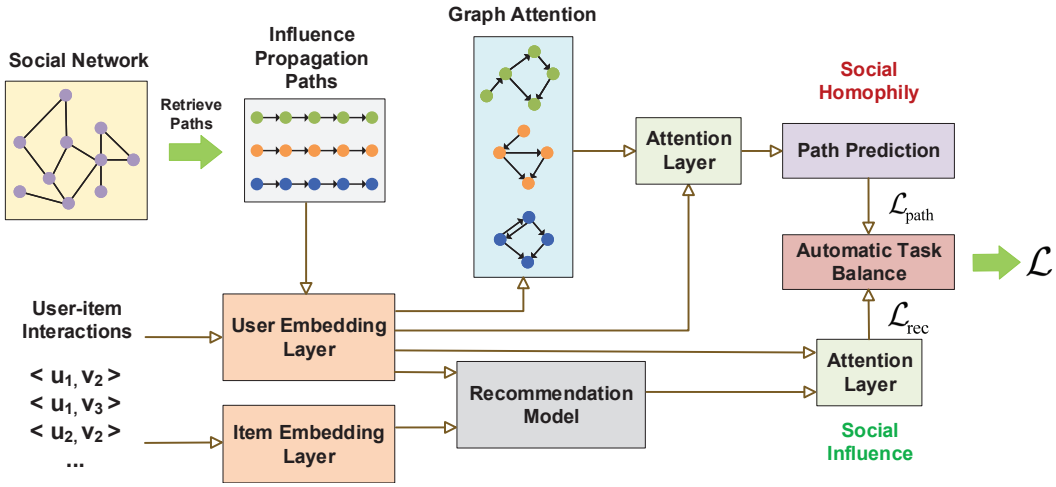

Fig. 2. An overview of SPEX with only direct feature sharing being illustrated.

a black box. SPEX uses an influence propagation path prediction module with a graph attention mechanism to model the core in the formation of social homophily (Section 4). The user embeddings are shared between the path prediction module and the neural recommendation model. With the modeling of influence propagation paths, SPEX is able to capture the rich information contained in the formation of social homophily and understand how users are or will become connected in the social network. Then, social influence, which is diffused along the influence propagation paths, can be better incorporated into the social recommendation process. The influence propagation path prediction task and the recommendation task are trained simultaneously with the balance of the two tasks being automatically achieved (Section 5).

4 MODELING THE FORMATION OF SOCIAL HOMOPHILY

The formation of social homophily is the premise of social influence, and the core of such a process is the establishment of influence propagation paths. As social influence propagates over the social network, it is not sufficient to model social homophily solely based on single-link influence propagation paths such as the relation $\langle Bob, Alice \rangle$. Instead, multi-hop paths such as $\langle Bob, Alice, Carol, David \rangle$ is desirable as they show the causal relationship:

(1) Carol can be affected by Bob since the social relation $\langle Bob, Alice \rangle$ was established earlier than $\langle Alice, Carol \rangle$.
(2) Bob may not be influenced by Carol as $\langle Bob, Alice, Carol, David \rangle$ shows a propagation path from Bob to Carol, not vice verse. Bob may not see Carol's historical actions when he is going to make some decisions.

The above inference, however, can not be drawn based on single-link influence propagation paths $\langle Bob, Alice \rangle$ and $\langle Alice, Carol \rangle$ only.

This section will first introduce the definition of influence propagation path and how the path is collected in SPEX. Then, we will explain how SPEX captures the core of the formation of social homophily via predicting influence propagation paths.

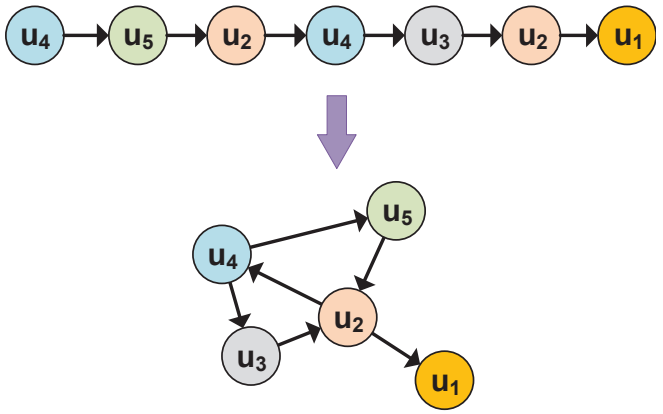

Fig. 3. The path graph of a 6-hop influence propagation path.

4.1 Influence Propagation Path

Firstly, we will review the concept of *temporal influence* [34] and then adopt it to provide the definition of influence propagation path which is the medium where the social influence propagates. Li et al. [34] point out that the time when two social actions occur needs to be considered in order to determine the causal relationship, and they propose the concept of temporal influence: only the action that occurred earlier may cause the action that took place later, not vice versa. The definition of influence propagation path is constrained by temporal influence:

Definition 2 (Influence Propagation Path). *A l-hop influence propagation path is a sequence of $l+1$ users $\{u_{i_1}, u_{i_2}, \cdots, u_{i_{l+1}}\}$ where (1) i_q indicates the q-th node ($1 \leq q \leq l+1$) and nodes at different positions can be the same (e.g., both u_{i_3} and u_{i_6} are u_2 in the 6-hop influence propagation path shown in Figure 3); (2) $s_{i_{q-1},i_q} = 1$, i.e., $(q-1)$-th node and q-th node are connected in the social network; (3) the social relation between $\langle u_{i_{q-2}}, u_{i_{q-1}} \rangle$ was created earlier than the one between $\langle u_{i_{q-1}}, u_{i_q} \rangle$. The social relation can be either undirected or directed. User u_{i_a} can be influenced by user u_{i_b} through the influence propagation path if $a > b$.*

Discussion: Difference Between Influence Propagation Path and Meta-path. There is another relevant concept called meta-path in Heterogeneous Information Network based RS [23]. However, it is necessary to define and use influence propagation path in SRS:

(1) Influence propagation path emphasizes the temporal influence while meta-path does not consider the factor of time. Thus, influence propagation path is more accurate for SRS. Only after the corresponding social relation has been created, one user can see and be influenced by his/her direct social neighbors' actions.
(2) All nodes in an influence propagation path are users, while nodes in a meta-path may have different types (e.g., location node, item node or item type node). Influence propagation path is available in the basic setting of social recommendation task where only user-item and user-user interactions are available. Meta-path may not be available in SRS as it requires more additional node information.

In the following, we will simply use "path" and "path prediction" to refer to "influence propagation path" and "influence propagation path prediction", respectively.

4.2 Generation of Paths in Social Recommenders

Starting from each user, we perform the path sampling for 50 times. And the sampling is done node by node for each path. Each path is independently sampled. Specifically, for each node u_i, we randomly sample the next node u_{i+1} in the path so that the path to be formed complies with Definition 2. That is, (1) the social relation between the current node u_i and the next node u_{i+1} exists, and (2) it is established after the social relation between u_i and u_{i-1}. Furthermore, as longer paths will introduce noisy semantics [66] and require more computation time, we set the maximal length of the path to be six. Thus, the sampling for generating each path terminates when (1) no possible social relation exists, or (2) the number of nodes in this path reaches six.

To fully utilize the data, we conduct *data augmentation* during training. For instance, for a path $u_1 \rightarrow u_2 \rightarrow u_3 \rightarrow u_4$ (i.e., the influence can be propagated from u_1 to u_4), SPEX is trained to predict all the nodes in this path except the first node, i.e., there are 3 predication tasks derived from this path: (1) Given $\{u_1\}$, predict u_2; (2) Given $\{u_1, u_2\}$, predict u_3; (2) Given $\{u_1, u_2, u_3\}$, predict u_4.

4.3 Path Prediction: User Representation

Next, we explore how to model the core of social homophily, i.e., the path prediction task. The reason for performing this task is that social homophily indicates the phenomenon that people tend to build relations with similar people, and modeling its formation is equivalent to predicting each link (i.e., social relation) in the influence propagation path:

Definition 3 (Influence Propagation Path Prediction). *A l-hop influence propagation path can be segmented into l relations: $\{\langle u_{i_1}, u_{i_2}\rangle, \langle u_{i_2}, u_{i_3}\rangle, \cdots, \langle u_{i_l}, u_{i_{l+1}}\rangle\}$, where u_{i_q} $(1 \leq q \leq l+1)$ is the q-th user in the path. The prediction problem for influence propagation path can then be defined as predicting the q-th user in the path, given its preceding $q-1$ users.*

The first step in path prediction is learning representations of users. Recently, Graph Neural Network (GNN) [84, 95] have been successfully deployed in several sequence based applications including sequential RS [83, 91]. For instance, Wu et al. [83] convert a sequence of user historical interactions into a graph and then apply gated GNN [36] for predicting next interaction in sequential RS. Inspired by these methods, we regard each path as a small path graph in social RS and adopt the idea of GNN to model the path graph and extract user representations. For instance, Figure 3 shows a 6-hop path from u_4 to u_1 and its corresponding path graph.

However, the positional information of each node in the path is missing in the path graph: duplicated nodes are merged into one node in the path graph. We record the distance between the last occurrence of each node and the right-most node in the path as the positional encoding. For instance, the position embeddings $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, $\mathbf{p}_4$ and $\mathbf{p}_5$, for u_1, u_2, u_3, u_4 and u_5 in the example of Figure 3 are vectors with all dimensions being 0, 1, 2, 3 and 5, respectively. And the initial d-dimensional representation $\mathbf{h}_i^{(t-1)} \in \mathbb{R}^d$ at epoch t for user u_i is:

$$\mathbf{h}_i^{(t-1)} = \mathbf{u}_i^{(t-1)} + \mathbf{p}_i, \tag{1}$$

where $\mathbf{u}_i^{(t-1)}$ is the user embedding for user u_i after epoch $t-1$ ($\mathbf{u}_i^{(0)}$ is the initial user embedding for u_i). Note that $\mathbf{p}_i$ is fixed and will not get updated during optimization.

Then, the d-dimensional representation $\mathbf{h}_i^{(t)} \in \mathbb{R}^d$ of user u_i after epoch t can be calculated as the aggregation of the representations of his/her direct neighbors and itself at epoch $t-1$:

$$\mathbf{h}_i^{(t)} = ELU\Big(\sum_{j \in \mathcal{N}_{i,p}} \mu_{i,j,p}^{(t)} \mathbf{W}_h \mathbf{h}_j^{(t-1)}\Big), \tag{2}$$

where $\mathcal{N}_{i,p}$ contains the direct neighbors of user u_i in the path p and i itself, $ELU(\cdot)$ is the Exponential Linear Unit, and $\mathbf{W}_h \in \mathbb{R}^{d\times d}$ is a learnable weight matrix. It is worth noting that the aggregation is path-specific, which means $\mathcal{N}_{i,p}$ does not contain directed neighbors of user i which are not in the path p. $\mathcal{N}_{i,p}$ also contains node u_i to avoid that the information of node i disappears during the iterative aggregation process. $\mathbf{h}_i^{(t)}$ will be updated via Equation 2 whenever SPEX is fed with a path p containing u_i at epoch t. $\mu_{i,j,p}^{(t)}$ in Equation 2 is the attention coefficient of node j to i in path p for epoch t. It emphasizes different degree of impacts that node i receives from neighbors in p, and it can be obtained through the following graph attention layer:

$$\mu_{i,j,p}^{(t)} = \frac{exp\Big(LeakyReLU\big(\mathbf{w}_\mu^T \left[\mathbf{h}_i^{(t-1)} \,\|\, \mathbf{h}_j^{(t-1)}\right]\big)\Big)}{\sum_{k\in\mathcal{N}_{i,p}} exp\Big(LeakyReLU\big(\mathbf{w}_\mu^T \left[\mathbf{h}_i^{(t-1)} \,\|\, \mathbf{h}_k^{(t-1)}\right]\big)\Big)}, \tag{3}$$

where "$\|$" is the vertical concatenation operation, $LeakyReLU(\cdot)$ indicates the leaky version of the Rectified Linear Unit, and $\mathbf{w}_\mu \in \mathbb{R}^{2d}$ is a learnable weight vector.

As pointed out by Velickovic et al. [71], the above learning process may be unstable and they adopt the multi-head attention mechanism which is similar to that used in the Transformer [70]. Following their idea, we apply g independent attention-based aggregations as shown in Equation 3 (i.e., g-head attention) and the results are concatenated to form the node representation:

$$\mathbf{h}_i^{(t)} = \Big\|_{q=1}^{g} ELU\Big(\sum_{j\in\mathcal{N}_{i,p}} \big(\mu_{i,j,p}^{(t)}\big)_q \big(\mathbf{W}_h\big)_q \mathbf{h}_j^{(t-1)}\Big), \tag{4}$$

where $\big(\cdot\big)_q$ indicates the q-th attention coefficient or learnable weight matrix, and "$\|$" is the vertical concatenation operation.

Finally, $\mathbf{h}_i^{(t)}$ is passed through another graph attention layer without multiple heads to generate the user representation $\hat{\mathbf{h}}_i^{(t)}$ for user i after epoch t:

$$\begin{aligned} v_{i,j,p}^{(t)} &= \frac{exp\Big(LeakyReLU\big(\mathbf{w}_v^T \left[\mathbf{h}_i^{(t)} \,\|\, \mathbf{h}_j^{(t)}\right]\big)\Big)}{\sum_{k\in\mathcal{N}_{i,p}} exp\Big(LeakyReLU\big(\mathbf{w}_v^T \left[\mathbf{h}_i^{(t)} \,\|\, \mathbf{h}_k^{(t)}\right]\big)\Big)} \\ \hat{\mathbf{h}}_i^{(t)} &= ELU\Big(\sum_{j\in\mathcal{N}_{i,p}} v_{i,j,p}^{(t)} \mathbf{W}_{\hat{h}} \mathbf{h}_j^{(t)}\Big) \end{aligned} \tag{5}$$

where $\mathbf{w}_v \in \mathbb{R}^{2gd}$ and $\mathbf{W}_{\hat{\mathbf{h}}} \in \mathbb{R}^{d\times gd}$ are a weight vector and a weight matrix, respectively.

Discussion 1: The Way to Model Paths. The problem of modeling the path can be converted to a sequence modeling problem, where sequence models like Markov Chains [58], deep neural networks like Convolutional Neural Network (CNN) [69], Recurrent Neural Network (RNN) [22], and attention based methods like Transformer [27] can be adopted [13, 56, 75]. However, GNN is superior in this task for three reasons:

(1) Firstly, GNN is able to handle loops in paths, e.g., the friends of your friends may become your direct friends in the future [79].
(2) Secondly, the topological structure among users is important, which can be captured by a path graph. GNN is able to handle the irregularity in the graph, i.e., each node may have a different number of neighbors, where some critical operations (e.g., convolution) in CNN and RNN are difficult [84, 95].

(3) Lastly, merging repeated nodes of a path into one node in the path graph and then applying GNN makes it easier to capture repeated patterns, which is difficult to model in the original sequence [78]. For example, we can observe from Figure 3 that u_2 and u_4 repeatedly appear in the sequence as they have more neighbors than other nodes in the path graph. Such repetitions can be captured by GNN and help model user representations better.

Another possible direction is to collect paths ending at the target user, combine them to construct a subgraph, and adopt GNN to model the subgraph. We do not adopt this alternative since such a method is coarse-grained: it may mix the influence sources. Moreover, modeling a subgraph is computationally intensive compared to processing paths individually.

Discussion 2: Connection with GNN Based Sequential RS. Representation learning in the path prediction module of SPEX is inspired by recent works on GNN based sequential RS [83, 91]. Modeling paths in social network and modeling historical interaction sequences in sequential RS are two similar tasks: both of them regard sequential data as graphs and then adopt the idea of GNN to model node representations. However, there also exist some differences between SPEX and GNN based sequential RS:

(1) Paths modeled by SPEX consist of users, and paths are originally part of the social network. No "transformation" between paths and path graphs actually occurs, meaning that there will be no information loss. Considering the example in Figure 3, the path graph at the bottom is the actual data, even though we say SPEX models the path (in the upper part of Figure 3) as a graph (at the bottom of Figure 3). Differently, interaction sequences in sequential RS consist of items. Such item sequences are not part of a larger graph. Therefore, there may exist information loss when converting sequences to graphs.
(2) Since path graphs are actually connected in the large social network, cross-path information can be naturally captured through information propagation in the social network. As a comparison, the cross-sequence information in sequential RS cannot be directly captured without special designs [54].
(3) Unlike the pioneering GNN based sequential RS [83] which treats items of a sequence equally, SPEX models the different impacts of each neighbor user in the path graph. The key idea is to leverage a recursive influence diffusion mechanism relying on the graph attention layer to obtain the latent vector of each node (i.e., user representation), which naturally models the influence propagation in the social network.
(4) Compared to those GNN based sequential RS [55] that also distinguish different impacts of nodes in the sequence, SPEX further takes the position of node in a path into account via the position embedding.

4.4 Path Prediction: Prediction Layer and Loss Function

The representation of the last node $u_{i_{l+1}}$ in a path $\{u_{i_1}, u_{i_2}, \cdots, u_{i_{l+1}}\}$ receives the information (directly or indirectly) from all its preceding nodes in the path. Hence, we use the last node $u_{i_{l+1}}$ as an "anchor" for other nodes in the path to compare with so that the construction of the path representation $\mathbf{e}$ can consider the importance of each node differently. More specifically, the path representation $\mathbf{e}^{(t)}_{p_{(l)}}$ of a l-hop path p after epoch t can be obtained via an attention layer defined as follows:

$$\begin{aligned} \gamma^{(t)}_{i_j} &= \mathbf{z}^T ELU\big(\mathbf{W}_1 \hat{\mathbf{h}}^{(t)}_{i_{l+1}} + \mathbf{W}_2 \hat{\mathbf{h}}^{(t)}_{i_j} + \mathbf{s}\big) \\ \mathbf{g}^{(t)}_{p(l)} &= \sum_{j=1}^{l+1} \gamma^{(t)}_{i_j} \hat{\mathbf{h}}^{(t)}_{i_j}, \quad \mathbf{e}^{(t)}_{p(l)} = \mathbf{W}_3\Big(\big[\mathbf{g}^{(t)}_{p(l)} \,\big\|\, \hat{\mathbf{h}}^{(t)}_{i_{l+1}}\big]\Big) \end{aligned} \tag{6}$$

where $\mathbf{z}, \mathbf{s} \in \mathbb{R}^d$ are learnable weight vector and bias vector, respectively. $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{d \times d}$ and $\mathbf{W}_3 \in \mathbb{R}^{d \times 2d}$ are learnable weight matrices.

Before making predictions and computing the loss value at epoch t, we further adopt a two-layer attention mechanism as defined in Equation 7. The motivation is to obtain a more robust final output path representation $\tilde{\mathbf{v}}_{p_{(l)}}^{(t)}$ for a l-hop path p at epoch t, by fusing the initial path representation $\mathbf{v}_{p_{(l)}}^{(t-1)}$ and the learned path representation $\mathbf{e}_{p_{(l)}}^{(t)}$ at epoch t. As the message passing in GNNs repeatedly aggregates excessive noise, making the training difficulty and unstable, the combination of both input representations and learned representations can alleviate this problem.

$$\begin{aligned} \mathbf{v}_{p_{(l)}}^{(t-1)} &= AveragePooling\big(\mathbf{u}_{i_1}^{(t-1)}, \cdots, \mathbf{u}_{i_{l+1}}^{(t-1)}\big), \;\; u_{i_1}, \cdots, u_{i_{l+1}} \in p \\ \mathbf{k}_{p_{(l)}}^{(t)} &= Softmax\big(\mathbf{W}^{(v)} \mathbf{v}_{p_{(l)}}^{(t-1)} + \mathbf{W}^{(e)} \mathbf{e}_{p_{(l)}}^{(t)}\big) \\ \tilde{\mathbf{v}}_{p_{(l)}}^{(t)} &= \big[\mathbf{v}_{p_{(l)}}^{(t)} \,\big\|_{(h)}\, \mathbf{e}_{p_{(l)}}^{(t)}\big] \cdot \mathbf{k}_{p_{(l)}}^{(t)} \end{aligned} \tag{7}$$

where the initial path representation $\mathbf{v}_{p_{(l)}}^{(t-1)}$ of a l-hop path p is the mean embedding of all the input embeddings of nodes in path p. $\mathbf{W}^{(v)}, \mathbf{W}^{(e)} \in \mathbb{R}^{2 \times d}$ are learnable parameter matrices. "$\|_{(h)}$" indicates the horizontal concatenation operation.

Given the path representation $\tilde{\mathbf{v}}_{p_{(q-1)}}^{(t)}$ of the first $q-1$ nodes in path p which can be calculated using Equation 7 (i.e., $l = q-1$), we use the product of $\tilde{\mathbf{v}}_{p_{(q-1)}}^{(t)}$ and the user embedding matrix $\mathbf{U}^{(t-1)} \in \mathbb{R}^{d \times m}$ (after updated at epoch $t-1$) to predict the probability distribution for the q-th node in Definition 3:

$$\hat{\mathbf{y}}_{p_q}^{(t)} = Softmax\Big(\big(\tilde{\mathbf{v}}_{p_{(q-1)}}^{(t)}\big)^T \mathbf{U}^{(t-1)}\Big), \tag{8}$$

where $\mathbf{y}_{p_q}^{(t)} \in \mathbb{R}^m$ contains the predicted probability of each user being the q-th node in the path p after epoch t, and m is the number of users.

We adopt the cross-entropy loss for the path prediction task:

$$\mathcal{L}_{path}^{(t)} = -\sum_{p \in P} \sum_{q=2}^{|p|+1} \Big(\mathbf{y}_{p_q} \log(\hat{\mathbf{y}}_{p_q}^{(t)})\Big), \tag{9}$$

where P is the path set, $\mathbf{y}_{p_q}$ indicates the one-hot encoding vector of the ground-truth user at q-th node in the path p, and $|p|$ denotes the number of hops in path p. Stochastic gradient descent based methods can be used for the optimization of Equation 9 and we adopt Adam [30] in SPEX. Note that, in this paper, we will simply use $\mathcal{L}_{path}$ to represent $\mathcal{L}_{path}^{(t)}$ if there is no ambiguity.

5 MULTI-TASK LEARNING

SPEX is a multi-task learning framework, which consists of two tasks: (1) the task of modeling the formation of social homophily via path prediction, and (2) the social recommendation task as defined in Section 2. The two tasks are linked by paths in the social network. We will denote the two tasks as $path$ and rec, respectively. In the sequel, we will first illustrate the target RS of SPEX. Then, we will explain the designs of feature sharing and task balancing in SPEX which are two essential components in multi-task learning [60].

5.1 Ranking and Embedding Based RS

Our goal is to make SPEX orthogonal to existing neural SRS or general neural RS applicable to the social recommendation task so that it can be easily plugged into existing systems. Thus, the prior knowledge of SPEX only includes: (1) the RS model needs iterative training and the user

embeddings are updated at each epoch, and (2) the RS model adopts binary cross entropy loss and negative sampling in its optimization.

The first prior knowledge is common for existing embedding based RS models including matrix factorization based methods [33] and neural RS models [92]. SPEX's feature sharing mechanism (we will illustrate it in Section 5.2), which is directly or indirectly built on embeddings, relies on the first prior knowledge. Furthermore, we plug a two-layer network at the end of the original RS model to stabilize the iterative training. This two-layer network is similar to what we design in the path prediction layer (Equation 7):

$$\begin{aligned} \mathbf{q}_i^{(t)} &= Softmax\big(\mathbf{W}^{(u)}\mathbf{u}_i^{(t-1)} + \mathbf{W}^{(f)}\mathbf{f}_i^{(t)}\big) \\ \tilde{\mathbf{f}}_i^{(t)} &= \big[\mathbf{u}_i^{(t-1)} \,\big\|_{(h)}\, \mathbf{f}_i^{(t)}\big] \cdot \mathbf{q}_i^{(t)} \end{aligned} \tag{10}$$

where $\mathbf{u}_i^{(t-1)}$ is the input user embedding for user u_i at epoch t ($\mathbf{u}_i^{(0)}$ is the initial embedding for user u_i before training), $\mathbf{f}_i^{(t)}$ is the user representation of user u_i output by the RS model after epoch t, and $\mathbf{W}^{(u)}, \mathbf{W}^{(f)} \in \mathbb{R}^{2\times d}$ are learnable parameter matrices.

Prevalent RS models typically use inner products of $\mathbf{f}_i^{(t)}$ and each item embeddings in the item embedding matrix, or a mapping network (e.g., multi-layer perceptron) using $\mathbf{f}_i^{(t)}$ as the input to generate the probability distribution for candidate items and make recommendations [59]. SPEX uses the output $\tilde{\mathbf{f}}_i^{(t)}$ from the above two-layer network instead of the original output $\mathbf{f}_i^{(t)}$ from the RS model for making recommendations. The motivation behind the above mechanism is similar to our design for the path prediction layer in Section 4.4: mitigate the difficulty of training and stabilize the performance without heavy changes of the original RS model.

The second prior knowledge is also a common practice in training recommendation models [21, 92]: train the model to rank positive user-item interactions (i.e., existing user-item interactions) higher than randomly sampled negative user-item interactions (i.e., non-existing user-item interactions). A wide range of shallow and deep neural RS that can be used for social recommendation are applicable. The objective used for such models can be defined as:

$$\mathcal{L}_{rec} = -\sum_{\langle i,j\rangle \in R \cup R^-} \Big(r_{i,j} \log \hat{r}_{i,j} + (1 - r_{i,j}) \log (1 - \hat{r}_{i,j}) \Big), \tag{11}$$

where R indicates the user-item interaction set. In each epoch, the recommendation model randomly samples n_{rec} items for each user u_i that he/she has not interacted with and constructs a negative sample set $n_{\text{rec}}(i)$ for user u_i. All the negative sample set form R^- for this epoch. $r_{i,j}$ is defined in Section 2 and $\hat{r}_{i,j}$ is the prediction of $r_{i,j}$ from the neural recommendation model of which the last layer commonly has the sigmoid activation function to bound the value.

It is worthy noting that SPEX can be extended to fit other common recommendation objectives like Personalized Ranking loss (BPR) [57] which also trains the model to rank the positive sample higher than negative samples, and all-rank loss (i.e., cross entropy loss) which trains the model to rank the ground-truth item higher than all possible item candidates.

5.2 Feature Sharing

The design of a multi-task learning architecture typically needs to consider how the feature can be shared among different tasks. In this subsection, we describe three different feature sharing methods, which are commonly used in the design of multi-task learning [60]. SPEX is flexible to accommodate different feature sharing designs in addition to these three methods. Note that we do not design the feature sharing between neural layers inside the RS model and the path prediction module. SPEX treats neural RS as a black box and does not assume that the prior knowledge of

the neural architecture of the recommendation model is available. Designing inter-layer sharing requires special considerations about the detailed neural architecture of RS.

The three different feature sharing methods are:

(1) **Direct Sharing.** In SPEX, both tasks need to encode users into corresponding user embeddings before further processing. As shown in Figure 2, our framework can adopt a simple yet effective feature sharing method: two tasks share features by using the same user embedding layer.

(2) **Cross-Stitch Sharing.** SPEX can also adopt the idea of the Cross-Stitch unit [50] and emphasize the correlation between different features during sharing. In this method, two tasks have their own user embedding layer. And the user embeddings $\mathbf{u}_i^{(rec)}$ (encoded by user embedding layer in the RS model) and $\mathbf{u}_i^{(path)}$ (encoded by user embedding layer in the path prediction module) for the same user u_i are multiplied by a shared weight matrix to obtain the input user embeddings $\hat{\mathbf{u}}_i^{(rec)}$ and $\hat{\mathbf{u}}_i^{(path)}$ which will be fed into the subsequent learning modules for the two tasks:

$$\begin{bmatrix} \hat{\mathbf{u}}_i^{(rec)} \\ \hat{\mathbf{u}}_i^{(path)} \end{bmatrix} = \begin{bmatrix} \chi_{(rec,rec)} & \chi_{(path,rec)} \\ \chi_{(path,path)} & \chi_{(rec,path)} \end{bmatrix} \begin{bmatrix} \mathbf{u}_i^{(rec)} \\ \mathbf{u}_i^{(path)} \end{bmatrix} \tag{12}$$

where the learnable parameters $\chi_{(*,*)}$ indicates the correlation coefficient of one task to the other task or the self-correlation coefficient.

(3) **Shared-Private Sharing.** Another more sophisticated design is a Shared-Private feature sharing architecture. This architecture emphasizes that features contain a task-specific part which should be private and a cross-task part which should be shared. In this method, two task-specific user embedding layers are used. The user embeddings $\mathbf{u}_i^{(rec)}$ and $\mathbf{u}_i^{(path)}$ for the same user u_i encoded by embedding layers in the RS model and the path prediction module are passed through a shared single-layer feedforward neural network and a private task-specific single-layer feedforward neural network. The results are concatenated and further processed to obtain the input user embeddings $\hat{\mathbf{u}}_i^{(rec)}$ and $\hat{\mathbf{u}}_i^{(path)}$ which will be fed into the subsequent learning modules for the two tasks:

$$\begin{aligned}
\acute{\mathbf{u}}_i^{(rec)} &= ReLU\left(\mathbf{W}_{shared}\mathbf{u}_i^{(rec)} + \mathbf{b}_{shared}\right) \\
\grave{\mathbf{u}}_i^{(rec)} &= ReLU\left(\mathbf{W}_{rec}^{(1)}\mathbf{u}_i^{(rec)} + \mathbf{b}_{rec}^{(1)}\right) \\
\hat{\mathbf{u}}_i^{(rec)} &= ReLU\left(\mathbf{W}_{rec}^{(2)}\left[\acute{\mathbf{u}}_i^{(rec)} \,\|\, \grave{\mathbf{u}}_i^{(rec)}\right] + \mathbf{b}_{rec}^{(2)}\right) \\
\acute{\mathbf{u}}_i^{(path)} &= ReLU\left(\mathbf{W}_{shared}\mathbf{u}_i^{(path)} + \mathbf{b}_{shared}\right) \\
\grave{\mathbf{u}}_i^{(path)} &= ReLU\left(\mathbf{W}_{path}^{(1)}\mathbf{u}_i^{(path)} + \mathbf{b}_{path}^{(1)}\right) \\
\hat{\mathbf{u}}_i^{(path)} &= ReLU\left(\mathbf{W}_{path}^{(2)}\left[\acute{\mathbf{u}}_i^{(path)} \,\|\, \grave{\mathbf{u}}_i^{(path)}\right] + \mathbf{b}_{path}^{(2)}\right)
\end{aligned} \tag{13}$$

where $ReLU(\cdot)$ indicates the Rectified Linear Unit. $\acute{\mathbf{u}}_i^{(rec)}$ and $\acute{\mathbf{u}}_i^{(path)}$ denote shared features, while $\grave{\mathbf{u}}_i^{(rec)}$ and $\grave{\mathbf{u}}_i^{(path)}$ indicate private features. $\mathbf{W}$ and $\mathbf{b}$ are learnable weight matrix and bias vector, respectively.

5.3 Automatic Task Balancing

The output of SPEX comprises a multi-class classification task trained with a cross entropy loss $\mathcal{L}_{path}$ in Equation 9, and a binary classification task trained with a binary cross entropy loss $\mathcal{L}_{rec}$ in Equation 11. In multi-task learning, the loss objectives are normally combined with different

task weights (i.e., w_{path}, w_{rec}) [60]:

$$\mathcal{L} = w_{path}\mathcal{L}_{path} + w_{\text{rec}}\mathcal{L}_{\text{rec}}. \tag{14}$$

However, manually setting task weights w_{path} and w_{rec} at the beginning is a suboptimal solution, as the two tasks may have different and changing converge speeds during training. On the other hand, searching and dynamically updating task weights during optimization is a difficult and expensive process. Task imbalances will impede proper training because they manifest as imbalances between backpropagated gradients. To overcome these issues, we need to design an automatic task balancing mechanism for SPEX.

5.3.1 Uncertainty based Multi-task Weighing. Engineering problems are typically solved within the confines of a model universe containing physical and probabilistic models (or sub-models) [32]. The model universe may have inherently *uncertain* quantities; furthermore, the (alternative) sub-models are invariably imperfect giving rise to additional uncertainties. Modeling these uncertainties is an important part of building the model universe, enhancing its reliability and reducing the risk [28, 32]. In view of this, we design the automatic task balancing mechanism for SPEX based on modeling uncertainty in multi-task learning.

Uncertainty can be captured with Bayesian modeling approaches [14, 28]. In Bayesian modeling, *Epistemic* uncertainty and *Aleatoric* uncertainty are two typical uncertainties. Epistemic uncertainty accounts for uncertainty in the model parameters which captures what the model does not know due to lack of collected data. Aleatoric uncertainty captures noise inherent in the observations. Aleatoric uncertainty can be further divided into *heteroscedastic* (data-dependent) uncertainty, which depends on the input data and is predicted as a model output, and *homoscedastic* (task-dependent) uncertainty, which stays constant for different data inputs. In multi-task learning, task-dependent uncertainty indeed captures the relative confidence between tasks, reflecting the uncertainty inherent to each task [29]. Therefore, we can weigh task losses via capturing homoscedastic uncertainty[5] in a multi-task learning setting.

Recently, Kendall et al. [29] has proposed an uncertainty based method for automatic setting task weights in multi-task learning which consists of any numbers of regression task(s) and/or multi-class classification task(s). However, *their method does not fit the binary classification trained with the negative sampling strategy and can not be used in the social recommendation task.* The design of the automatic task balancing mechanism in SPEX is built on the top of their idea, but we provide significant improvements so that it can be used in the social recommendation task.

Firstly, using Bayesian modeling, we derive the log likelihood for the outputs of the path prediction part in Equation 14 based on maximizing the Gaussian likelihood with homoscedastic uncertainty. As Kendall et al. [29], we squash a *scaled* output of path prediction in SPEX:

$$Pr_{path}\big(y_t\big|\mathbf{p}(x_{1:t-1}), \alpha\big) = Softmax\Big(y_t = c, \frac{1}{\alpha^2}\mathbf{p}(x_{1:t-1})\Big), \tag{15}$$

where α is a positive scalar and c is the id of the ground-truth user of the t-th node. $\mathbf{p}(x_{1:t-1})$ indicates the output of the path prediction module given the input $x_{1:t-1}$ that includes the first $t-1$ nodes in the path x, i.e., $\mathbf{p}(x_{1:t-1})$ is the probability distribution of all m user candidates for the t-th node in the path x. Equation 15 can be interpreted as a Boltzmann distribution (also called Gibbs distribution) [29] and α is often referred to as *temperature*. The magnitude of α indicates how "uniform" the discrete distribution is, which is related to its uncertainty as measured in entropy [29].

[5]Note that the type of uncertainty is only specified in this paragraph. In other parts of this paper, the term *uncertainty* simply refers to *homoscedastic uncertainty*.

Taking the negative logarithm of the likelihood (i.e., $\mathcal{L}_{path}$ in Equation 9) of t-th node being user with id c, we have:

$$\begin{aligned}\mathcal{L}_{path}(y_t = c, x_{1:t-1}) &= -\log Softmax\big(y_t = c, \mathbf{p}(x_{1:t-1})\big) \\ &= -\mathbf{p}_c(x_{1:t-1}) + \log \sum_{i=1}^{m} \exp\big(\mathbf{p}_i(x_{1:t-1})\big),\end{aligned} \tag{16}$$

where $\mathbf{p}_i(x_{1:t-1})$ indicates the i-th dimension of the probability distribution vector $\mathbf{p}(x_{1:t-1})$. Then, the log likelihood for the output being class c in Equation 15 can be derived as follows:

$$\begin{aligned}\log Pr\big(y_t = c \big| \mathbf{p}(x_{1:t-1}), \alpha\big) =& \frac{1}{\alpha^2}\mathbf{p}_c(x_{1:t-1}) - \log \sum_{i=1}^{m} \exp\Big(\frac{1}{\alpha^2}\mathbf{p}_i(x_{1:t-1})\Big) \\ =& -\frac{1}{\alpha^2}\mathcal{L}_{path}(y_t = c, x_{1:t-1}) + \log\Big(\sum_{i=1}^{m} \exp\big(\mathbf{p}_i(x_{1:t-1})\big)\Big)^{\frac{1}{\alpha^2}} \\ & - \log \sum_{i=1}^{m} \exp\big(\frac{1}{\alpha^2}\mathbf{p}_i(x_{i:t-1})\big).\end{aligned} \tag{17}$$

Using the following approximation proposed by Kendall et al. [29] which becomes an equality when $\alpha \rightarrow 1$:

$$\frac{1}{\alpha}\sum_{i=1}^{m} \exp\Big(\frac{1}{\alpha^2}\mathbf{p}_i(x_{1:t-1})\Big) \approx \Big(\sum_{i=1}^{m}\Big(\exp\big(\mathbf{p}_i(x_{1:t-1})\big)\Big)\Big)^{\frac{1}{\alpha^2}}, \tag{18}$$

we have:

$$\begin{aligned}&\log Pr\big(y_t = c \big| \mathbf{p}(x_{1:t-1}), \alpha\big) \\ =& -\frac{1}{\alpha^2}\mathcal{L}_{path}(y_t = c, x_{1:t-1}) - \log \frac{\sum_{i=1}^{m} \exp\big(\frac{1}{\alpha^2}\mathbf{p}_i(x_{i:t-1})\big)}{\Big(\sum_{i=1}^{m} \exp\big(\mathbf{p}_i(x_{1:t-1})\big)\Big)^{\frac{1}{\alpha^2}}} \\ \approx& -\frac{1}{\alpha^2}\mathcal{L}_{path}(y_t = c, x_{1:t-1}) - \log \alpha.\end{aligned} \tag{19}$$

Unlike the above log likelihood for the path prediction outputs in Equation 14, the log likelihood of the recommendation outputs in Equation 14 is usually passed through the sigmoid activation function $\sigma(\cdot)$ to generate the probability. *The approximation (Equation 18) proposed by Kendall et al. [29] is designed for the cross entropy loss, but it cannot be used for the negative sampling and the binary cross entropy loss.*

5.3.2 Novel Approximations for Recommendation. To incorporate the binary classification task with negative sampling into the uncertainty based method so that task weights can be automatically set in SPEX, we propose another approximation:

$$\frac{1}{\lambda^2}\Big(\exp\big(\frac{x}{\lambda^2}\big) + 1\Big) \approx \big(\exp(x) + 1\big)^{\frac{1}{\lambda^2}}, \tag{20}$$

which becomes an equality when the scalar $\lambda \rightarrow 1$. Based on Equation 20, we can derive the following approximations for the sigmoid function $\sigma(\cdot)$ which is the last layer of deep neural

networks for binary classification tasks:

$$\begin{aligned}\sigma(\frac{x}{\lambda^2}) &= \frac{\exp(\frac{x}{\lambda^2})}{\exp(\frac{x}{\lambda^2})+1} \approx \frac{1}{\lambda^2}\big(\frac{\exp(x)}{\exp(x)+1}\big)^{\frac{1}{\lambda^2}} = \frac{1}{\lambda^2}\big(\sigma(x)\big)^{\frac{1}{\lambda^2}} \\ 1-\sigma(\frac{x}{\lambda^2}) &= \frac{1}{\exp(\frac{x}{\lambda^2})+1} \approx \frac{1}{\lambda^2}\Big(\frac{1}{\exp(x)+1}\Big)^{\frac{1}{\lambda^2}} = \frac{1}{\lambda^2}\Big(1-\sigma(x)\Big)^{\frac{1}{\lambda^2}}.\end{aligned} \tag{21}$$

Let $\langle i, j\rangle$ be a user-item interaction and $r_{i,j}$ represents the label for $\langle i, j\rangle$ (1 if $\langle i, j\rangle$ is positive, otherwise 0). $\hat{R}$ indicates the outputs of the recommendation model for all the interactions in the user-item interaction set R. And $\hat{r}_{i,j}$ is the predicted probability of a specific user-item interaction $\langle i, j\rangle$ being positive. Then, the binary classification likelihood of the recommendation using negative sampling is as follows:

$$Pr(R|\hat{R}) = \prod_{\langle i,j\rangle\in R} Pr\big(r_{i,j}\big|\hat{r}_{i,j}\big) = \prod_{\langle i,j\rangle\in R}\left(\sigma\big(\hat{r}_{i,j}\big)\prod_{j'\in \mathrm{neg}_{\mathrm{rec}}(i)}\Big(1-\sigma\big(r_{i,j'}\big)\Big)\right), \tag{22}$$

where $n_{rec}(i)$ is the negative sample set for the user i.

Similar to Equation 15, we introduce a scalar β into Equation 22 to get a *scaled* version of the output from the recommendation model:

$$Pr(R|\hat{R},\beta) = \prod_{\langle i,j\rangle\in R}\left(\sigma(\frac{\hat{r}_{i,j}}{\beta^2})\prod_{j'\in \mathrm{neg}_{\mathrm{rec}}(i)}\Big(1-\sigma(\frac{\hat{r}_{i,j'}}{\beta^2})\Big)\right), \tag{23}$$

which can also be interpreted as a Boltzmann distribution (i.e., Gibbs distribution) like Equation 15. The input is scaled by β^2 (i.e., *temperature*). Then the log likelihood for the recommendation part in SPEX can be written as:

$$\begin{aligned}\log Pr\Big(R\big|\hat{R},\beta\Big) &= \sum_{\langle i,j\rangle\in R}\left(\log\Big(\sigma(\frac{\hat{r}_{i,j}}{\beta^2})\Big) + \sum_{j'\in \mathrm{neg}_{\mathrm{rec}}(i)}\log\Big(1-\sigma(\frac{\hat{r}_{i,j'}}{\beta^2})\Big)\right) \\ &\approx \sum_{\langle i,j\rangle\in R}\left(\frac{1}{\beta^2}\log\Big(\sigma\big(\hat{r}_{i,j}\big)\Big) + \frac{1}{\beta^2}\sum_{j'\in \mathrm{neg}_{\mathrm{rec}}(i)}\log\Big(1-\sigma\big(\hat{r}_{i,j'}\big)\Big) - 2(n_{rec}+1)\log\beta\right) \\ &= -\frac{1}{\beta^2}\mathcal{L}_{rec} - 2(n_{rec}+1)\cdot|R|\cdot\log\beta,\end{aligned} \tag{24}$$

where $|R|$ is the number of user-item interactions in R. We use the approximations in Equation 21 to the penultimate transition of Equation 24.

5.3.3 Optimization. We maximize the log likelihood of SPEX in maximum likelihood inference, while we minimize the joint objective during optimization. Thus, the joint loss $\mathcal{L}$ in Equation 14 can be formulated as:

$$\begin{aligned}\mathcal{L} &= -\log\Big(Pr\big(P\big|\hat{P},\alpha\big)\cdot Pr\big(R\big|\hat{R},\beta\big)\Big) \\ &\approx \frac{1}{\alpha^2}\mathcal{L}_{path} + \frac{1}{\beta^2}\mathcal{L}_{rec} + |P|\cdot\log\alpha + 2(n_{rec}+1)\cdot|R|\cdot\log\beta \\ &= \exp(-2\alpha')\cdot\mathcal{L}_{path} + \exp(-2\beta')\cdot\mathcal{L}_{rec} + |P|\cdot\alpha' + 2(n_{rec}+1)\cdot|R|\cdot\beta',\end{aligned} \tag{25}$$

where P is the path set, $\hat{P}$ is the outputs of the path prediction module for all paths, and $|P|$ is the number of total hops in all paths. Equation 19 (aggregation of all paths) and Equation 24 are used in the penultimate transition of Equation 25. In the last transition of Equation 25, we let $\alpha' = \log\alpha$

and $\beta' = \log \beta$, and train the model to learn α', β' instead of the unconstrained scalars α, β. This is for the numerical stability since $\frac{1}{\alpha^2}$, $\frac{1}{\beta^2}$ may encounter the overflow error for very small α and β, and $\log \alpha$, $\log \beta$ will have the math domain error for nonpositive α and β.

The above mechanism learns the relative weights of $\mathcal{L}_{path}$ and $\mathcal{L}_{rec}$, and automatically balances the path prediction task and the social recommendation task. α' and β' are automatically learned and used for balancing. A small value of α' (β') will decrease the contribution of $\mathcal{L}_{path}$ ($\mathcal{L}_{rec}$), whereas a large value will increase its contribution. α' and β' are regulated by the last two terms in Equation 25. This way, we provide an automatic mechanism to balance the two tasks. Compared to previous works [29, 60], our method fits the common setting of using negative sampling for training RS. *It is nontrivial to design such a new approach for automatic task balancing without the approximations that we proposed in Equations 20 and 21.*

5.4 Complexity of SPEX

Observed from Equation 25 that the cost of SPEX mainly consists of computing two parts: $\mathcal{L}_{rec}$ and $\mathcal{L}_{path}$. The complexity of $\mathcal{L}_{rec}$ depends on the chosen RS. The cost for the calculation of $\mathcal{L}_{path}$ is dominated by the g independent attention-based aggregations. Each aggregation receives state information from each neighbor node (i.e., $\mathcal{N}_{i,p}$ in Equation 2) and updates the state of the concerned nodes. The cost for the propagation is $O\big(\bar{m}(\bar{n}+1)p\big)$ where $\bar{m}$ is the average number of nodes per path, $\bar{n}$ is the average direct neighbors of each node in one path (note that $\mathcal{N}_{i,p}$ in Equation 2 contains both direct neighbors in the path and the concerned node itself), and p is the number of paths. Therefore, the overall cost for SPEX is $O(R) + O\big(g\bar{m}(\bar{n}+1)p\big)$, where $O(R)$ is the complexity of the RS model.

The additional cost of SPEX, compared to original RS models, is for path prediction. It is worthy pointing out that prevalent GNN based sequence modeling methods that are used in sequential RS [83] typically have a complexity of $O(\bar{m}^2 p)$ where $\bar{m}$ is the average number of nodes per sequence and p is the number of sequences. The cost for each attention-based aggregation in SPEX is $O\big(\bar{m}(\bar{n}+1)p\big)$, which is of the same order of magnitude as $O(\bar{m}^2 p)$. To stabilize the learning process, we adopt the multi-head attention mechanism which increases the cost of path prediction to $O\big(g\bar{m}(\bar{n}+1)p\big)$. However, a small value (e.g., 1-3) as we will show in our experiments (see Section 6.7) is sufficient for g. Therefore, the actually cost $O\big(g\bar{m}(\bar{n}+1)p\big)$ of path prediction in SPEX does not deviate too much from the common cost $O(\bar{m}^2 p)$ of prevalent GNN based sequence modeling methods.

6 EXPERIMENT

In this section, we conduct an experimental study using real data sets to verify the effectiveness of SPEX. Through experiments, we aim to answer the following research questions:

- **RQ1:** Can SPEX improve the performance of state-of-the-art recommendation models for social recommendation?
- **RQ2:** How does SPEX perform on the auxiliary task, i.e., path prediction?
- **RQ3:** Is SPEX able to further enhance RS that also model social homophily?
- **RQ4:** Does the automatic task balancing mechanism in SPEX provide better performance than the naive method which sets the task weights equally?
- **RQ5:** What are the effects of choosing different settings (e.g., feature sharing and number of heads) in SPEX?

In the sequel, we first explain our experimental settings. Then, we provide experimental results and analyses to answer the above questions. We will investigate RQ1, RQ2, RQ3 and RQ4 in

Table 1. Statistics of three data sets.

	Epinions	Weibo	Twitter
# of Users	12,645	6,812	8,930
# of Items	12,455	19,519	232,849
# of User-item Interactions	365,219	157,555	466,259
# of User-User Relations	699,893	133,712	96,718
# of Influence Propagation Paths	152,638	189,468	35,402

Sections 6.2, 6.3, 6.4, and 6.5, respectively. RQ5 will be analyzed in Sections 6.6 and 6.7. The source code of our work is public available[6].

6.1 Experiment Settings

Data: SPEX can handle traditional social relationships (e.g., friends or followers). However, we do not find public data sets containing creation time or order of creation for social relationships. Since constructing influence propagation paths requires such information, we adopt three public social recommendation data sets Epinions, Weibo and Twitter which provide creation time for alternative social relations including trust, retweeting and replying in our experiments. The statistics of the data are shown in Table 1. In the following, we briefly introduce three data sets:

- **Epinions**[7] contains user ratings on products. In addition, each user can indicate their trust (with creation time) towards other users' ratings or reviews. The trust relation is treated as the social relation in our experiments.
- **Weibo**[8] contains users from Sina Weibo[9] as well as their tweets and retweets. As the retweeting behavior shows interest and trust, we treat it as the social relationship. If a user u_i has retweeted a microblog from another user u_j at time t, a social relation from u_i to u_j will be created with the timestamp t.
- **Twitter**[10] contains the retweeting and replying behaviors crawled from Twitter. If a user u_i retweets or replies to a tweet from user u_j at time t, a social relation from u_i to u_j will be created with the timestamp t.

Recommendation Models: The design goal of SPEX is to make it flexible and easy to be plugged into existing recommendation models. Therefore, we choose the following state-of-the-art neural social recommenders and general neural recommenders, and verify whether their performance on social recommendation can be further improved with the help of SPEX *without heavy modifications*:

- **NCF**[11] [21] is the neural network based collaborative filtering method which ensembles both generalized matrix factorization (GMF) and multi-layer perceptron (MLP).
- **NGCF**[12] [76] is the neural graph collaborative filtering method which exploits the user-item bipartite graph and propagates embeddings on it so that the collaborative signal can be injected into the embedding process.

[6]https://github.com/XMUDM/SPEX
[7]https://www.cse.msu.edu/~tangjili/trust.html
[8]https://www.aminer.cn/data-sna#Weibo-Net-Tweet
[9]https://www.weibo.com
[10]https://www.aminer.cn/data-sna#Twitter-Dynamic-Net
[11]https://github.com/guoyang9/NCF
[12]https://github.com/liu-jc/PyTorch_NGCF

- **LightGCN**[13] [20] is the improved version of NGCF which only includes the most essential component of GNN to make the model more concise and appropriate for the recommendation task.
- **GraphRec**[14] [11] is a GNN based social recommendation approach which captures both the interactions in user-item bipartite graph and opinions in user-user social network to provide better social recommendation.
- **SAMN**[15] [5] uses the memory network to learn user-friend relation vectors, which can capture the varying aspect attentions that a user shares with his/her different friends. A friend-level attention component is used to identify different influence strength of a user's friends.
- **DiffNet++**[16] [80] models both neural influence diffusion in user-user social network and interest diffusion in user-item interaction graph for social recommendation.
- **FuseRec** [51] is a fusion recommender which models several social impacts (including social homophily) separately and later combines them to make recommendations.
- **DANSER**[17] [82] is the dual graph attention networks which collaboratively learn representations for different social impacts, including social homophily, for social recommendation.

For RS except FuseRec, we directly modify their implementations from original authors, and plug SPEX into them. The modifications include:

- For NGCF, the original paper adopts Bayesian Personalized Ranking loss (BPR) [57] and we modify it to use binary cross entropy loss so that all recommendation approaches are compared fairly.
- NCF has two types of user embeddings for GMF (d-dimensional embeddings) and MLP ($2d$-dimensional embeddings), respectively. When adopting SPEX to enhance its performance, we concatenate two types of user embeddings ($3d$-dimensional embeddings) and pass it to a single-layer feedforward neural network with ReLU to get a unified user embeddings (d-dimensional embeddings). The unified user embeddings are used as the input (from NCF) to the feature sharing module. For the recommendation model itself, the two types of user embeddings are still used to keep the recommendation process the same as the original NCF.
- Since we assume the simplest setting of social recommendation and there is no additional user/item features, we remove the fusion layer that fuses additional features in DiffNet++ as suggested by the authors [80].

For FuseRec, the implementation is not public available. Thus, we carefully implement it following the guidance of its paper [51] and then plug SPEX into it.

Notations: In our reported results, "XX & SPEX" denotes that our framework SPEX is used to enhance the recommendation model "XX" where "XX" is one of the recommendation approaches used in our experiments. The percentage, if shown, indicates the improvement of the corresponding model, when enhanced with SPEX, over its original version. Positive improvements or best results are shown in bold fonts.

Hardware: The experiments were run on a machine with two Intel(R) Xeon(R) CPU E5-2678 v3 @ 2.50GHz, 256 GB main memory and 8 GeForce RTX 2080 Ti graphics cards with 11 GB memory per

[13]https://github.com/gusye1234/LightGCN-PyTorch
[14]https://github.com/wenqifan03/GraphRec-WWW19
[15]https://github.com/chenchongthu/SAMN
[16]https://github.com/PeiJieSun/diffnet
[17]https://github.com/qitianwu/DANSER-WWW-19

card. During running, each program will monopolize one graphics card even if it does not require the complete 11 GB video memory.

Hyper-parameters: For all methods, negative sample number n_{rec} is set to 5, and batch size is 256. For all method except DANSER, the embedding size d is set to 64. DANSER with the embedding size 64 will encounter the out-of-memory problem using the graphics card in our experiment environment. Thus, we compare the original DNASER and the SPEX-enhanced DNASER using 16 as the embedding size for both methods. Adam optimizer [30] is used for all methods with an initial learning rate of 0.001. For SPEX, the default number of heads g in path prediction is 3. By default, SPEX uses the direct feature sharing as shown in Figure 2. For all methods, the initial embeddings are generated from $\mathcal{N}(0, 1)$.

Evaluation: We evaluate the recommendation performance and the path prediction performance of all the models using two common evaluation measures for ranking tasks, namely *Recall@K* and *NDCG@K* with $K = \{10, 20, 50\}$:

- We adopt the *leave-one-out* evaluation used by NCF [21], which is also widely used in the evaluation of other RS, to assess the recommendation performance. For each user, we hold out his/her latest interacted item as test data, the second latest interacted item as validation data, and the remaining interacted items as training data. We treat each test item as the positive item and randomly sample 99 negative items that this user has not interacted with before. Then, the model ranks the 100 items (1 positive and 99 negative items) for this user and we evaluate whether the positive item is in the top K. We calculate *Recall@K* and *NDCG@K* for each test user and report the average score for all test users.
- To assess the performance of path prediction, paths in each data set are randomly split into training set (90%) and test set (10%). The user to be predict in a path is regarded as the positive user. Assume that his/her preceding node in this path is u_p. We randomly sample 99 users, who are not direct social neighbors of u_p in the data set, as negative users. Then, the model ranks the 100 users (1 positive and 99 negative users) and we evaluate whether the positive user is in the top K. We calculate *Recall@K* and *NDCG@K* for each test path and report the average score for all test paths.

6.2 Performance of SPEX on Enhancing Social Recommendation (RQ1)

Tables 2, 3 and 4 illustrate the results of NCF, NGCF, LightGCN, GraphRec, SAMN and DiffNet++ on three data sets for the social recommendation task. FuseRec and DANSER are social homophily based models and we will investigate the influence of SPEX on them in Section 6.4 separately for RQ3.

From the results in Tables 2, 3 and 4, we can observe that SPEX is able to improve the social recommendation performance of NCF, NGCF, LightGCN, GraphRec, SAMN and DiffNet++ in most cases. To be specific, we can see that:

- For data sets Epinions and Weibo, all the RS models consistently get improved by SPEX with a large margin for both *Recall@K* and *NDCG@K*. For instance, modeling social recommendation task and path prediction task collaboratively using SPEX helps enhance the recommendation performance of NCF by 2.51%-8.38% and 5.75%-15.27% on Epinions and Weibo, respectively.
- For data set Twitter, all the RS models can get improved by SPEX, but the improvements are not as significant as the results on Epinions and Weibo. The reason is that RS models without SPEX already achieve hard-to-improve performance on Twitter data set and there is

relatively little room for improvements. For example, LightGCN achieves 0.9852, which is close to 1, for *Recall*@10 on Twitter. Using SPEX boosts its performance by 0.47% only.

The above observations verify that SPEX achieves our main design goal: *Improve the performance of RS models in social recommendation task via modeling the social recommendation task, the formation of social homophily, and their mutual effect in the manner of multi-task learning.*

Table 2. Performance of social recommendation on Epinions.

Method	Recall			NDCG		
	@10	@20	@50	@10	@20	@50
NCF	0.5973	0.7428	0.9161	0.3806	0.4155	0.4517
NCF & SPEX	0.6421 ↑**7.50%**	0.7812 ↑**5.17%**	0.9391 ↑**2.51%**	0.4125 ↑**8.38%**	0.4387 ↑**5.58%**	0.4702 ↑**4.10%**
NGCF	0.6520	0.7805	0.9376	0.4249	0.4574	0.4889
NGCF & SPEX	0.6669 ↑**2.29%**	0.7952 ↑**1.88%**	0.9483 ↑**1.14%**	0.4332 ↑**1.95%**	0.4658 ↑**1.84%**	0.4945 ↑**1.15%**
LightGCN	0.6510	0.7575	0.8939	0.4365	0.4631	0.4900
LightGCN & SPEX	0.6639 ↑**1.98%**	0.7964 ↑**5.14%**	0.9436 ↑**5.56%**	0.4379 ↑**0.32%**	0.4648 ↑**0.37%**	0.4932 ↑**0.65%**
GraphRec	0.6062	0.7661	0.9410	0.3526	0.3915	0.4291
GraphRec & SPEX	0.6301 ↑**3.94%**	0.7828 ↑**2.18%**	0.9507 ↑**1.03%**	0.3824 ↑**8.45%**	0.4211 ↑**7.56%**	0.4548 ↑**5.99%**
SAMN	0.6130	0.7557	0.9177	0.3690	0.4051	0.4382
SAMN & SPEX	0.6265 ↑**2.20%**	0.7687 ↑**1.72%**	0.9300 ↑**1.34%**	0.3801 ↑**3.01%**	0.4173 ↑**3.01%**	0.4492 ↑**2.51%**
DiffNet++	0.6561	0.7900	0.9386	0.4273	0.4612	0.4909
DiffNet++ & SPEX	0.6699 ↑**2.10%**	0.7984 ↑**1.06%**	0.9451 ↑**0.69%**	0.4397 ↑**2.90%**	0.4722 ↑**2.39%**	0.5016 ↑**2.18%**

Table 3. Performance of social recommendation on Weibo.

Method	Recall			NDCG		
	@10	@20	@50	@10	@20	@50
NCF	0.4866	0.6173	0.8315	0.3136	0.3465	0.3890
NCF & SPEX	0.5512 ↑**13.28%**	0.6913 ↑**11.99%**	0.8793 ↑**5.75%**	0.3615 ↑**15.27%**	0.3901 ↑**12.58%**	0.4274 ↑**9.87%**
NGCF	0.5118	0.6285	0.8307	0.3393	0.3704	0.4093
NGCF & SPEX	0.5896 ↑**15.20%**	0.7173 ↑**14.13%**	0.8900 ↑**7.14%**	0.3936 ↑**16.00%**	0.4260 ↑**15.01%**	0.4603 ↑**12.46%**
LightGCN	0.5150	0.6332	0.8110	0.3423	0.3706	0.4052
LightGCN & SPEX	0.6257 ↑**21.50%**	0.7495 ↑**18.37%**	0.9076 ↑**11.91%**	0.4270 ↑**24.74%**	0.4581 ↑**23.61%**	0.4895 ↑**20.80%**
GraphRec	0.4829	0.6492	0.8693	0.2851	0.3232	0.3697
GraphRec & SPEX	0.5217 ↑**8.03%**	0.6620 ↑**1.97%**	0.8752 ↑**0.68%**	0.3262 ↑**14.42%**	0.3616 ↑**11.88%**	0.4012 ↑**8.52%**
SAMN	0.4533	0.5815	0.7828	0.2859	0.3187	0.3601
SAMN & SPEX	0.5400 ↑**19.13%**	0.6607 ↑**13.62%**	0.8478 ↑**8.30%**	0.3581 ↑**25.25%**	0.3871 ↑**21.46%**	0.4248 ↑**17.97%**
DiffNet++	0.5585	0.6788	0.8598	0.3840	0.4144	0.4503
DiffNet++ & SPEX	0.6338 ↑**13.48%**	0.7527 ↑**10.89%**	0.9106 ↑**5.91%**	0.4376 ↑**13.96%**	0.4677 ↑**12.86%**	0.4993 ↑**10.88%**

Table 4. Performance of social recommendation on Twitter.

Method	Recall			NDCG		
	@10	@20	@50	@10	@20	@50
NCF	0.9848	0.9897	0.9960	0.8891	0.8904	0.8918
NCF & SPEX	0.9849	0.9908	0.9964	0.8893	0.8907	0.8920
	↑0.01%	**↑0.11%**	**↑0.04%**	**↑0.02%**	**↑0.03%**	**↑0.02%**
NGCF	0.9555	0.9620	0.9713	0.8473	0.8489	0.8507
NGCF & SPEX	0.9649	0.9782	0.9920	0.8791	0.8821	0.8848
	↑0.98%	**↑1.68%**	**↑2.13%**	**↑3.75%**	**↑3.91%**	**↑4.01%**
LightGCN	0.9852	0.9889	0.9947	0.8720	0.8729	0.8743
LightGCN & SPEX	0.9898	0.9924	0.9969	0.8825	0.8832	0.8839
	↑0.47%	**↑0.35%**	**↑0.22%**	**↑1.20%**	**↑1.18%**	**↑1.10%**
GraphRec	0.9251	0.9730	0.9954	0.7365	0.7426	0.7479
GraphRec & SPEX	0.9695	0.9853	0.9973	0.7969	0.8037	0.8087
	↑4.80%	**↑1.26%**	**↑0.19%**	**↑8.20%**	**↑8.23%**	**↑8.13%**
SAMN	0.9852	0.9886	0.9924	0.8807	0.8814	0.8821
SAMN & SPEX	0.9907	0.9929	0.9948	0.8914	0.8843	0.8830
	↑0.56%	**↑0.43%**	**↑0.24%**	**↑1.21%**	**↑0.33%**	**↑0.10%**
DiffNet++	0.9819	0.9908	0.9953	0.8804	0.8815	0.8824
DiffNet++ & SPEX	0.9931	0.9947	0.9954	0.8991	0.8996	0.8997
	↑1.14%	**↑0.39%**	**↑0.01%**	**↑2.12%**	**↑2.05%**	**↑1.96%**

6.3 Performance of SPEX on Enhancing Path Prediction (RQ2)

Next, we report and analyze the performance on the auxiliary task, i.e., path prediction. In Tables 5, 6 and 7, we show the path prediction results of using the path prediction module in SPEX only (denoted as "SPEX") and the complete SPEX with NCF, NGCF, LightGCN, GraphRec, SAMN or DiffNet++ embedded. FuseRec and DANSER are social homophily based models and we will investigate the influence of SPEX on them in Section 6.4 separately for RQ3.

Table 5. Performance of path prediction on Epinions.

Method	Recall			NDCG		
	@10	@20	@50	@10	@20	@50
SPEX	0.7883	0.8562	0.9267	0.6058	0.6231	0.6372
NCF & SPEX	0.8105	0.8683	0.9349	0.6232	0.6370	0.6484
	↑2.82%	**↑1.41%**	**↑0.88%**	**↑2.87%**	**↑2.23%**	**↑1.76%**
NGCF & SPEX	0.7975	0.8615	0.9277	0.6160	0.6321	0.6453
	↑1.17%	**↑0.62%**	**↑0.11%**	**↑1.68%**	**↑1.44%**	**↑1.27%**
LightGCN & SPEX	0.8619	0.9167	0.9650	0.6764	0.6903	0.7000
	↑9.34%	**↑7.07%**	**↑4.13%**	**↑11.65%**	**↑10.78%**	**↑9.86%**
GraphRec & SPEX	0.8812	0.9318	0.9742	0.6974	0.7103	0.7188
	↑11.78%	**↑8.83%**	**↑5.13%**	**↑15.12%**	**↑13.99%**	**↑12.81%**
SAMN & SPEX	0.8646	0.9212	0.9656	0.6693	0.6837	0.6927
	↑9.68%	**↑7.59%**	**↑4.20%**	**↑10.48%**	**↑9.73%**	**↑8.71%**
DiffNet++ & SPEX	0.8312	0.8981	0.9541	0.6446	0.6571	0.6641
	↑5.44%	**↑4.89%**	**↑2.96%**	**↑6.40%**	**↑5.46%**	**↑4.22%**

Table 6. Performance of path prediction on Weibo.

Method	Recall			NDCG		
	@10	@20	@50	@10	@20	@50
SPEX	0.8932	0.9239	0.9570	0.7885	0.7962	0.8029
NCF & SPEX	0.9193 ↑**2.92%**	0.9457 ↑**2.36%**	0.9705 ↑**1.41%**	0.8121 ↑**2.99%**	0.8187 ↑**2.83%**	0.8237 ↑**2.59%**
NGCF & SPEX	0.9314 ↑**4.28%**	0.9518 ↑**3.02%**	0.9725 ↑**1.62%**	0.8277 ↑**4.97%**	0.8329 ↑**4.61%**	0.8370 ↑**4.25%**
LightGCN & SPEX	0.9519 ↑**6.57%**	0.9691 ↑**4.89%**	0.9854 ↑**2.97%**	0.8496 ↑**7.75%**	0.8547 ↑**7.35%**	0.8580 ↑**6.86%**
GraphRec & SPEX	0.9333 ↑**4.49%**	0.9534 ↑**3.19%**	0.9747 ↑**1.85%**	0.8334 ↑**5.69%**	0.8385 ↑**5.31%**	0.8428 ↑**4.97%**
SAMN & SPEX	0.9490 ↑**6.25%**	0.9651 ↑**4.46%**	0.9811 ↑**2.52%**	0.8495 ↑**7.74%**	0.8536 ↑**7.21%**	0.8568 ↑**6.71%**
DiffNet++ & SPEX	0.9149 ↑**2.43%**	0.9386 ↑**1.59%**	0.9655 ↑**0.89%**	0.8137 ↑**3.20%**	0.8197 ↑**2.95%**	0.8251 ↑**2.76%**

Table 7. Performance of path prediction on Twitter.

Method	Recall			NDCG		
	@10	@20	@50	@10	@20	@50
SPEX	0.7649	0.8160	0.8830	0.5549	0.5726	0.5867
NCF & SPEX	0.8784 ↑**14.84%**	0.9143 ↑**12.05%**	0.9625 ↑**9.00%**	0.6873 ↑**23.86%**	0.7027 ↑**22.72%**	0.7162 ↑**22.07%**
NGCF & SPEX	0.8897 ↑**16.32%**	0.9239 ↑**13.22%**	0.9619 ↑**8.94%**	0.6949 ↑**25.23%**	0.7197 ↑**25.69%**	0.7264 ↑**23.81%**
LightGCN & SPEX	0.8748 ↑**14.37%**	0.8861 ↑**8.59%**	0.8935 ↑**1.19%**	0.7166 ↑**29.14%**	0.7295 ↑**27.40%**	0.7310 ↑**24.60%**
GraphRec & SPEX	0.8327 ↑**8.86%**	0.8610 ↑**5.51%**	0.9014 ↑**2.08%**	0.7005 ↑**26.24%**	0.7079 ↑**23.63%**	0.7163 ↑**22.09%**
SAMN & SPEX	0.8412 ↑**9.98%**	0.8889 ↑**8.93%**	0.9461 ↑**7.15%**	0.6629 ↑**19.46%**	0.6751 ↑**17.90%**	0.6865 ↑**17.01%**
DiffNet++ & SPEX	0.8091 ↑**5.78%**	0.8582 ↑**5.17%**	0.9161 ↑**3.75%**	0.6841 ↑**23.28%**	0.6964 ↑**21.62%**	0.7081 ↑**20.69%**

We can find that, in most cases, SPEX can produce better performance for predicting paths with various RS embedded. Specifically, we can observe that:

- Solely using SPEX can achieve satisfying results for predicting paths in terms of both *Recall@K* and *NDCG@K*, showing that our GNN based design for the path prediction module in SPEX is reasonable and practical.
- In most cases, modeling both the social recommendation task and the path prediction task can improve the performance of path prediction. For example, SAMN & SPEX can improve the performance of SPEX by 4.20%-10.48%, 2.52%-7.74% and 7.15%-19.46% on Epinions, Weibo and Twitter, respectively.

Based on the above observations for path prediction and the conclusion from Section 6.2, we can conclude that: *The design of multi-task learning in SPEX can benefit both the social recommendation task and the path prediction task.*

6.4 Performance of SPEX on Other Social Homophily Based RS (RQ3)

Similar to the idea of SPEX, FuseRec and DANSER are designed to collaboratively model several important factors, including social homophily, for social recommendation. We compare their performance before and after using SPEX to investigate whether SPEX can further improve social homophily based methods. It is worth noting that, as pointed out in the descriptions for our hyper-parameter settings, DANSER with the embedding size 64 will encounter the out-of-memory problem using the graphics card in our experiment environment. Therefore, for the experiments in this section, we set the embedding size to 64 for FuseRec (like other RS models) and 16 for DANSER, respectively.

Tables 8 and 9 illustrate the performance of FuseRec and DANSER for social recommendation and path prediction, respectively. Since Weibo and Twitter data sets do not contain item category information that FuseRec requires, we only report results on Epinions in this section. From Table 8, we can see that SPEX is able to further enhance the recommendation performance of FuseRec and DANSER. FuseRec and DANSER already consider modeling social homophily. Hence the increase percentages of using SPEX over FuseRec and DANSER are not as significant as what we have observed on other RS models in Section 6.2. But the improvements are still consistent for different evaluation measures. We can draw a similar conclusion for the path prediction task from Table 9. In summary, *SPEX still works on social homophily based RS methods, i.e., both social recommendation and path prediction tasks can get further boosted.*

Table 8. Recommendation performance of SPEX-enhanced FuseRec and DANSER on Epinions.

Method	Recall			NDCG		
	@10	@20	@50	@10	@20	@50
FuseRec ($d = 64$)	0.5444	0.6930	0.8838	0.3316	0.3691	0.4072
FuseRec & SPEX ($d = 64$)	0.5494 **↑0.92%**	0.6984 **↑0.78%**	0.8875 **↑0.42%**	0.3369 **↑1.60%**	0.3735 **↑1.19%**	0.4113 **↑1.01%**
DANSER ($d = 16$)	0.4221	0.5662	0.7831	0.2503	0.2866	0.3300
DANSER & SPEX ($d = 16$)	0.4229 **↑0.19%**	0.5670 **↑0.14%**	0.7839 **↑0.10%**	0.2509 **↑0.24%**	0.2871 **↑0.17%**	0.3304 **↑0.12%**

Table 9. Path prediction performance of SPEX-enhanced FuseRec and DANSER on Epinions.

Method	Recall			NDCG		
	@10	@20	@50	@10	@20	@50
SPEX ($d = 64$)	0.7980	0.8198	0.8453	0.6390	0.6446	0.6496
FuseRec & SPEX ($d = 64$)	0.8161 **↑2.27%**	0.8293 **↑1.16%**	0.8502 **↑0.58%**	0.6430 **↑0.63%**	0.6474 **↑0.43%**	0.6516 **↑0.31%**
SPEX ($d = 16$)	0.7492	0.8487	0.9321	0.5210	0.5463	0.5631
DANSER & SPEX ($d = 16$)	0.7700 **↑2.78%**	0.8637 **↑1.77%**	0.9397 **↑0.82%**	0.5453 **↑4.66%**	0.5692 **↑4.19%**	0.5845 **↑3.80%**

6.5 Effects of Automatic Task Balancing (RQ4)

We also compare the automatic task balancing method introduced in Section 5.3 with a common method for setting task weights in multi-task learning, i.e., assigning equal weights to the two tasks for the joint loss in Equation 14.

Figure 4 illustrates the social recommendation performance and the path prediction performance of SPEX-enhanced NCF, NGCF, LightGCN, GraphRec, SAMN or DiffNet++ on Epinions using different settings for SPEX. In Figure 4, "XX & SPEX (1:1)" indicate that task weights are manually set to be equal and kept fixed during optimization when using SPEX to enhance the RS model "XX". Other parts of "XX & SPEX (1:1)" are the same as "XX & SPEX". We can observe that, for most cases, our proposed automatic task balancing method (i.e., "XX & SPEX") can achieve better performance than manually setting task weights (i.e., "XX & SPEX (1:1)"). The better results show that, compared to the commonly used naive method, SPEX does not require users to manually tune the task weights to achieve good performance. Consequently, SPEX has the merit of reducing the human cost in tuning multi-task learning for improving social recommendation.

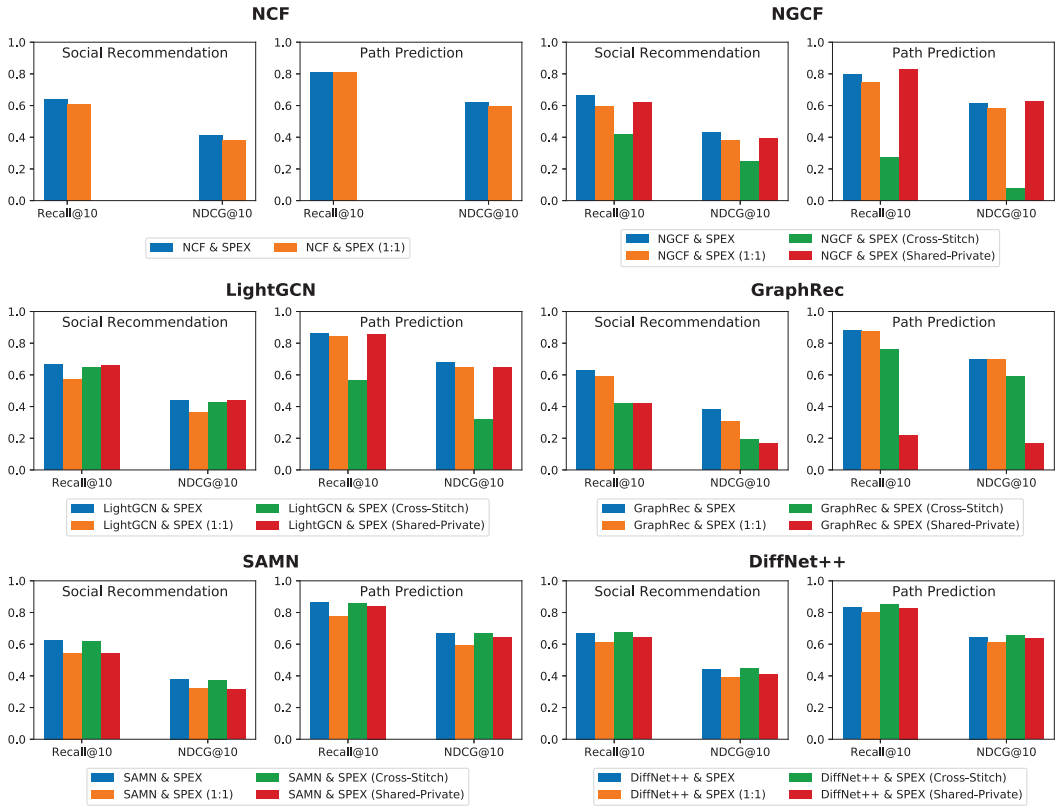

Fig. 4. Comparison of different settings of SPEX on Epinions.

We also report the changes of $exp(-2\alpha^{'})$, $exp(-2\beta^{'})$, $\mathcal{L}_{path}$ and $\mathcal{L}_{rec}$ in the first 50 epochs for NGCF & SPEX on Twitter in Figure 5. As we can see, both $\mathcal{L}_{path}$ and $\mathcal{L}_{rec}$ decrease smoothly with the help of the automatic task balancing mechanism, showing that SPEX is able to prevent loss fluctuations and help each task converge in multi-task learning. Moreover, we can observe that $exp(-2\alpha^{'})$ and $exp(-2\beta^{'})$ eventually approach 1.1 and 1.2, respectively. Recall that we let

$\alpha' = \log \alpha$ and $\beta' = \log \beta$ in Section 5.3.3 for numerical stability. Hence, we can estimate that α and β approach 0.9535 and 0.9129, respectively. Note that the approximations in Equations 18 and 20 become equalities when $\alpha \to 1$ and $\beta \to 1$ (β is denoted by λ in Equation 20), respectively. Therefore, we can conclude that the approximations in Equations 18 and 20 do not introduce large value changes in practice.

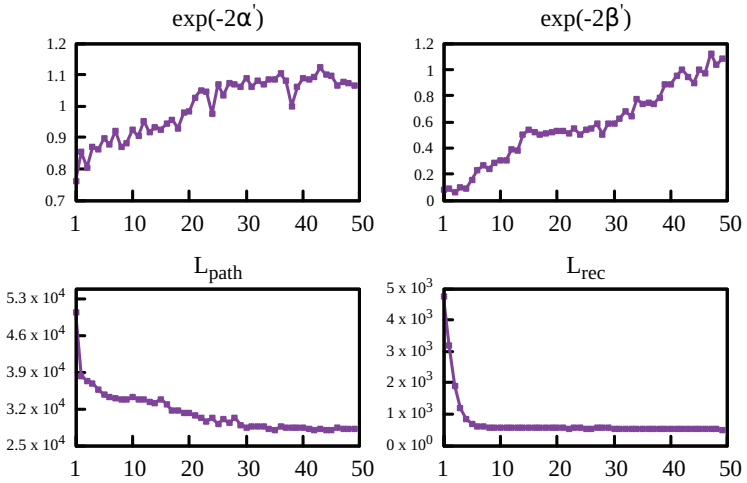

Fig. 5. Changes of $exp(-2\alpha')$, $exp(-2\beta')$, $\mathcal{L}_{path}$ and $\mathcal{L}_{rec}$ in the first 50 epochs for NGCF & SPEX on Twitter. X axis indicates the number of epochs.

6.6 Effects of Different Feature Sharing Designs (RQ5)

We then compare the different feature sharing methods introduced in Section 5.2, namely the direct sharing method, the correlation based Cross-Stitch unit and the Shared-Private method.

Figure 4 shows the social recommendation performance and the path prediction performance of SPEX-enhanced NCF, NGCF, LightGCN, GraphRec, SAMN or DiffNet++ on Epinions using different feature sharing methods in SPEX on Epinions. In Figure 4, "XX & SPEX (Cross-Stitch)" and "XX & SPEX (Shared-Private)" represent that the direct feature sharing method in "XX & SPEX" is replaced with the Cross-Stitch unit and the Shared-Private method, respectively. Other parts of "XX & SPEX (Cross-Stitch)" and "XX & SPEX (Shared-Private)" are the same as "XX & SPEX". Note that NCF has two types of user embeddings for GMF (d-dimensional embeddings) and MLP ($2d$-dimensional embeddings), respectively. Therefore, the correlation based Cross-Stitch unit and the Shared-Private method can not be directly adopted for NCF and they are not reported in Figure 4. Nevertheless, the simplest direct sharing method in SPEX (i.e., NCF & SPEX) can already significantly boost the performance of NCF as we have seen in previous sections.

From Figure 4, we can see a surprising result: direct feature sharing ("XX & SPEX") is the best feature sharing method in most cases. It does not exceed another more complex feature sharing method only in two cases: enhancing NGCF or DiffNet++ for the path prediction task. But "XX & SPEX" is still the second best method with relatively good performance for the two cases. The underlying reason, which is also pointed out by He et al. in [20], may be that user data and item data in RS only has an ID without concrete semantics. More complex feature sharing designs are powerful in computer vision applications, because the same data in these applications may have different roles and semantics in different tasks [60] which can be better captured by complex feature

sharing mechanisms. For RS, simple direct feature sharing is sufficient and more sophisticated designs may even downgrade the performance.

6.7 Impact of Number of Heads (RQ5)

Lastly, we investigate the impact of number of heads on the performance of both social recommendation and path prediction tasks using SPEX-enhanced NCF, NGCF, LightGCN, GraphRec, SAMN or DiffNet++.

Figure 6 demonstrates the results when 1 head, 2 heads or 3 heads are used in SPEX on Weibo data set. Using 3 heads is the default setting for g in SPEX, i.e., "X & SPEX" in previous sections is the same as "X & SPEX (3 heads)" in this section. From the results reported in Figure 6, we can find that the performance get enhanced as the number of heads increases. But when we change from 2 heads to 3 heads, the improvements are marginal. We can observe similar trends on Epinions and Twitter data sets. Thus, we conclude that using 2 or 3 heads is sufficient for SPEX to work well.

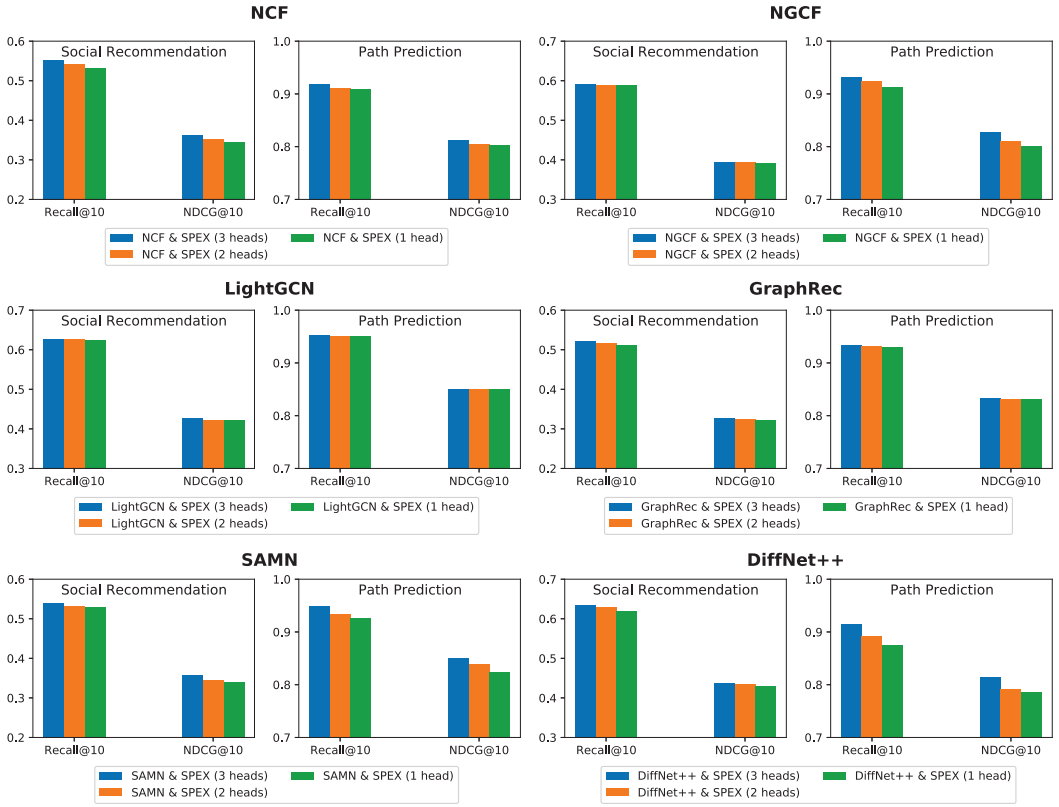

Fig. 6. Performance with varying head numbers on Weibo.

7 RELATED WORK

In this section, we will elaborate on several research areas which are related to our work. As SPEX adopts Graph Neural Network as its backbone, we will first discuss it. Then, we will introduce related work on recommender systems including social recommender systems and multi-task learning based recommender systems.

7.1 Graph Neural Network and Its Applications

Due to the prevalence of graph data in real world and the success of deep learning in various applications [53], there is a surge of works on extending deep learning techniques to graph data, i.e., Graph Neural Network (GNN) [84, 95].

In early studies of GNN [16, 61], each node has its input features and each edge may also have its features. A parametric function called local transition function, shared among all nodes, is used to update the node state according to the neighborhood. Another parametric function called local output function is used for producing the output of the node. Experimental results showed that such a vanilla GNN [16, 61] is powerful for modeling structural data. One limitation of the vanilla GNN is that the update of node hidden states follows a sequential process which suffers from the high computational cost [95].

Later, convolution operation is generalized to non-Euclidean graph data and Graph Convolutional Network (GCN) is developed to overcome the limitations of vanilla GNN. GCN can be divided into spectral methods and spatial methods. The former define the convolution via graph Fourier transform and convolution theorem [4, 9], while the latter define the convolution as a weighted average function over all vertices located in the neighborhood of the target vertex [17, 31, 49, 52]. Since spatial methods do not need to explicitly specify the convolution operation and they are in fact the general case of spectral methods, spatial methods have being deployed in numerous applications [84, 93, 95].

In addition to GCN, there are other variants trying to improve the vanilla GNN. Graph Recurrent Network (GRN) [36, 37] leverages the gate mechanism to facilitate the long-term information propagation across the graph. Graph Attention Network (GAT) [71] incorporates the attention mechanism into GNN and assign different attention score to each neighbor. Graph Pooling [88] adopts the idea of pooling operation to generate hierarchical representations of graphs.

7.2 Recommender Systems

7.2.1 Social Recommender Systems. Recommender systems (RS) are an essential tool to overcome information overload [1]. Social recommender systems (SRS) have been an important domain in the RS community for a long time [68, 86]. It leverages and models additional social features provided by the social network to enhance recommendation.

Early studies of SRS have exploited using social network to improve traditional RS. Ma et al. propose various methods for social recommendation including employing matrix factorization (MF) over both social network and user-item interaction graph [45], capturing social trust to enhance MF [43], modeling both social trust and social distruct relations [44], introducing social regularization into MF [46]. Jamali and Ester propose to use trust propagation to improve MF [25]. Later, they design another method which combines random walk and collaborative filtering to capture social trust and improve recommendation [24]. Yang et al. [87] propose category-specific social trust circles to improve MF. Li et al. [34] uses temporal influence to improve SRS. Li et al. [35] introduce social community based regularization to enhance MF.

Recently, GNN has become dominating for the social recommendation problem, since it is a natural fit for modeling information diffusion in social network. Song et al. [63] propose a dynamic GNN to capture the dynamic interests of users in SRS. Fan et al. [11] propose GraphRec which aggregates both user-item interactions and user-user social interactions in SRS. Wu et al. [81] design DiffNet which simulates the diffusion of social influence by recursively using a influence diffusion layer. They further extend DiffNet to DiffNet++ [80] and consider the diffusion processes in both user-user social network and user-item interaction graph. Xu et al. [85] introduce the relation-aware aggregation function into the GNN based social recommender and the aggregation depends on the

type of the edge. Furthermore, they utilize meta-paths in Heterogeneous Information Network [62] to better depict the nodes in the graph. Wang et al. [73] design the translation-based opinion elicitation to identify the elite opinions and then use GNN to model their diffusion when making recommendations. Wu et al. [82] propose a dual GAT to collaboratively learn representations for different social effects in SRS. M et al. [42] merge user-item interaction graph and social network into one unified graph and use multi-head GATs to predict the rating of a user-item pair. Yu et al. [90] tailor GNN to aggregate information from motif-induced neighborhood to generate new social relations to overcome data sparsity.

In addition to GNN, other deep learning techniques have also been successfully deployed in SRS. Sun et al. [65] use LSTM with attention to model both static and dynamic preferences. Narang et al. [51] propose a fusion RS which models multiple important factors (e.g., temporal context, social influence and item similarity) for social recommendation using LSTM and the attention mechanism. Fan et al. [12] design a random walk based method to sample useful item-aware social sequences and then feeds them into Bi-LSTM to obtain the representations. Chen et al. [6] opt to model the social recommendation problem via bipartite graph embeddings, self-attention mechanism and inductive learning. Chen et al. [5] propose to use an attention based memory network to capture user-friend relations. Fan et al. [10] and Yu et al. [89] harness Generative Adversarial Network to provide informative and reliable guidance towards the training of social recommendation models.

As explained in Section 5.1, SPEX is a general framework and it is orthogonal to various existing RS models (i.e., neural SRS, general neural RS and matrix factorization based methods) that are applicable to the social recommendation task. Thus, SPEX can be plugged into existing systems to enhance social recommendation.

7.2.2 Recommender Systems Using Multi-task Learning. Although few works use multi-task learning to assist the social recommendation task, multi-task learning has shown its power in the general recommendation task or other specific recommendation tasks.

Gao et al. [15] consider the cascading relationship among different types of behaviors in RS and perform a joint optimization. Jin et al. [26] design a GNN based method to learn the strength of different behaviors in RS via user-item propagation layer and capture behavior semantics by item-item propagation layer. Tang et al. [67] propose the Progressive Layered Extraction model with a new sharing structure design for the video recommendation task. Zhao et al. [94] adopt the idea of Mixture-of-Experts for multi-task video recommendation. Bansal et al. [2] design a multi-task learning based scientific paper recommendation model. To improve the interpretability of RS, several recent works [7, 41, 74**?**] perform item recommendation and recommendation explanation jointly. A few knowledge graph based RS [72, 77] perform some tasks over knowledge graphs jointly with the recommendation task in order to improve the overall results.

Compared to the above works, SPEX is specially designed for the social recommendation task and it is easy to use SPEX to enhance existing social recommendation models without heavy modifications. Another merit of SPEX is that it can automatically balance different tasks during the joint optimization and avoid manual tuning task weights.

8 CONCLUSION

In this paper, we explore how to model both the formation of social homophily and the social influence driven recommendation task to improve the performance of neural SRS or general RS when social information is available. Previous works only focus on leveraging social influence and neglect the importance of social homophily. We design a generic framework SPEX, which simultaneously models both aspects and their mutual effect in the manner of multi-task learning. Moreover, SPEX can automatically balance different tasks during the joint optimization to generate

better performance. Experiments verify that SPEX can enhance the performance of various neural SRS or general neural RS for the social recommendation task without heavy modifications. In the future, we plan to enhance the interpretability of the prediction results of influence propagation paths so that SPEX can help users understand how the social recommendation is provided. This way, SPEX will be able to increase user satisfaction in addition to the performance gain it already helps RS achieve.

ACKNOWLEDGEMENTS

Chen Lin is the corresponding author. Chen Lin is supported by the Natural Science Foundation of China (No. 61972328) and Joint Innovation Research Program of Fujian Province China (No.2020R0130). Hui Li is supported by the Natural Science Foundation of China (No. 62002303) and Natural Science Foundation of Fujian Province China (No. 2020J05001).

REFERENCES

[1] Charu C. Aggarwal. 2016. *Recommender Systems - The Textbook*. Springer.
[2] Trapit Bansal, David Belanger, and Andrew McCallum. 2016. *Ask the GRU*: Multi-task Learning for Deep Text Recommendations. In *RecSys*. 107–114.
[3] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. 2016. Interaction Networks for Learning about Objects, Relations and Physics. In *NIPS*. 4502–4510.
[4] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral Networks and Locally Connected Networks on Graphs. In *ICLR*.
[5] Chong Chen, Min Zhang, Yiqun Liu, and Shaoping Ma. 2019. Social Attentional Memory Network: Modeling Aspect- and Friend-Level Differences in Recommendation. In *WSDM*. 177–185.
[6] Hongxu Chen, Hongzhi Yin, Tong Chen, Weiqing Wang, Xue Li, and Xia Hu. 2020. Social Boosted Recommendation with Folded Bipartite Network Embedding. *IEEE Trans. Knowl. Data Eng.* (2020).
[7] Zhongxia Chen, Xiting Wang, Xing Xie, Tong Wu, Guoqing Bu, Yining Wang, and Enhong Chen. 2019. Co-Attentive Multi-Task Learning for Explainable Recommendation. In *IJCAI*. 2137–2143.
[8] Connor W Coley, Wengong Jin, Luke Rogers, Timothy F Jamison, Tommi S Jaakkola, William H Green, Regina Barzilay, and Klavs F Jensen. 2019. A graph-convolutional neural network model for the prediction of chemical reactivity. *Chemical Science* 10, 2 (2019), 370–377.
[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NIPS*. 3837–3845.
[10] Wenqi Fan, Tyler Derr, Yao Ma, Jianping Wang, Jiliang Tang, and Qing Li. 2019. Deep Adversarial Social Recommendation. In *IJCAI*. 1351–1357.
[11] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Yihong Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph Neural Networks for Social Recommendation. In *WWW*. 417–426.
[12] Wenqi Fan, Yao Ma, Dawei Yin, Jianping Wang, Jiliang Tang, and Qing Li. 2019. Deep social collaborative filtering. In *RecSys*. 305–313.
[13] Hui Fang, Danning Zhang, Yiheng Shu, and Guibing Guo. 2019. Deep Learning-based Sequential Recommender Systems: Concepts, Algorithms, and Evaluations. *arXiv Preprint* abs/1905.01997 (2019).
[14] Yarin Gal. 2016. *Uncertainty in Deep Learning*. Ph.D. Dissertation. Uncertainty in Deep Learning.
[15] Chen Gao, Xiangnan He, Dahua Gan, Xiangning Chen, Fuli Feng, Yong Li, Tat-Seng Chua, and Depeng Jin. 2019. Neural Multi-task Recommendation from Multi-behavior Data. In *ICDE*. 1554–1557.
[16] Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A New Model for Learning in Graph Domains. In *IJCNN*. 729–734.
[17] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. 1024–1034.
[18] Zhongkai Hao, Chengqiang Lu, Zhenya Huang, Hao Wang, Zheyuan Hu, Qi Liu, Enhong Chen, and Cheekong Lee. 2020. ASGN: An Active Semi-supervised Graph Neural Network for Molecular Property Prediction. In *KDD*. 731–752.
[19] Shonosuke Harada, Hirotaka Akita, Masashi Tsubaki, Yukino Baba, Ichigaku Takigawa, Yoshihiro Yamanishi, and Hisashi Kashima. 2020. Dual graph convolutional neural network for predicting chemical networks. *BMC Bioinformatics* 21 (2020), 1–13.
[20] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yong-Dong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR*. 639–648.

[21] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *WWW*. 173–182.
[22] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In *ICLR*.
[23] Binbin Hu, Chuan Shi, Wayne Xin Zhao, and Philip S. Yu. 2018. Leveraging Meta-path based Context for Top-N Recommendation with A Neural Co-Attention Model. In *KDD*. 1531–1540.
[24] Mohsen Jamali and Martin Ester. 2009. *TrustWalker*: a random walk model for combining trust-based and item-based recommendation. In *KDD*. 397–406.
[25] Mohsen Jamali and Martin Ester. 2010. A matrix factorization technique with trust propagation for recommendation in social networks. In *RecSys*. 135–142.
[26] Bowen Jin, Chen Gao, Xiangnan He, Depeng Jin, and Yong Li. 2020. Multi-behavior Recommendation with Graph Convolutional Networks. In *SIGIR*. 659–668.
[27] Wang-Cheng Kang and Julian J. McAuley. 2018. Self-Attentive Sequential Recommendation. In *ICDM*. 197–206.
[28] Alex Kendall and Yarin Gal. 2017. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?. In *NIPS*. 5574–5584.
[29] Alex Kendall, Yarin Gal, and Roberto Cipolla. 2018. Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. In *CVPR*. 7482–7491.
[30] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.
[31] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[32] Armen Der Kiureghian and Ove Ditlevsen. 2009. Aleatory or epistemic? Does it matter? *Structural Safety* 31, 2 (2009), 105–112.
[33] Yehuda Koren, Robert M. Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37.
[34] Hui Li, Dingming Wu, and Nikos Mamoulis. 2014. A revisit to social network-based recommender systems. In *SIGIR*. 1239–1242.
[35] Hui Li, Dingming Wu, Wenbin Tang, and Nikos Mamoulis. 2015. Overlapping Community Regularization for Rating Prediction in Social Recommender Systems. In *RecSys*. 27–34.
[36] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. 2016. Gated Graph Sequence Neural Networks. In *ICLR*.
[37] Xiaodan Liang, Xiaohui Shen, Jiashi Feng, Liang Lin, and Shuicheng Yan. 2016. Semantic Object Parsing with Graph LSTM. In *ECCV*, Vol. 9905. 125–143.
[38] Jovian Lin, Richard Jayadi Oentaryo, Ee-Peng Lim, Casey Vu, Adrian Vu, Agus Trisnajaya Kwee, and Philips Kokoh Prasetyo. 2016. A Business Zone Recommender System Based on Facebook and Urban Planning Data. In *ECIR*, Vol. 9626. 641–647.
[39] Xuan Lin, Zhe Quan, Zhi-Jie Wang, Tengfei Ma, and Xiangxiang Zeng. 2020. KGNN: Knowledge Graph Neural Network for Drug-Drug Interaction Prediction. In *IJCAI*. 2739–2745.
[40] Zheng Liu, Xiaohan Li, Hao Peng, Lifang He, and Philip S. Yu. 2020. Heterogeneous Similarity Graph Neural Network on Electronic Health Records. In *IEEE BigData*. 1196–1205.
[41] Yichao Lu, Ruihai Dong, and Barry Smyth. 2018. Why I like it: multi-task learning for recommendation and explanation. In *RecSys*. 4–12.
[42] Vijaikumar M, Shirish Shevad, and M N Murt. 2019. SoRecGAT: Leveraging Graph Attention Mechanism for Top-N Social Recommendation. In *ECML/PKDD*.
[43] Hao Ma, Irwin King, and Michael R. Lyu. 2009. Learning to recommend with social trust ensemble. In *SIGIR*. 203–210.
[44] Hao Ma, Michael R. Lyu, and Irwin King. 2009. Learning to recommend with trust and distrust relationships. In *RecSys*. 189–196.
[45] Hao Ma, Haixuan Yang, Michael R. Lyu, and Irwin King. 2008. SoRec: social recommendation using probabilistic matrix factorization. In *CIKM*. 931–940.
[46] Hao Ma, Dengyong Zhou, Chao Liu, Michael R. Lyu, and Irwin King. 2011. Recommender systems with social regularization. In *WSDM*. 287–296.
[47] PETER V. MARSDEN and NOAH E. FRIEDKIN. 1993. Network studies of social influence. *Sociol. Method.* 22, 1 (1993), 127–151.
[48] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a feather: Homophily in social networks. *Annu. Rev. Sociol.* 27, 1 (2001), 415–444.
[49] Alessio Micheli. 2009. Neural Network for Graphs: A Contextual Constructive Approach. *IEEE Trans. Neural Networks* 20, 3 (2009), 498–511.
[50] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. 2016. Cross-Stitch Networks for Multi-task Learning. In *CVPR*. 3994–4003.

[51] Kanika Narang, Yitong Song, Alexander G. Schwing, and Hari Sundaram. 2021. FuseRec: fusing user and item homophily modeling with temporal recommender systems. *Data Min. Knowl. Discov.* 35, 3 (2021), 837–862.
[52] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning Convolutional Neural Networks for Graphs. In *ICML*, Vol. 48. 2014–2023.
[53] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria E. Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. 2019. A Survey on Deep Learning: Algorithms, Techniques, and Applications. *ACM Comput. Surv.* 51, 5 (2019), 92:1–92:36.
[54] Ruihong Qiu, Zi Huang, Jingjing Li, and Hongzhi Yin. 2020. Exploiting Cross-session Information for Session-based Recommendation with Graph Neural Networks. *ACM Trans. Inf. Syst.* 38, 3 (2020), 22:1–22:23.
[55] Ruihong Qiu, Jingjing Li, Zi Huang, and Hongzhi Yin. 2019. Rethinking the Item Order in Session-based Recommendation with Graph Neural Networks. In *CIKM*. 579–588.
[56] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. 2018. Sequence-Aware Recommender Systems. *ACM Comput. Surv.* 51, 4 (2018), 66:1–66:36.
[57] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *UAI*. 452–461.
[58] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized Markov chains for next-basket recommendation. In *WWW*. 811–820.
[59] Steffen Rendle, Walid Krichene, Li Zhang, and John R. Anderson. 2020. Neural Collaborative Filtering vs. Matrix Factorization Revisited. In *RecSys*. 240–248.
[60] Sebastian Ruder. 2017. An Overview of Multi-Task Learning in Deep Neural Networks. *arXiv Preprint* (2017). https://arxiv.org/abs/1706.05098
[61] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The Graph Neural Network Model. *IEEE Trans. Neural Networks* 20, 1 (2009), 61–80.
[62] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and Philip S. Yu. 2017. A Survey of Heterogeneous Information Network Analysis. *IEEE Trans. Knowl. Data Eng.* 29, 1 (2017), 17–37.
[63] Weiping Song, Zhiping Xiao, Yifan Wang, Laurent Charlin, Ming Zhang, and Jian Tang. 2019. Session-Based Social Recommendation via Dynamic Graph Attention Networks. In *WSDM*. 555–563.
[64] Laure Soulier, Lynda Tamine, and Gia-Hung Nguyen. 2016. Answering Twitter Questions: a Model for Recommending Answerers through Social Collaboration. In *CIKM*. 267–276.
[65] Peijie Sun, Le Wu, and Meng Wang. 2018. Attentive Recurrent Social Recommendation. In *SIGIR*. 185–194.
[66] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. *Proc. VLDB Endow.* 4, 11 (2011), 992–1003.
[67] Hongyan Tang, Junning Liu, Ming Zhao, and Xudong Gong. 2020. Progressive Layered Extraction (PLE): A Novel Multi-Task Learning (MTL) Model for Personalized Recommendations. In *RecSys*. 269–278.
[68] Jiliang Tang, Xia Hu, and Huan Liu. 2013. Social recommendation: a review. *Social Netw. Analys. Mining* 3, 4 (2013), 1113–1133.
[69] Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In *WSDM*. 565–573.
[70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NIPS*. 5998–6008.
[71] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.
[72] Hongwei Wang, Fuzheng Zhang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2019. Multi-Task Feature Learning for Knowledge Graph Enhanced Recommendation. In *WWW*. 2000–2010.
[73] Jianling Wang, Kaize Ding, Ziwei Zhu, Yin Zhang, and James Caverlee. 2020. Key Opinion Leaders in Recommendation Systems: Opinion Elicitation and Diffusion. In *WSDM*. 636–644.
[74] Nan Wang, Hongning Wang, Yiling Jia, and Yue Yin. 2018. Explainable Recommendation via Multi-Task Learning in Opinionated Text Data. In *SIGIR*. 165–174.
[75] Shoujin Wang, Longbing Cao, and Yan Wang. 2019. A Survey on Session-based Recommender Systems. *arXiv Preprint* (2019). https://arxiv.org/abs/1902.04864
[76] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural Graph Collaborative Filtering. In *SIGIR*. 165–174.
[77] Xiang Wang, Dingxian Wang, Canran Xu, Xiangnan He, Yixin Cao, and Tat-Seng Chua. 2019. Explainable Reasoning over Knowledge Graphs for Recommendation. In *AAAI*. 5329–5336.
[78] Yanlin Wang and Hui Li. 2021. Code Completion by Modeling Flattened Abstract Syntax Trees as Graphs. In *AAAI*.
[79] Stanley Wasserman and Katherine Faust. 1994. *Social Network Analysis: Methods and Applications*. Cambridge University Press.

[80] Le Wu, Junwei Li, Peijie Sun, Richang Hong, Yong Ge, and Meng Wang. 2020. DiffNet++: A Neural Influence and Interest Diffusion Network for Social Recommendation. *IEEE Trans. Knowl. Data Eng.* (2020).
[81] Le Wu, Peijie Sun, Yanjie Fu, Richang Hong, Xiting Wang, and Meng Wang. 2019. A Neural Influence Diffusion Model for Social Recommendation. In *SIGIR*. 235–244.
[82] Qitian Wu, Hengrui Zhang, Xiaofeng Gao, Peng He, Paul Weng, Han Gao, and Guihai Chen. 2019. Dual Graph Attention Networks for Deep Latent Representation of Multifaceted Social Effects in Recommender Systems. In *WWW*. 2091–2102.
[83] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-Based Recommendation with Graph Neural Networks. In *AAAI*. 346–353.
[84] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2020. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Knowl. Data Eng.* (2020).
[85] Fengli Xu, Jianxun Lian, Zhenyu Han, Yong Li, Yujian Xu, and Xing Xie. 2019. Relation-Aware Graph Convolutional Networks for Agent-Initiated Social E-Commerce Recommendation. In *CIKM*. 529–538.
[86] Xiwang Yang, Yang Guo, Yong Liu, and Harald Steck. 2014. A survey of collaborative filtering based social recommender systems. *Comput. Commun.* 41 (2014), 1–10.
[87] Xiwang Yang, Harald Steck, and Yong Liu. 2012. Circle-based recommendation in online social networks. In *KDD*. 1267–1275.
[88] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *NeurIPS*. 4805–4815.
[89] Junliang Yu, Min Gao, Hongzhi Yin, Jundong Li, Chongming Gao, and Qinyong Wang. 2019. Generating Reliable Friends via Adversarial Training to Improve Social Recommendation. In *ICDM*. 768–777.
[90] Junliang Yu, Hongzhi Yin, Jundong Li, Min Gao, Zi Huang, and Lizhen Cui. 2020. Enhance Social Recommendation with Adversarial Graph Convolutional Networks. *IEEE Trans. Knowl. Data Eng.* (2020).
[91] Mengqi Zhang, Shu Wu, Meng Gao, Xin Jiang, Ke Xu, and Liang Wang. 2020. Personalized Graph Neural Networks with Attention Mechanism for Session-Aware Recommendation. *IEEE Trans. Knowl. Data Eng.* (2020).
[92] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Comput. Surv.* 52, 1 (2019), 5:1–5:38.
[93] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2018. Deep Learning on Graphs: A Survey. *IEEE Trans. Knowl. Data Eng.* (2018).
[94] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed H. Chi. 2019. Recommending what video to watch next: a multitask ranking system. In *RecSys*. 43–51.
[95] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph Neural Networks: A Review of Methods and Applications. *arXiv Preprint* (2018). https://arxiv.org/abs/1812.08434
[96] Weicheng Zhu and Narges Razavian. 2019. Graph Neural Network on Electronic Health Records for Predicting Alzheimer's Disease. *arXiv Preprint* (2019). https://arxiv.org/abs/1912.03761