Web Intelligence 19 (2021) 87–102
DOI 10.3233/WEB-210458
IOS Press

Graph-based knowledge tracing: Modeling student proficiency using graph neural networks

Hiromi Nakagawa[*], Yusuke Iwasawa and Yutaka Matsuo
Department of Engineering, The University of Tokyo, Tokyo, Japan
E-mails: nakagawa@weblab.t.u-tokyo.ac.jp, iwasawa@weblab.t.u-tokyo.ac.jp, matsuo@weblab.t.u-tokyo.ac.jp

Abstract. Recent advancements in computer-assisted learning systems have caused an increase in the research in *knowledge tracing*, wherein student performance is predicted over time. Student coursework can potentially be structured as a graph. Incorporating this graph-structured nature into a knowledge tracing model as a relational inductive bias can improve its performance; however, previous methods, such as deep knowledge tracing, did not consider such a latent graph structure. Inspired by the recent successes of graph neural networks (GNNs), we herein propose a GNN-based knowledge tracing method, i.e., graph-based knowledge tracing. Casting the knowledge structure as a graph enabled us to reformulate the knowledge tracing task as a time-series node-level classification problem in the GNN. As the knowledge graph structure is not explicitly provided in most cases, we propose various implementations of the graph structure. Empirical validations on two open datasets indicated that our method could potentially improve the prediction of student performance and demonstrated more interpretable predictions compared to those of the previous methods, without the requirement of any additional information.

Keywords: Knowledge tracing, graph neural network, educational data mining, learning sciences

1. Introduction

Recent advancements in computer-assisted learning systems have led to an increase in research on *knowledge tracing* [8], in which student performance in coursework exercises is predicted over time. Accurate predictions help students identify content suitable to their individual knowledge level, thereby facilitating more efficient learning. This is particularly important for e-learning platforms and teachers, as it helps increase student engagement and reduce dropout rates. Although various knowledge tracing methods have been proposed, Piech et al. [22] reported that a method called deep knowledge tracing (DKT), which leverages recurrent neural networks (RNNs) [31], performs significantly better than any previously reported method.

In terms of data structure, coursework can potentially be structured as a graph. The requirements for mastering the coursework are divided into knowledge concepts, known as nodes, and these concepts share dependency relationships, known as edges. Consider an example where coursework knowledge is divided into three concepts $v = \{v_1, v_2, v_3\}$. An understanding of v_1 is dependent on an understanding of v_2, and similarly, v_2 is dependent on v_3 (e.g., v_1, v_2, and v_3 represent *solving quadratic equations*, *solving linear equations*, and *transposition of a term*, respectively). Here, the concepts and their dependencies can be perceived as nodes and edges of a graph,

[*]Corresponding author. E-mail: nakagawa@weblab.t.u-tokyo.ac.jp.

respectively, where the edges are directed from v_3 to v_2 and from v_2 to v_1. Incorporating prior knowledge about the graph-structured nature of data into models improves their performance and interpretability [3]. Thus, incorporating the graph-structured nature of coursework knowledge can be effective for improving knowledge tracing models; however, previous deep learning-based methods, such as DKT, do not consider this nature. The architectures of previous deep-learning-based methods, such as RNN, generally perform well on sequential data, but cannot effectively handle graph-structured data.

To incorporate the graph-structured nature of coursework knowledge into knowledge tracing models, we reformulate knowledge tracing as a graph neural network (GNN) [12] application in this work. Recently, research on GNNs, which handle graph-structured data by deep learning, has been garnering attention. Although operation on irregular domain data has been challenging for existing machine-learning approaches, various generalization frameworks and important operations have been developed successfully [3,11,30] in various research areas. Battaglia et al. [3] explained the expressive power of GNNs from the perspective of relational inductive biases, which improve the sample efficiency of machine-learning models by incorporating prior human knowledge about the nature of data. To incorporate such benefits into knowledge tracing, we reformulate it as a GNN application and propose a novel model that can predict coursework proficiency by considering the latent knowledge structure.

A challenge encountered when performing knowledge tracing using a GNN is the definition of the latent graph structure. GNN has considerable expressive power for modeling graph-structured data; however, in several cases of knowledge tracing settings, the graph structure itself, i.e., the related concepts and the strength of the relationships, is not explicitly provided. It is possible for human experts to heuristically and manually annotate the content relationships. However, this requires deep domain knowledge and a substantial amount of time, so it is difficult to define the graph structure for all of the content in an e-learning platform in advance. We call this problem the *implicit graph structure problem*. A straightforward solution to this is to define the graph structure using simple statistics that can be automatically derived from data, such as the transition probability between answered concepts by students. Another solution is to learn the graph structure itself in parallel with the optimization of the primary task. A relevant topic in the recent GNN literature is edge-feature learning, for which several methods have been proposed. Although these techniques cannot be directly applied to our problem, they can be extended to allow their application in our case.

In this study, we propose a GNN-based knowledge tracing method, i.e., graph-based knowledge tracing (GKT). Our model reformulates knowledge tracing as a time-series node-level classification problem in the GNN. This formulation is based on three assumptions: 1) Coursework knowledge is decomposed into a certain number of knowledge concepts. 2) Students have their own temporal knowledge state, which represents their proficiency in the concepts of the coursework. 3) Coursework knowledge is structured as a graph, which affects the updating of the student knowledge state: if a student answers a concept, correctly or incorrectly, his/her knowledge state is affected, not only regarding the answered concept, but also other related concepts that are represented as neighboring nodes in the graph.

Using a subset of several open datasets of math exercise logs, we empirically validated our method. In terms of prediction performance, our model outperforms previous deep-learning-based methods, which indicates that our model possesses high potential to improve the prediction of student performance. In addition, by analyzing the prediction patterns of the trained model, student proficiency, i.e., the concepts which the students gained an understanding of, and the time required to do so, can be clearly interpreted from the model's predictions, whereas the previous methods demonstrated inferior interpretability. This means that our model provides predictions that are more interpretable compared to previous models. The obtained results verify the potential of our model to improve the performance and applicability of knowledge tracing for application in real educational environments.

Our contributions are as follows:

- We demonstrated that formulating knowledge tracing as an application of GNNs improves performance prediction, without requiring any additional information. Students can master the coursework more efficiently based on more precisely personalized content, while e-learning platforms can provide a higher quality of service to maintain high user engagement.
- Our model improves the interpretability of the model's predictions. Teachers and students can recognize the students' knowledge states more precisely, and the students can be more motivated to work on the exercises

by understanding why they are recommended. E-learning platforms and teachers can redesign the coursework curriculum more easily by understanding the points where students fail.

– To solve the implicit graph structure problem, we propose various implementations and empirically validate their effectiveness. With our results, researchers can benefit from a performance boost without requiring a costly manual annotation process to track the relationships between concepts. Educational experts can obtain new criteria to improve curriculum design that can help in finding a good knowledge structure.

2. Related work

2.1. Knowledge tracing

Knowledge tracing [8] is the task of predicting student performance over time based on coursework exercises, and can be formulated as follows:

$$\mathbf{y}^t = KT(\mathbf{x}^1, \dots, \mathbf{x}^t),$$

where $\mathbf{x}^t = \{q^t, r^t\}$ is a tuple that considers an answered exercise q^t and whether the exercise is answered correctly r^t at time step t; $\mathbf{y}^t$ is the probability of the student answering each exercise correctly at the next time step $t + 1$, and KT is the knowledge tracing model.

2.1.1. DKT

Since Piech et al. [22] first proposed a deep-learning-based knowledge tracing method, i.e., DKT, and demonstrated the considerable expressive power of RNNs, many researchers have adopted RNN or its extensions as the KT. These models define a hidden state, or a student's temporal knowledge state, $\mathbf{h}^t$, and update it recurrently based on the student's exercise performance over time. RNN-based models must represent $\mathbf{x}$ as a fixed-length vector, and in many cases, $\mathbf{x}^t$ is represented by concatenating two binary vectors that represent which exercises were answered correctly and which were answered incorrectly. Thus, for datasets with N unique exercises, $\mathbf{x}^t \in \{0, 1\}^{2N}$. Table 1 shows an example of the answer logs and input vectors when $N = 3$. The output vector $\mathbf{y}^t$ has the same length as the number of exercises, where each element represents the predicted probability that the student would answer that particular exercise correctly. The training objective is to minimize the negative log-likelihood of the observed sequence of student responses under the model.

2.1.2. Dynamic key-value memory network

DKT is a simple RNN-based model that has demonstrated a remarkable performance compared to traditional methods [8]. However, DKT represents the temporal knowledge state $\mathbf{h}^t$ using a single hidden vector, thus complicating the modeling of the state for each concept separately. This results in performance degradation for long time sequences and reduces interpretability of how the model predicted the mastery of each concept. Zhang et al. [33] proposed a dynamic key-value memory network (DKVMN) that uses two memory matrices to address these shortcomings. DKVMN defines a *value* matrix that can be regarded as a stack of temporal knowledge states $\mathbf{h}^t$, defined separately for each concept. First, DKVMN computes the dot-product attention between the embedding of $\mathbf{x}^t$ and the *key* matrix; this can be considered as representing the relativity between the answered concept and other concepts. Based on the attention score, DKVMN updates the knowledge state for the related concepts and predicts student performance. However, DKVMN adopted a single dot-product attention mechanism to estimate the

Table 1
Example of DKT input

Answer log				Input vector
Student ID	Time	Concept ID	Response	
A	1	1	Incorrect	$\mathbf{x}^1 = [000 : 100]$
A	2	1	Correct	$\mathbf{x}^2 = [100 : 000]$
A	3	2	Correct	$\mathbf{x}^3 = [010 : 000]$

relativity between the concepts, and this complicated the modeling of complex and multiple relationships between the concepts. We address this problem by modeling the relation weights, or edge weights, between input concepts, using K different neural networks for K edge types. We explain this in detail in Section 3.4.

2.1.3. *Knowledge structure in knowledge tracing*

As mentioned previously, coursework can potentially be structured as a graph. It is known that incorporating prior knowledge about the graph-structured nature of data into models improves their performance and interpretability [3]. Thus, incorporating the graph-structured nature of coursework knowledge can be effective in improving knowledge tracing models; however, previous deep-learning-based methods, such as DKT, do not consider this nature.

One of the few studies which use knowledge structure for deep-learning-based adaptive learning is Liu et al. [20]. They first estimated student knowledge levels using DKT, then applied reinforcement learning algorithms to identify the appropriate learning items, reducing the search space of recommended items based on the predefined knowledge structure. However, they do not incorporate the knowledge structure information into the former knowledge tracing stage, and they use a plain DKT model, failing to use fully the latent knowledge structure for student proficiency modeling. Another problem is that they take the human-predefined knowledge structure information [4] as given. Manually annotating the content relationships requires deep domain knowledge and a substantial amount of time, and it is difficult to define the structure in advance for all the coursework. Thus, in many cases of real-world knowledge tracing settings, the graph structure is not explicitly provided, which hinders the application of frameworks like [20].

2.2. *GNN*

GNNs [12] are a type of neural networks that can operate on graph-structured data that represents objects and their relationships as nodes and edges, respectively. Although operating on such irregular domain data has been challenging for existing machine-learning approaches, the considerable expressive power of graphs has led to increasing research on GNNs. In recent years, various generalization frameworks and important operations have been developed [3,11,30] with successful results in various research areas, such as social science [13,18] and natural science [2,10,23], and in areas relevant to this work, like time-series recommendation [32].

Our primary motivation for using GNN is the success of convolutional neural networks (CNNs) [19]. Using a local connection, weight sharing, and multilayer architecture, a CNN can extract multiscale local spatial features and use them to construct expressive representations as it has resulted in breakthroughs in various research areas such as computer vision. However, CNNs can only operate on regular Euclidean data, such as images and text, whereas applications in the real world often generate non-Euclidean data. A GNN, on the other hand, regards these non-Euclidean data structures as graphs and enables the same advantages of the CNN to be reflected on these highly diverse data. Battaglia et al. [3] explained the expressive power of the GNN and CNN from the perspective of relational inductive biases, which improve the sample efficiency of machine-learning models by incorporating prior human knowledge about the nature of data.

2.2.1. *Edge feature learning*

Among the research topics in GNNs, edge feature learning [2,5,11] is the most relevant to our work. Graph attention networks [28] apply the multi-head attention mechanism [27] to the GNN and enable learning edge weights during training without requiring their predefinition. Neural relational inference [17] leverages a variational autoencoder (VAE) [16] to learn the latent graph structure in an unsupervised manner. Our method assumes a latent graph structure underlying the knowledge concepts of coursework and models the temporal transitions of student proficiency in each concept using graph operators. However, the graph structure itself is not explicitly provided in many cases. We address this problem by designing models that learn the edge connection itself in parallel with the optimization of the student performance prediction by extending these edge feature learning mechanisms. We explain this in detail in Section 3.3.

3. Methods

3.1. Problem definition

Here, we assume that the coursework is potentially structured as a graph $G = (V, E)$; the requirements for mastering the coursework are decomposed into N knowledge concepts, known as nodes $V = \{v_1, \dots, v_N\}$, and these concepts share dependency relationships, known as edges $E \subseteq V \times V$. In addition, we assume a student having a temporal knowledge state for each concept independently at time step t, $\mathbf{h}^t = \{\mathbf{h}^t_{i \in V}\}$, and this knowledge state is updated over time as follows: when the student solves an exercise associated with concept v_i, then the student's knowledge state for the answered concept $\mathbf{h}^t_i$ and its related concepts $\mathbf{h}^t_{j \in \mathcal{N}_i}$ is updated. Here, $\mathcal{N}_i$ denotes a set of nodes neighboring v_i.

3.2. Proposed method

GKT applies a GNN to the knowledge tracing task and leverages the graph-structured nature of knowledge. We present the architecture of GKT in Fig. 1. The following paragraphs explain the processes in detail.

3.2.1. Aggregate

First, the model aggregates the hidden states and embeddings for the answered concept i and its neighboring concepts $j \in \mathcal{N}_i$:

$$\mathbf{h}'^t_k = \begin{cases} [\mathbf{h}^t_k, \mathbf{x}^t \mathbf{E_x}] & (k = i) \\ [\mathbf{h}^t_k, \mathbf{E}_c(k)] & (k \neq i), \end{cases}$$

where $\mathbf{x}^t \in \{0, 1\}^{2N}$ is an input vector that represents the exercises answered correctly and incorrectly at time step t; $\mathbf{E_x} \in \mathbb{R}^{2N \times e}$ is a matrix embedding the concept index and the correctness of the answers; $\mathbf{E_c} \in \mathbb{R}^{N \times e}$ is a matrix embedding the concept index; $\mathbf{E_c}(k)$ represents the k-th row of $\mathbf{E_c}$, and e is the embedding size.

3.2.2. Update

Next, the model updates the hidden states based on the aggregated features and the knowledge graph structure:

$$\mathbf{m}^{t+1}_k = \begin{cases} f_{\text{self}}(\mathbf{h}'^t_k) & (k = i) \\ f_{\text{neighbor}}(\mathbf{h}'^t_i, \mathbf{h}'^t_k) & (k \neq i) \end{cases} \tag{1}$$

$$\tilde{\mathbf{m}}^{t+1}_k = \mathcal{G}_{ea}(\mathbf{m}^{t+1}_k)$$

$$\mathbf{h}^{t+1}_k = \mathcal{G}_{gru}(\tilde{\mathbf{m}}^{t+1}_k, \mathbf{h}^t_k)$$

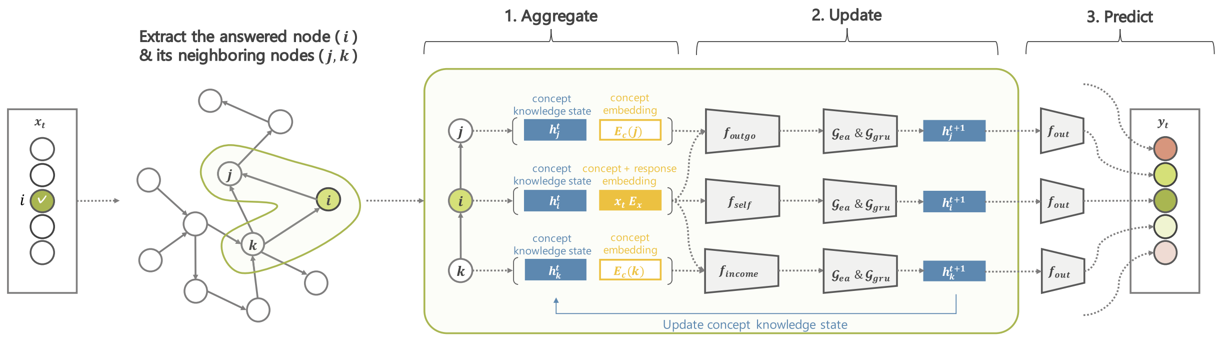


Fig. 1. Architecture of GKT. When a student answers a concept, GKT first aggregates the node features related to the answered concept, then updates the student's knowledge states for the related concepts, and finally predicts the probability of the student answering each concept correctly at the next time step.

where f_{self} is a multilayer perceptron (MLP), $\mathcal{G}_{ea}$ is an erase–add gate used in Zhang et al. [33], and $\mathcal{G}_{gru}$ is a gated recurrent unit [6]. f_{neighbor} is an arbitrary function that defines information propagation to neighboring nodes based on a knowledge graph. We propose various implementations of f_{neighbor} in Section 3.3.

3.2.3. *Predict*

Finally, the model outputs the predicted probability of a student answering each concept correctly at the next time step:

$$\mathbf{y}_k^t = \sigma\left(\mathbf{W}_{\text{out}}\mathbf{h}_k^{t+1} + \mathbf{b}_k\right),$$

where $\mathbf{W}_{\text{out}}$ is a weight matrix common for all nodes; $\mathbf{b}_k$ is a bias term for node k, and σ is a sigmoid function. The model is trained to minimize the negative log-likelihood of the observations.

We can use the edge information to collect the knowledge state from the neighboring concepts. We confirmed that it was better to predict $\mathbf{y}_k^t$ based only on the knowledge state of the target concept $\mathbf{h}_k^t$; therefore, we limited the use of graph-structure information to the updating phase only.

3.3. *Implementation of latent graph structure and $f_{neighbor}$*

GKT can leverage the graph-structured nature of knowledge for the purpose of knowledge tracing; however, the structure itself is not explicitly provided in most cases. To implement the latent graph structure and f_{neighbor} in equation (1), we introduce two approaches.

3.3.1. *Statistics-based approach*

The statistics-based approach implements the adjacency matrix $\mathbf{A}$ based on certain statistics and applies it to f_{neighbor} as follows:

$$f_{\text{neighbor}}\left(\mathbf{h}'^t_i, \mathbf{h}'^t_j\right) = \mathbf{A}_{i,j} f_{\text{out}}\left(\left[\mathbf{h}'^t_i, \mathbf{h}'^t_j\right]\right) + \mathbf{A}_{j,i} f_{\text{in}}\left(\left[\mathbf{h}'^t_i, \mathbf{h}'^t_j\right]\right), \tag{2}$$

where f_{out} and f_{in} are MLPs. Here, we introduce three types of graphs.

Dense graph is a simple densely connected graph, where $\mathbf{A}$ is defined as follows:

$$\mathbf{A}_{i,j} = \begin{cases} 0 & (i = j) \\ \frac{1}{|V|-1} & (i \neq j) \end{cases}$$

Transition graph is a transition probability matrix-based graph, where $\mathbf{A}$ is defined as follows:

$$\mathbf{A}_{i,j} = \begin{cases} 0 & (i = j) \\ \frac{n_{i,j}}{\sum_k n_{i,k}} & (i \neq j) \end{cases}$$

Here, $n_{i,j}$ represents the number of times concept j was answered immediately after concept i was answered.

DKT graph is a graph generated based on the conditional prediction probability of the trained DKT model, as proposed by Piech et al. [22]. For each directed pair (i, j) between exercise i and j, where j appears more than $N\%$ in the remainder of the sequence after i appears, an influence matrix J is defined by the following equation:

$$J_{i,j} = \frac{y(j|i)}{\sum_k y(j|k)},$$

where $y(j|i)$ is the predicted probability of a student answering concept j correctly the next time after the student answered concept i correctly for the first time. The graph is constructed by drawing an edge between concepts i and j, such that $J_{i,j}$ is higher than a given threshold θ. $J_{i,j}$ can be considered to represent how easily a student can acquire concept j when the student has already acquired concept i. We set $N = 1$ and $\theta = 0.01$ following Piech et al. [22], draw edges from the top three nodes with high influence to each node, and define $\mathbf{A}$ as follows:

$$\mathbf{A}_{i,j} = \begin{cases} 0 & (i = j) \\ J_{i,j} & (i \neq j) \end{cases}$$

3.3.2. Learning-based approach

In this approach, the graph structure is learned in parallel with the optimization of the performance prediction. Here, we introduce three approaches for learning the graph structure.

Parametric adjacency matrix (PAM) simply parameterizes the adjacency matrix $\mathbf{A}$ and optimizes it with the other parameters under certain constraints, such that $\mathbf{A}$ satisfies the property of an adjacency matrix. f_{neighbor} is defined as in equation (2).

Multi-head attention (MHA) leverages the multi-head attention mechanism [27] to infer the edge weights between two nodes based on their features. f_{neighbor} is defined as follows:

$$f_{\text{neighbor}}\left(\mathbf{h}'^{t}_{i}, \mathbf{h}'^{t}_{j}\right) = \frac{1}{K} \sum_{k \in K} \alpha^{k}_{ij} f_k\left(\mathbf{h}'^{t}_{i}, \mathbf{h}'^{t}_{j}\right),$$

where k is the head index among a total of K heads; α^{k}_{ij} is the k-th head's attention weight from v_i to v_j, and f_k is a neural network for the k-th head.

Variational autoencoder assumes the discrete latent variable that represents the types of edges and infers them based on node features. f_{neighbor} is defined as follows:

$$f_{\text{neighbor}}\left(\mathbf{h}'^{t}_{i}, \mathbf{h}'^{t}_{j}\right) = \sum_{k \in K} z^{k}_{ij} f_k\left(\mathbf{h}'^{t}_{i}, \mathbf{h}'^{t}_{j}\right),$$

where k is the edge type among a total of K types; z^{k}_{ij} is a latent variable sampled from the Gumbel–Softmax distribution [21], and f_k is a neural network for the k-th edge type. VAE minimizes the negative log-likelihood and the Kullback–Leibler divergence between the encoded distribution $q(\mathbf{z}|\mathbf{x})$ and prior distribution $p(\mathbf{z})$. Using one edge type to represent the "non-edge" class implies that no messages are passed along this edge type; additionally, setting a high probability on the "non-edge" label encourages the generation of a sparse graph.

The learning-based approach is close to the concept of edge feature learning [2,5,11], and MHA and VAE were motivated by the graph attention network [28] and neural relational inference [17], respectively; however, we modify them based on two aspects. First, we calculate the edge weights based on static features, such as the embeddings of concepts and responses, instead of dynamic ones. This enables the learning of the knowledge graph structure invariant to the students and time steps, which is more natural considering the actual knowledge tracing settings. Second, for the VAE, we limit the edge-type inference for nodes related to the answers at each time step. This fits the knowledge tracing situation wherein students answer only a small subset of concepts at each time step, thereby reducing the computational cost from $\mathcal{O}(KN^2)$ for the original neural relational inference to $\mathcal{O}(KN)$.

3.4. Comparison with previous methods

A comparison of the proposed method with the previous ones can be performed using two criteria. We present the comparison in Fig. 2.

The first criterion is the definition of the student temporal knowledge state, $\mathbf{h}^t$. In DKT, $\mathbf{h}^t$ is represented as a single hidden vector, and the knowledge states for each concept are not separated. This complicates the modeling of the knowledge state for each concept separately and results in performance degradation in long-time sequences and low interpretability of how the model predicted the student proficiency in each concept. To address these shortcomings, Zhang et al. [33] proposed DKVMN, which uses two memory matrices, one of which can be regarded as a stack of the student temporal knowledge states $\mathbf{h}^t$, defined separately for each concept. Although this is almost the same as GKT, they are slightly different because GKT directly models the knowledge state for each concept, whereas DKVMN defines another low-dimensional latent concept and then models its knowledge state.

The other criterion involves the interactions between concepts during knowledge state updating. In a DKVMN, the relation weights between the original input concepts and latent concepts are calculated using a simple dot-product attention mechanism, which can be insufficient for modeling the complex and multiple relationships between knowledge concepts. Meanwhile, GKT models the relation weights, or the edge weights, between input concepts using K different neural networks for K edge types. This enables the modeling of multiple and complex relationships between concepts.

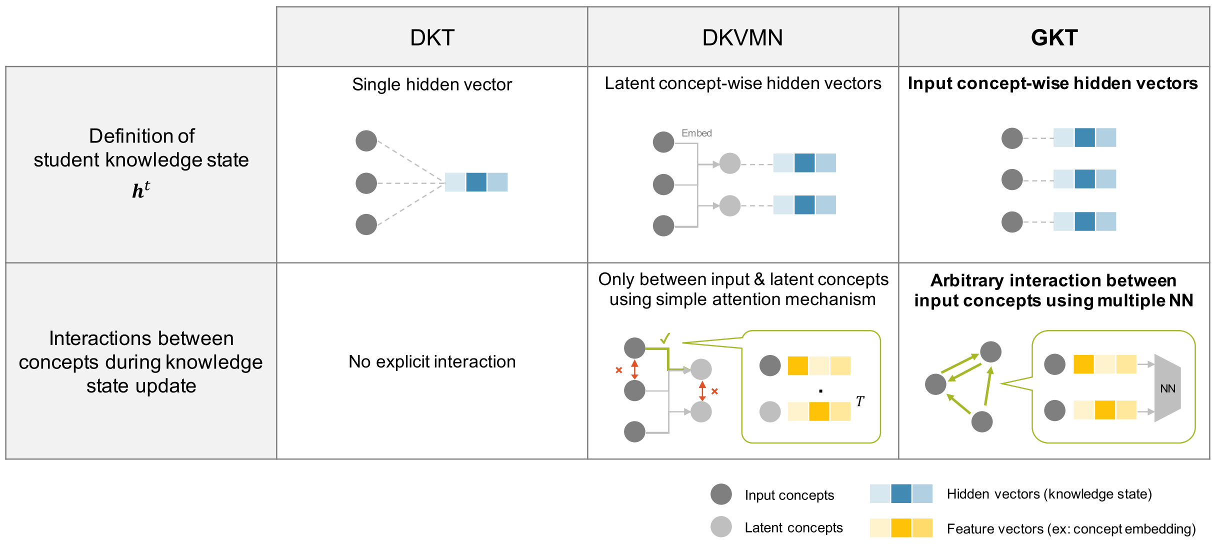

DKT
DKVMN
GKT
Definition of student knowledge state h^t
Single hidden vector
Latent concept-wise hidden vectors
Embed
Input concept-wise hidden vectors
Interactions between concepts during knowledge state update
No explicit interaction
Only between input & latent concepts using simple attention mechanism
T
Arbitrary interaction between input concepts using multiple NN
NN
Input concepts
Latent concepts
Hidden vectors (knowledge state)
Feature vectors (ex: concept embedding)

Fig. 2. Comparison of GKT with previous methods.

4. Experiments

4.1. Datasets

For the experiments, we used eight open datasets of student math exercise logs. We provide overviews of each dataset below. The parenthesis after the dataset names contain abbreviations that will be used in the tables of our experimental results.

4.1.1. ASSISTments

ASSISTments[1] is an online educational service. It provides various math student exercise log datasets, which are used in many knowledge tracing studies as benchmark datasets. We used the following four datasets [9] for our experiments.

– **ASSISTments2009 (As'09)**: Dataset from the "skill-builder" subset of 2009–2010 ASSISTment Data.
– **ASSISTments2012 (As'12)**: Dataset from 2012–13 School Data with Affect.
– **ASSISTments2015 (As'15)**: Dataset from 2015 ASSISTments Skill Builder Data.
– **ASSISTments2017 (As'17)**: Dataset from ASSISTments Data Mining Competition 2017.

4.1.2. KDDCup

The datasets used in the KDD Cup 2010 Educational Data Mining Challenge are also used as benchmark datasets for knowledge tracing. We used the following three development datasets for our experiments.

– **Algebra I 2005–2006 (Al'05)** [24]
– **Algebra I 2006–2007 (Al'06)** [25]
– **Bridge to Algebra 2006–2007 (Br'06)** [26]

4.1.3. Junyi academy

Junyi Academy[2] is an e-learning platform in Taiwan, from which Chang, Hsu and Chen [4] collected a student math exercise log dataset. We used this **Junyi Academy (Junyi)** dataset for our experiments.

[1] https://sites.google.com/site/assistmentsdata/home/assistment-2009-2010-data/skill-builder-data-2009-2010
[2] https://www.junyiacademy.org/

Table 2
Concept tags

Dataset	Concept name
As'09	Absolute value
	Greatest common factor
	Proportion
Br'06	Calculate area of overlap
	Identify GCF
	Measure acute angle

Table 3
Dataset statistics

Dataset	Statistics		
	# students	# concepts	# logs
As'09	959	102	65,987
As'12	969	167	79,773
As'15	979	99	37,097
As'17	1,000	92	548,559
Al'05	574	110	606,728
Al'06	994	419	1,276,747
Br'06	1,000	451	1,517,223
Junyi	827	465	110,593

4.2. *Dataset preprocessing*

All of the datasets have the students' time-series exercise logs, in which each exercise is associated with one or more human-predefined knowledge concept tags that are required to solve the exercise. We provide examples of the knowledge concept tags in Table 2. In the following, we describe the preprocessing procedures for these datasets. We show the statistics of the datasets after the preprocessing in Table 3.

4.2.1. *Remove invalid concept tags*

Some datasets have invalid concept tags that are noise for experiments, for example, unnamed tags or tags containing "DO NOT TRACK ME" strings in the KDD Cup datasets. We exclude these tags from our analysis.

4.2.2. *Combine simultaneous answer logs*

Because some datasets have multiple concept tags for one exercise, answers for each tag can be recorded separately, even if those tags are answered at a single time step. Using these records directly could cause unfairly high prediction performance because of the frequently co-occurring tags. Thus, we combine simultaneous answer logs into a single time-step answer vector or tensor, which is input into the knowledge tracing models.

4.2.3. *Randomly sample users (optional)*

Because some datasets have many logs, we randomly sample up to 1,000 users' logs for analysis to perform exhaustive validations efficiently. However, this process can hinder the accurate validation of our method, so we also performed experiments using all users' logs for some datasets in Section 4.5.2 and ensured that this user sampling has little impact on our analysis.

4.2.4. *Threshold concept tags based on the number of answers*

Because some concept tags have few user answer logs, we excluded such tags to remove noise. We extracted only the logs associated with concept tags answered at least 10 times.

4.3. *Implementation details*

For each dataset, we applied a user-wise five-fold split, trained and evaluated the models' performance for each split, and took the average of the five-fold results. In each split, we took 10% of the training data as validation data and trained the model for up to 50 epochs while adjusting the hyperparameters using the validation data. We used an Adam [15] optimizer with a learning rate of 0.001 for all models. In each iteration of training, we randomly sampled a partial sequence with a length of 100 from the whole sequence of each student to enable efficient computation and large batch sizes. In testing, we used the whole sequence for each student.

DKT We set the hyperparameters following Piech et al. [22]. The size of the hidden layer was 200, and we used a gated recurrent unit for the RNN. We applied a dropout from $\mathbf{h}^t$ to $\mathbf{y}^t$ with a drop rate of 0.5. The batch size was 32.

DKVMN We set the hyperparameters following Zhang et al. [33]. The size of the memory slot was 20 and the size of the hidden vectors was 50. The batch size was 32.

GKT The size of all the hidden vectors and the embedding matrix was 32. For the MLPs in the model, we followed [17]. We used a two-layer MLP with an exponential linear unit [7] activation function, applied a dropout from the hidden vectors to the output vectors with a drop rate of 0.5, and applied batch normalization [14] to the output layers. The batch size was 16. We set $K = 2$ in the MHA and the VAE for a fair comparison of the learning-based approaches with the statistics-based ones, as the latter assumed two edge types, that is, incoming edges and outgoing edges.

4.4. *Prediction performance*

First, we evaluated the prediction performance of GKT. For evaluation metrics, we used the area under the curve (AUC) score, which is commonly used in previous studies [22,33]. It is calculated by comparing the predicted probability of a student answering each exercise correctly and his/her actual response, and reflects a model's ability to discriminate correct responses from incorrect ones. We list the results in Table 4. The highest scores are denoted in bold for each dataset, and their relative improvements from the baselines' best scores are also shown. In all datasets, GKT achieved the highest AUC score, and its relative improvement was 6.25% at best. This suggests that GKT can trace the student knowledge state better than the previous deep-learning-based methods, which do not sufficiently consider the knowledge graph structure.

Focusing on the definition of the graph structure of GKT, the statistics-based approaches and PAM in the learning-based approaches showed better performance on average; however, the performance difference between other graph-structure-based GKTs was minor. We discuss this in Section 5.2

4.5. *Ablation study*

4.5.1. *Impact of the model size*

To determine the model size, we referred to the original papers for the baselines and set both the hidden layer size and the embedding size as 32 for GKT. Using these settings, GKT showed better performance than the baselines.

Table 4

Prediction performance

Method		AUC							
		As'09	As'12	As'15	As'17	Al'05	Al'06	Br'06	Junyi
Baseline	DKT	0.719 ± 0.009	0.692 ± 0.012	0.666 ± 0.007	0.673 ± 0.004	0.795 ± 0.009	0.779 ± 0.004	0.772 ± 0.004	0.710 ± 0.029
	DKVMN	0.689 ± 0.011	0.647 ± 0.010	0.642 ± 0.011	0.649 ± 0.003	0.778 ± 0.007	0.758 ± 0.002	0.752 ± 0.004	0.724 ± 0.023
Statistics	Dense	$\mathbf{0.726} \pm 0.011$	$\mathbf{0.701} \pm 0.014$	0.678 ± 0.006	0.696 ± 0.003	0.796 ± 0.008	0.777 ± 0.002	0.783 ± 0.004	0.750 ± 0.020
	Transition	0.725 ± 0.012	$\mathbf{0.701} \pm 0.014$	$\mathbf{0.679} \pm 0.006$	0.697 ± 0.003	0.796 ± 0.008	0.777 ± 0.003	0.783 ± 0.004	0.751 ± 0.021
	DKT	$\mathbf{0.726} \pm 0.011$	$\mathbf{0.701} \pm 0.014$	0.678 ± 0.006	0.696 ± 0.003	0.796 ± 0.008	0.773 ± 0.003	0.779 ± 0.001	$\mathbf{0.752} \pm 0.022$
Learning	PAM	0.719 ± 0.012	0.688 ± 0.012	0.662 ± 0.004	$\mathbf{0.698} \pm 0.003$	$\mathbf{0.799} \pm 0.008$	$\mathbf{0.783} \pm 0.003$	$\mathbf{0.787} \pm 0.004$	0.739 ± 0.020
	MHA	0.725 ± 0.014	0.695 ± 0.014	0.672 ± 0.007	0.696 ± 0.004	0.798 ± 0.007	0.782 ± 0.003	0.785 ± 0.004	0.746 ± 0.022
	VAE	0.725 ± 0.011	0.698 ± 0.011	0.677 ± 0.004	0.696 ± 0.002	0.798 ± 0.008	0.781 ± 0.003	0.781 ± 0.007	0.745 ± 0.021
Relative Improvement		1.25%	1.31%	2.03%	3.84%	0.58%	0.52%	1.94%	6.25%

Table 5

Prediction performance using different model sizes

Method	AUC and # params						
	As'09	As'12	As'15	As'17	Al'05	Al'06	Br'06
DKT (small)	0.716 ± 0.012	0.685 ± 0.014	0.663 ± 0.008	0.657 ± 0.005	0.777 ± 0.009	0.755 ± 0.004	0.754 ± 0.003
	$33K$	$52K$	$32K$	$30K$	$36K$	$125K$	$134K$
GKT (small)	**0.726** ± 0.011	**0.701** ± 0.014	**0.678** ± 0.006	**0.696** ± 0.003	**0.796** ± 0.008	**0.777** ± 0.002	**0.783** ± 0.004
	$36K$	$42K$	$35K$	$35K$	$36K$	$66K$	$70K$
Relative Improvement	1.57%	2.36%	2.48%	6.25%	2.87%	3.75%	4.32%
DKT (large)	0.719 ± 0.009	0.692 ± 0.012	0.666 ± 0.007	0.673 ± 0.004	0.795 ± 0.009	0.779 ± 0.004	0.772 ± 0.004
	$345K$	$462K$	$339K$	$327K$	$359K$	$916K$	$973K$
GKT (large)	**0.720** ± 0.011	**.693** ± 0.014	**0.673** ± 0.007	**0.713** ± 0.003	**0.806** ± 0.009	**0.789** ± 0.003	**0.793** ± 0.005
	$451K$	$476K$	$450K$	$447K$	$454K$	$573K$	$586K$
Relative Improvement	0.12%	0.11%	1.00%	6.05%	1.40%	1.35%	2.70%

Table 6

Prediction performance on full data

Method	AUC						
	As'09	As'12	As'15	As'17	Al'05	Al'06	Br'06
DKT	0.745 ± 0.003	0.728 ± 0.001	0.713 ± 0.001	0.667 ± 0.004	0.778 ± 0.008	0.764 ± 0.005	0.758 ± 0.002
GKT	**0.748** ± 0.004	**0.732** ± 0.001	**0.714** ± 0.002	**0.707** ± 0.004	**0.796** ± 0.008	**0.780** ± 0.004	**0.783** ± 0.002
Relative Improvement	0.50%	0.54%	0.17%	5.93%	2.36%	2.14%	3.22%

However, we found that the number of parameters of GKT with the above hyperparameter is nearly 10 times smaller than that of baseline DKT. To further investigate this gap in the model size and its effect on the models' performance, we performed additional experiments. First, for DKT, which required a large number of parameters with the original settings (hereinafter called "DKT (large)"), we experimented with a smaller version with a hidden layer size of 32 (hereinafter called "DKT (small)"). Then, for GKT, which required a small number of parameters with the original settings (hereinafter called "GKT (small)"), we experimented with a larger version with the hidden layer size and the embedding size set to 128 (hereinafter called "GKT (large)"). For simplicity, we selected the statistics-based approach with the dense graph as the GKT model, which is the simplest variant of GKT. The small and large version pairs have almost the same number of parameters to each other in most of the datasets; thus, we can align the model size and compare their performance.

We show the results in Table 5. The numbers below the AUC scores represent the number of trainable parameters, where the numbers are rounded to thousands. In both the small and the large versions, GKT showed better performance than DKT. Furthermore, "GKT (small)" also outperformed "DKT (large)," as shown in Section 4.4. These results indicate that our method can achieve better performance with fewer parameters. This can be partly explained by the efficient hidden representation of GKT that handles the hidden states of each concept separately, whereas DKT needs to represent hidden states for all the concepts in single hidden states.

4.5.2. *Impact of the number of users*

As described in Section 4.2.3, for efficient experiments, we randomly sampled up to 1,000 users' logs for analysis in each dataset. However, this process can hinder accurate validation of our method, so we also performed experiments using all users' logs for some datasets. We used "DKT (small)" and "GKT (small)" mentioned in Section 4.5.1 as benchmarks. As shown in Table 6, in all the datasets, our model also outperformed baseline models in the full data as well as in the user-sampled data. This ensures that the user sampling has little relation to the performance improvement of our method, and therefore the validity of the experiments in Section 4.4 and 4.5.1 can be confirmed.

4.6. *Interpretability of the prediction*

Next, we visualize how the model predicts the student knowledge state to change over time and evaluate the interpretability of its prediction. The visualization helps students and teachers recognize the former's knowledge state efficiently and intuitively, and consequently its interpretability is of importance. Here, we evaluate interpretability based on the following two points: 1) Whether the model updates only the concepts related to the answered concepts at each time step. 2) Whether the update is reasonable with the given graph structure. Although previous works [22,33] performed a similar analysis, our study extends this approach as follows, to analyze the temporal transitions more precisely:

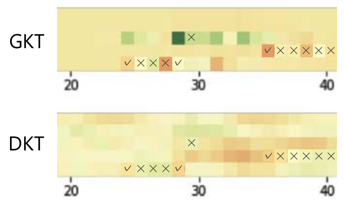
1. Randomly sample a student log until time step T.
2. Remove bias vectors from the output layers in a trained model.
3. Input the student answer logs $\mathbf{x}_{t\leqslant T}$ to the trained model and stack the output $\mathbf{y}_{t\leqslant T}$.
4. Normalize the output values at each time step from 0 to 1.

The objective of removing the bias vectors and normalizing the values at each time step is to remove the concept-specific and time-step-specific biases, so that the knowledge state transitions can be clearly visualized as a heatmap.

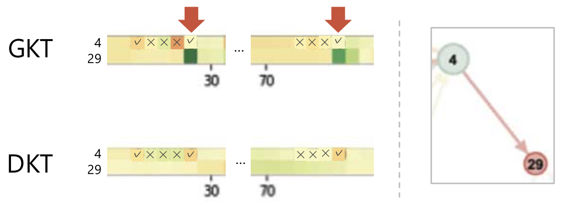
Figures 3(a) and 3(b) depict a randomly sampled student's knowledge-state transition to a subset of concepts in ASSISTments2009. The x-axis and y-axis show the time steps and concept indices, respectively, and the cell color shows the extent to which the proficiency level changed at each time step. Green denotes an increase and red denotes a decrease. We fill the elements answered correctly and incorrectly with "✓" and "×," respectively. As shown in Fig. 3(a), GKT updates the knowledge state of only the related concepts, whereas DKT updates the state of all the concepts indistinctly and cannot model the change in the related concepts. In addition, Fig. 3(b) shows that although concept 29 is not answered, its knowledge state is clearly updated at $t = 28$ and $t = 75$. At these time steps, concept 4 is answered correctly, and the given graph shows an edge between concepts 4 and 29, as shown on the right-hand side of the figure. This suggests that GKT definitively models the student knowledge state based on the given graph. However, DKT does not exhibit this behavior. These results indicate that GKT can model the student proficiency level for each concept distinctively and reasonably and provide more interpretable predictions.

4.7. *Network analysis*

Finally, we extracted the learned graph structure from the trained GKT model and analyzed it. In the learning-based approaches, GKT learns the graph structure that helps predict student performance. Thus, analyzing and



(a) Subset of frequently answered concepts



(b) Subset of concepts neighboring the graph

Fig. 3. Transitions of predicted knowledge states for concept subsets.

comparing the graphs extracted from the models can provide insights into how the models learn the relationships between concepts.

The process of extracting the graph structure from each of the leaning-based models is as follows:

1. For a given dataset, train the models on each of the five-split folds.
2. Create the adjacency matrix from each of the five models.
3. Extract edge connections common in at least n of the five models.

We used the "Bridge to Algebra 2006–2007" dataset as an example. Extracting the edge connections common in n models enables us to remove noise specific to each split and obtain the structure learned by multiple models in common. We set $n = 4$.

The networks are depicted in Fig. 4, where the top of the figure shows the network overview, and the bottom of the figure shows the local connections. We used Gephi [1] for network visualization. The color of the nodes is graded from blue to red, where the earlier an exercise is answered, the bluer the shade. The size of the nodes is proportional to their out-degree, implying that larger nodes affect more other nodes.

First, we focus on the global structure of the graphs. In the PAM graph, the nodes are forming small clusters, and they are split from other clusters. In the MHA graph, although the nodes are forming clusters like the PAM graph, they are sparsely connected with other clusters. In the VAE graph, although most of the nodes are forming small clusters, some nodes are forming a large cluster, where the nodes are densely connected with each other. Although each model shows the different structure influenced by their model architecture, the difference in prediction performance was small as shown in Section 4.4. Thus, we cannot determine which of the structure better represents the student knowledge structure.

On the other hand, focusing on the local connections, we found that some similar connections on related concepts appear in all the three models, as shown in the bottom of the figure. This indicates the potential of the GKT to learn important concept relationships purely from students' exercise logs, independent of the detailed architecture of learning the graph structure.

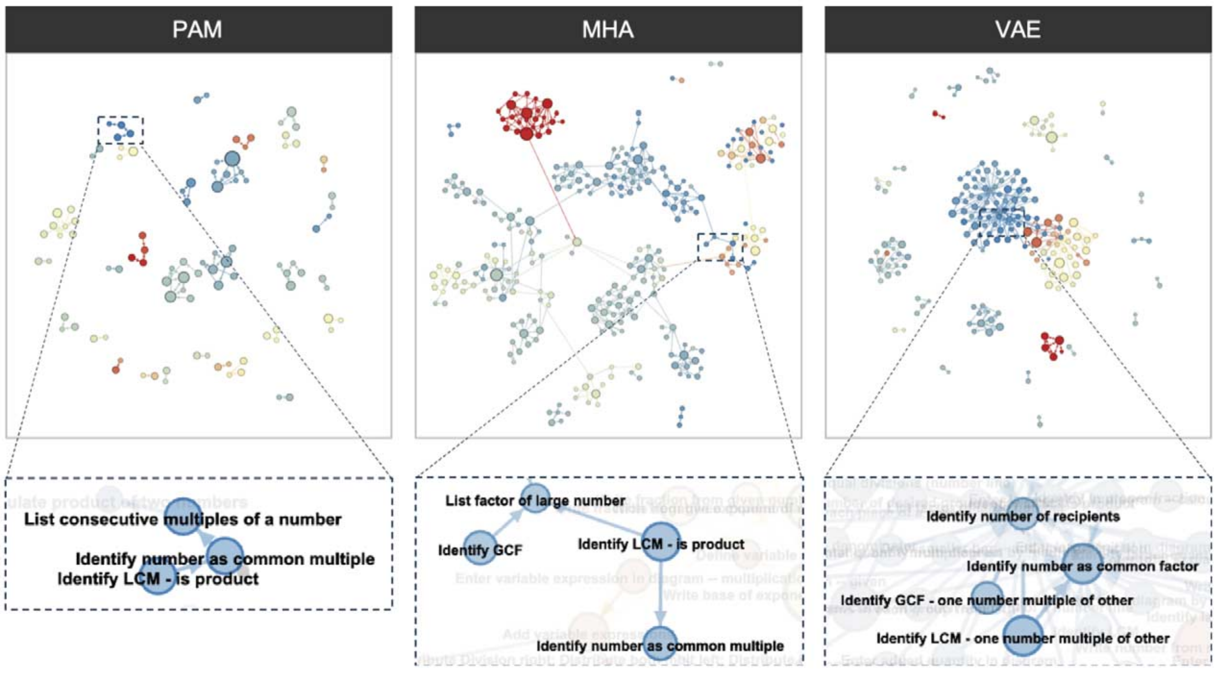

PAM
MHA
VAE
List consecutive multiples of a number
Identify number as common multiple
Identify LCM - is product
List factor of large number
Identify GCF
Identify LCM - is product
Identify number as common multiple
Identify number of recipients
Identify number as common factor
Identify GCF - one number multiple of other
Identify LCM - one number multiple of other

Fig. 4. Network visualizations of graphs derived from GKT models.

5. Discussion

5.1. Benefits of GNN-based reformulation of knowledge tracing

In this paper, we proposed a GNN-based knowledge tracing method, and empirically validated that our method improves the prediction of student proficiency (Table 4–6) and its interpretability (Fig. 3) compared with previous deep-learning-based methods [22,33]. These results support our assumption that coursework is potentially structured as a graph; therefore, incorporating such a nature into a knowledge tracing model improves its performance, thus working as a relational inductive bias. The improvements in prediction performance and its interpretability provide benefits to various aspects of real-world educational environments: accurate predictions help students identify content suitable for their individual knowledge level, thereby facilitating more efficient learning; moreover, such predictions help e-learning platforms and teachers to increase student engagement, and thus reduce dropout rates.

5.2. Solving the implicit graph structure problem

In reformulating knowledge tracing as a GNN application, we pointed out the problem that the graph structure itself is unknown in many practical cases. Further, we proposed and empirically validated various implementations of the graph structure as shown in Table 4 and Fig. 4. This enables researchers to benefit from a performance boost without requiring a costly manual annotation process to track the relationships between concepts. In addition, it facilitates educational experts to set a new criterion for improving curriculum design; this in turn, would help them to determine a good knowledge structure.

On the other hand, focusing on the implementation of the latent graph structure of GKT, there are small differences in prediction performance between different implementations as shown in Table 4; therefore, it is not possible to conclude which approach is the best. One reason for this phenomenon could be that the neural networks used to represent edges have a strong expressive power. Thus, the model could sufficiently optimize by adjusting the parameters without adjusting the edge weights, and the contribution of the latter became relatively small. The differences can be expanded by applying some constraints on edge-weight distribution or information propagation between nodes, for example, sparse edge connections or edge direction-aware propagations, so that the models have to leverage the edge weights more. Further investigation into incorporating the constraints into models and analyzing the relationships between the graph structure and prediction performance is planned for future work.

5.3. Dataset generalizability

In the experiments, we conducted an exhaustive validation on a diverse set of mathematical datasets and confirmed that the performance improvement of our method is not limited to specific datasets, as shown in Table 4–6. Although the subject of the datasets is limited to Mathematics and given that there exists a report of DKT applications to programming education [29], GKT can also be effective in various subjects. Moreover, GKT is a generic algorithm that completes previous algorithms such as the DKT. Because the proposed method can learn the latent graph structure from data without expensive manual annotation, our method can also be easily applied to and validated on other subject datasets. In addition, it is expected to learn different graph structures on different subject datasets, which could impart novel information on human knowledge structure. One limitation is that there exist few public datasets for subjects other than Mathematics, and the validation on other subjects is fundamentally difficult. Collecting student exercise logs on various subjects and validating various methods, including GKT, with them will be a common theme for knowledge tracing research. This would extend the application of knowledge tracing to wider educational environments.

5.4. Potentials by incorporating richer GNN architectures

In this work, we proposed the first GNN-based knowledge tracing method and validated relatively simple architectures. In the following, we discuss three future directions that can be pursued to improve our model by incorporating richer GNN architectures.

One is to impose appropriate constraints for information propagation between nodes based on their edge types. In this study, for a fair comparison, we defined two types of edges for both the statistics-based and learning-based approaches. However, we did not impose any constraint on each node type; therefore, the meaning of each node type such as dependency direction and causality may be slight, especially for the learned edges. A solution to this is to impose some constraints for information propagation between nodes based on their edge types, e.g., defining directions for edges and limiting propagation to only one direction, from source nodes to target nodes. Additionally, this can serve as a relational inductive bias and improve the sample efficiency and interpretability of GKT.

Another direction is to incorporate a hidden state common to all concepts, such as that of DKT, into GKT. Although adopting only a single hidden vector to represent the knowledge state complicated the modeling of complex interactions between concepts in DKT, as described in Section 3.4, adding these representations to GKT could improve the performance because they could serve as global features [3]. The term "global features" indicates features common to each node, and these features can represent knowledge states that are common across variable concepts or the student's original intelligence and are invariant to individual concept understandings.

The third possible solution is to implement multi-hop propagation. In this study, we limited the propagation to a single hop, i.e., the information from answering a certain node is propagated only to its neighboring node in one time step. However, to effectively model the complex human learning mechanism, using multiple hops will be more effective. Additionally, this can enable the model to learn sparse edge connections because the model can propagate features to distant nodes without having any direct connections to the nodes.

6. Conclusions

We proposed a GNN-based knowledge tracing method called GKT, which considers a latent knowledge structure that has been ignored by previous deep-learning-based methods. Casting the knowledge structure as a graph, we reformulated the knowledge tracing task as an application of a GNN. Because the knowledge graph structure is not explicitly provided in many cases, we proposed various implementations of the graph structure. Empirical validations on eight open datasets indicated that our method could potentially improve the prediction of student proficiency and demonstrated more highly interpretable predictions than those of the previous methods. These results confirmed the potential of our method to enhance knowledge tracing performance and its applicability to real educational environments. We believe this work could help improve the learning experience of students in diverse settings.

References

[1] M. Bastian, S. Heymann and M. Jacomy, Gephi: An Open Source Software for Exploring and Manipulating Networks, 2009, http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154.

[2] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende et al., Interaction networks for learning about objects, relations and physics, in: *Advances in Neural Information Processing Systems*, 2016, pp. 4502–4510.

[3] P.W. Battaglia, J.B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner et al., Relational inductive biases, deep learning, and graph networks, 2018, Preprint, arXiv:1806.01261.

[4] H.-S. Chang, H.-J. Hsu and K.-T. Chen, Modeling exercise relationships in E-learning: A unified approach, in: *EDM*, 2015, pp. 532–535.

[5] Z. Chen, L. Li and J. Bruna, Supervised Community Detection with Line Graph Neural Networks, 2018.

[6] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014, Preprint, arXiv:1406.1078.

[7] D.-A. Clevert, T. Unterthiner and S. Hochreiter, Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs), 2015, Preprint, arXiv:1511.07289.

[8] A.T. Corbett and J.R. Anderson, Knowledge tracing: Modeling the acquisition of procedural knowledge, *User Modeling and User-adapted Interaction* **4**(4) (1994), 253–278. doi:10.1007/BF01099821.

[9] M. Feng, N. Heffernan and K. Koedinger, Addressing the assessment challenge with an online system that tutors as it assesses, *User Modeling and User-Adapted Interaction* **19**(3) (2009), 243–266. doi:10.1007/s11257-009-9063-7.

[10] A. Fout, J. Byrd, B. Shariat and A. Ben-Hur, Protein interface prediction using graph convolutional networks, in: *Advances in Neural Information Processing Systems*, 2017, pp. 6530–6539.

[11] J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals and G.E. Dahl, Neural message passing for quantum chemistry, 2017, Preprint, arXiv:1704.01212.

[12] M. Gori, G. Monfardini and F. Scarselli, A new model for learning in graph domains, in: *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, Vol. 2, IEEE, 2005, pp. 729–734. doi:10.1109/IJCNN.2005.1555942.

[13] W. Hamilton, Z. Ying and J. Leskovec, Inductive representation learning on large graphs, in: *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[14] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015, Preprint, arXiv:1502.03167.

[15] D. Kingma and J. Ba, Adam: A method for stochastic optimization, 2014, Preprint, arXiv:1412.6980.

[16] D.P. Kingma and M. Welling, Auto-encoding variational bayes, 2013, Preprint, arXiv:1312.6114.

[17] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling and R. Zemel, Neural relational inference for interacting systems, 2018, Preprint, arXiv:1802.04687.

[18] T.N. Kipf and M. Welling, Semi-supervised classification with graph convolutional networks, 2016, Preprint, arXiv:1609.02907.

[19] A. Krizhevsky, I. Sutskever and G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[20] Q. Liu, S. Tong, C. Liu, H. Zhao, E. Chen, H. Ma and S. Wang, Exploiting cognitive structure for adaptive learning, in: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 627–635. doi:10.1145/3292500.3330922.

[21] C.J. Maddison, A. Mnih and Y.W. Teh, 2016, The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. Preprint, arXiv:1611.00712.

[22] C. Piech, J. Bassen, J. Huang, S. Ganguli, M. Sahami, L.J. Guibas and J. Sohl-Dickstein, Deep knowledge tracing, in: *Advances in Neural Information Processing Systems*, 2015, pp. 505–513. doi:10.1007/978-3-319-26561-2_60.

[23] A. Sanchez-Gonzalez, N. Heess, J.T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell and P. Battaglia, Graph networks as learnable physics engines for inference and control, 2018, Preprint, arXiv:1806.01242.

[24] J. Stamper, A. Niculescu-Mizil, S. Ritter, G.J. Gordon and K.R. Koedinger, 2010a. Algebra I 2005–2006. Development data set from KDD Cupucational Data Mining Challenge, 2010th edn, http://pslcdatashop.web.cmu.edu/KDDCup/downloads.jsp.

[25] J. Stamper, A. Niculescu-Mizil, S. Ritter, G.J. Gordon and K.R. Koedinger, 2010b. Algebra I 2006–2007. Development data set from KDD Cupucational Data Mining Challenge, 2010th edn, http://pslcdatashop.web.cmu.edu/KDDCup/downloads.jsp.

[26] J. Stamper, A. Niculescu-Mizil, S. Ritter, G.J. Gordon and K.R. Koedinger, 2010c. Bridge to Algebra 2006–2007. Development data set from KDD Cupucational Data Mining Challenge, 2010th edn, http://pslcdatashop.web.cmu.edu/KDDCup/downloads.jsp.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser and I. Polosukhin, Attention is all you need, in: *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[28] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio and Y. Bengio, Graph attention networks, Vol. 1, 2017, p. 2, Preprint, arXiv:1710.10903.

[29] L. Wang, A. Sy, L. Liu and C. Piech, Deep knowledge tracing on programming exercises, in: *Proceedings of the Fourth*, ACM Conference on Learning@ Scale, ACM, 2017, pp. 201–204.

[30] X. Wang, R. Girshick, A. Gupta and K. He, Non-local neural networks, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7794–7803.

[31] R.J. Williams and D. Zipser, A learning algorithm for continually running fully recurrent neural networks, *Neural computation* **1**(2) (1989), 270–280. doi:10.1162/neco.1989.1.2.270.

[32] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie and T. Tan, Session-based recommendation with graph neural networks, in: *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, 2019, pp. 346–353.

[33] J. Zhang, X. Shi, I. King and D.-Y. Yeung, Dynamic Key-Value Memory Network for Knowledge Tracing, 2016, Preprint, arXiv:1611.08108.