Graph-based Knowledge Tracing: Modeling Student Proficiency Using Graph Neural Network

Hiromi Nakagawa
nakagawa@weblab.t.u-tokyo.ac.jp
The University of Tokyo
Tokyo, Japan

Yusuke Iwasawa
iwasawa@weblab.t.u-tokyo.ac.jp
The University of Tokyo
Tokyo, Japan

Yutaka Matsuo
matsuo@weblab.t.u-tokyo.ac.jp
The University of Tokyo
Tokyo, Japan

ABSTRACT

Recent advancements in computer-assisted learning systems have caused an increase in the research of *knowledge tracing*, wherein student performance on coursework exercises is predicted over time. From the viewpoint of data structure, the coursework can be potentially structured as a graph. Incorporating this graph-structured nature into the 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 the graph neural network (GNN), 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.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Information systems** → *Data mining*; • **Applied computing** → *Learning management systems.*

KEYWORDS

Knowledge tracing, Graph neural network, Educational data mining, Learning sciences

ACM Reference Format:
Hiromi Nakagawa, Yusuke Iwasawa, and Yutaka Matsuo. 2019. Graph-based Knowledge Tracing: Modeling Student Proficiency Using Graph Neural Network. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI '19), October 14–17, 2019, Thessaloniki, Greece.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3350546.3352513

1 INTRODUCTION

Recent advancements in computer-assisted learning systems have led to an increase in the research of *knowledge tracing* [5], in which student performance on coursework exercises is predicted over time. Accurate predictions help students identify contents suitable for their individual knowledge level, thereby facilitating more efficient learning. This is particularly important for e-learning platforms and teachers as they can increase student engagement and also prevent dropout. Although various knowledge tracing methods have been proposed, Piech et al. [17] reported that a method called deep knowledge tracing (DKT), which leverages recurrent neural networks (RNNs) [24], performs significantly better than any previous method.

From the viewpoint of data structure, a coursework can be potentially 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. Let us consider an example as follows. A coursework knowledge is divided into three concepts $v = \{v_1, v_2, v_3\}$, and understanding of v_1 is dependent on understanding of v_2. Simultaneously, 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 respectively perceived as nodes and edges of a graph, where the edges are directed from v_3 to v_2 and from v_2 to v_1. It is known that incorporating prior knowledge about the graph-structured nature of data into models improves their performance and interpretability [2]. 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 such a nature. The architectures of the previous deep-learning-based methods, such as the RNN, generally perform well on sequential data, but cannot effectively handle graph-structured data.

Recently, the research on the graph neural network (GNN) [8], which handles graph-structured data by deep learning, is garnering attention. Although operating on such irregular domain data has been challenging for the existing machine learning approaches, various generalization frameworks and important operations have been developed [2, 7, 23] with successful results in various research areas. Battaglia et al. [2] explained the expressive power of the GNN from the perspective of relational inductive biases, which improve the sample efficiency of machine learning models by incorporating the human's prior 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 the process of coursework proficiency while considering the latent knowledge structure.

A challenge encountered when performing knowledge tracing using the GNN is the definition of the latent graph structure. The GNN possesses 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 contents relationships; however, it requires deep domain knowledge and a substantial amount of time; therefore, it is difficult to define in advance the graph structure for all the contents in an e-learning platform. 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 of concept answering. 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 studies 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 enable their application in our case.

In this paper, we propose a GNN-based knowledge tracing method, i.e., a 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)The 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)The coursework knowledge is structured as a graph, which affects the updating of student knowledge state: if a student answers a concept, correctly or incorrectly, his/her knowledge state is affected not only with regard to the answered concept but also other related concepts that are represented as neighboring nodes in the graph.

Using a subset of two open datasets of math exercise logs, we empirically validated our method. In terms of prediction performance, our model outperformed previous deep learning-based methods, which indicates that our model possesses high potential for improving the prediction of student performance. In addition, by analyzing the prediction patterns of the trained model, the process of student proficiency, i.e., the concepts which the students gained understanding of, and the time required by them for the same, can be clearly interpreted from the model's predictions, whereas the previous methods demonstrated inferior interpretability. This means that our model provides more interpretable predictions compared to the previous ones. The obtained results verify the potential of our model to improve the performance and applicability of knowledge tracing for application in real educational environments, while assuming that the target coursework is graph-structured.

Our contributions are as follows:

- We demonstrated that formulating the knowledge tracing as an application of GNN improves the student performance prediction, without requiring any additional information. Students can master the coursework more efficiently based on a more precise content personalization. E-learning platforms can provide a higher quality service to maintain high user engagement.
- Our model improves the interpretability of the model's prediction. Teachers and students can recognize the students' knowledge state more precisely and the students can be more motivated to work on the recommended exercises by understanding why they are recommended. E-learning platforms and teachers can redesign the coursework curriculum more easily by analyzing in which point students fail.
- To solve the implicit graph structure problem, we propose various implementations and empirically validate their effectiveness. Researchers can benefit from a performance boost without requiring costly annotations by human experts about the relationships between concepts. The educational experts can have a new criterion to consider what the good knowledge structure is to improve the curriculum design.

2 RELATED WORK

2.1 Knowledge Tracing

Knowledge tracing is the task of predicting student performance based on coursework exercises over time. It can be formulated as $\mathbf{y}^t = KT(\mathbf{x}^1, \cdots, \mathbf{x}^t)$, where $\mathbf{x}^t = \{q^t, r^t\}$ is a tuple that considers an answered exercise q^t and whether the exercise was 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.

Since Piech et al. [17] first proposed the deep-learning-based knowledge tracing method, i.e., DKT, and demonstrated the considerable expressive power of the RNN, many researchers have adopted the RNN or its extensions as the KT. The models define a hidden state, or a student's temporal knowledge state, $\mathbf{h}^t$, and recurrently update it based on the student exercise performance over time. RNN-based models must represent $\mathbf{x}$ as a fixed size vector, and in many cases, $\mathbf{x}^t$ is represented by concatenating two binary vectors that represent which exercise was answered correctly and incorrectly, respectively. Thus, for datasets with N unique exercises, $\mathbf{x}^t \in \{0, 1\}^{2N}$. 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 (NLL) of the observed sequence of student responses under the model.

2.2 GNN

The GNN [8] is a type of neural network that can operate on graph-structured data. The graph is a type of data structure that represents objects and their relationships as nodes and edges, respectively. Although operating on such irregular domain data has been challenging for the existing machine learning approaches, the considerable expressive power of graphs has increased research on the GNN, and in recent years, various generalization frameworks and important operations have been developed [2, 7, 23] with successful results in various research areas, such as social science [9, 14] and natural science [1, 6, 18].

Our primary motivation for using GNN is the success of convolutional neural networks (CNNs) [15]. Using a local connection, weight sharing, and multilayer architecture, the CNN can extract multiscale local spatial features and compose them to construct expressive representations, thereby resulting in breakthroughs in various research areas, such as computer vision. However, CNNs can only operate on regular Euclidean data such as images and texts, whereas several applications in the real world generate non-Euclidean data. The 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. [2] explained such an 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 the human's prior knowledge about the nature of data.

Among the several research topics in GNN, edge feature learning [1, 3, 7] is the most relevant to our work. Graph attention networks (GATs) [21] apply the multi-head attention mechanism [20] to the GNN and enable learning edge weights during training without requiring their predefinition. Neural relational inference (NRI) [13] leverages a variational autoencoder (VAE) [12] to learn the latent graph structure in an unsupervised manner. Our method assumes a latent graph structure underlying the knowledge concepts of a coursework and models the temporal transition of the 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 GRAPH-BASED KNOWLEDGE TRACING

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 $V = \{v_1, \cdots, v_N\}$, 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 itself $\mathbf{h}^t_i$ and for 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 the GNN to the knowledge tracing task and leverages the graph-structured nature of knowledge. We present the architecture of GKT in Figure 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 $j \in \mathcal{N}_i$:

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

where $\mathbf{x}^t \in \{0,1\}^{2N}$ is an input vector that represents the exercise 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 response of 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:

$$\begin{aligned} \mathbf{m}^{t+1}_k &= \begin{cases} f_{\text{self}}(\mathbf{h}'^t_k) & (k = i) \\ f_{\text{neighbor}}(\mathbf{h}'^t_i, \mathbf{h}'^t_k) & (k \neq i) \end{cases} \qquad (1) \\ \tilde{\mathbf{m}}^{t+1}_k &= \mathcal{G}_{ea}(\mathbf{m}^{t+1}_k) \\ \mathbf{h}^{t+1}_k &= \mathcal{G}_{gru}(\tilde{\mathbf{m}}^{t+1}_k, \mathbf{h}^t_k) \end{aligned}$$

where f_{self} is a multilayer perceptron (MLP), $\mathcal{G}_{ea}$ is an erase–add gate used in Zhang et al. [25], and $\mathcal{G}_{gru}$ is a gated recurrent unit (GRU) gate [4]. f_{neighbor} is an arbitrary function that defines the information propagation to neighboring nodes based on a knowledge graph structure. 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:

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

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 NLL of the observations.

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

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:

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

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

Dense graph is a simple densely connected graph, where $\mathbf{A}_{i,j}$ is $\frac{1}{|V|-1}$ if $i \neq j$; else, it is 0.

Transition graph is a transition probability matrix, where $\mathbf{A}_{i,j}$ is $\frac{n_{i,j}}{\sum_k n_{i,k}}$ if $i \neq j$; else, it is 0. 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, which was proposed by Piech et al. [17].

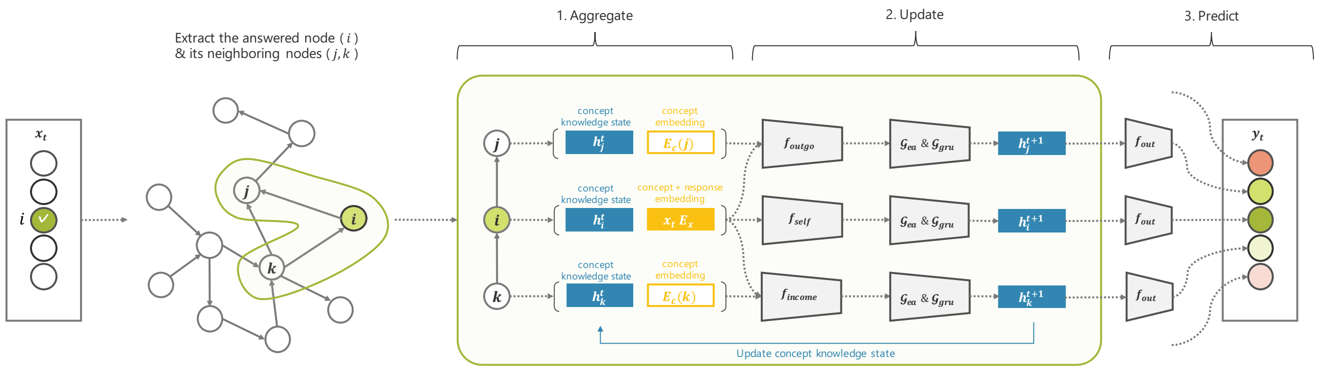


Figure 1: Architecture of GKT. When a student answers a concept, GKT first aggregates the node features related to the answered concept, subsequently updates the student's knowledge states for only the related concepts, and finally predicts the probability of the student answering each concept correctly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

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 similarly as equation 2.

Multi-head Attention (MHA) leverages the multi-head attention mechanism [20] to infer the edge weights between two nodes based on their features. f_{neighbor} is defined as follows:

$$f_{\text{neighbor}}(\mathbf{h'}_i^t, \mathbf{h'}_j^t) = \frac{1}{K} \sum_{k \in K} \alpha_{ij}^k f_k(\mathbf{h'}_i^t, \mathbf{h'}_j^t),$$

where k is the head index among a total of K heads, α_{ij}^k 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 (VAE) assumes the discrete latent variable that represents the types of edges and infers them based on node features. f_{neighbor} is defined as follows:

$$f_{\text{neighbor}}(\mathbf{h'}_i^t, \mathbf{h'}_j^t) = \sum_{k \in K} z_{ij}^k f_k(\mathbf{h'}_i^t, \mathbf{h'}_j^t),$$

where k is the edge type among a total of K types, z_{ij}^k is a latent variable sampled from the Gumbel–Softmax distribution [16], and f_k is a neural network for the k-th edge type. VAE minimizes the NLL 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 [1, 3, 7], and the MHA and VAE were motivated by the GAT [21] and NRI [13], respectively; however, we modify them based on two aspects. First, we calculate the edge weights based on the static features, such as the embeddings of concepts and responses, instead of the dynamic ones. This enables the learning of the knowledge graph structure invariant to 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 $O(KN^2)$ of the original NRI to $O(KN)$.

We discuss the difference between these three learning-based approaches in Section 5.1

3.4 Comparison with Previous Methods

The comparison of the proposed method with the previous ones can be done using two aspects. We present the comparison in Figure 2.

The first aspect 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 state for each concept is 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. [25] proposed the dynamic key-value memory network (DKVMN) that utilizes two memory matrices, one of which can be regarded as a stack of the student temporal knowledge state $\mathbf{h}^t$, 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 the DKVMN defines another low-dimensional latent concept and subsequently models their knowledge states.

The other aspect refers to the interactions between concepts during knowledge state update. In the 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 the concepts.

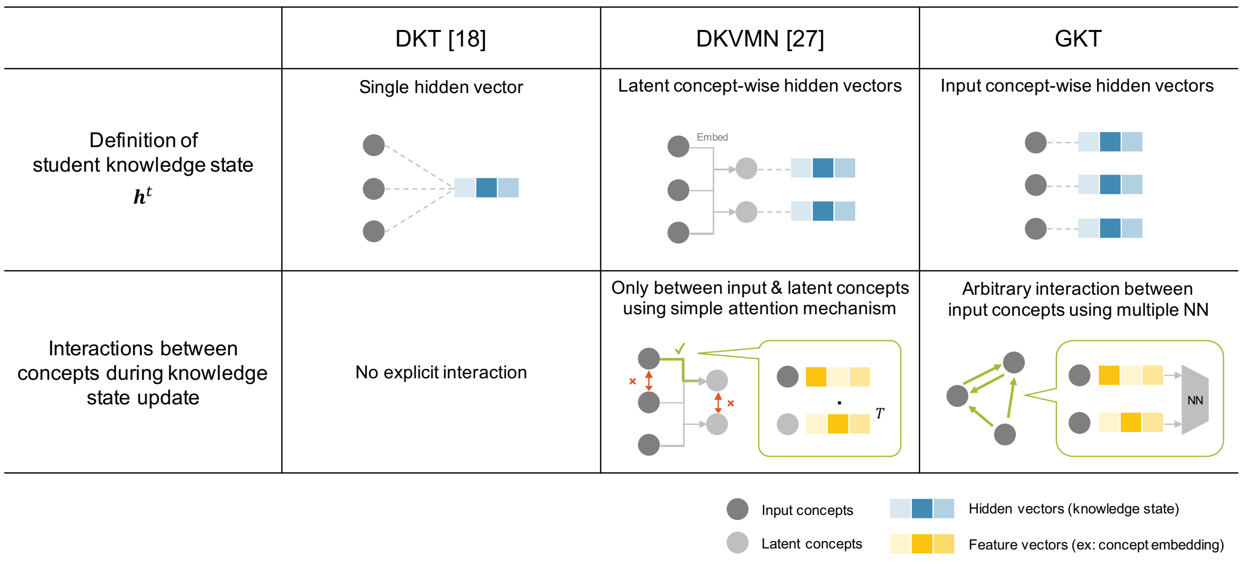

DKT [18]
DKVMN [27]
GKT
Definition of student knowledge state h^t
Single hidden vector
Latent concept-wise hidden vectors
Embed
Input concept-wise hidden vectors
Interactions between concepts during knowledge state update
No explicit interaction
Only between input & latent concepts using simple attention mechanism
Arbitrary interaction between input concepts using multiple NN
NN
Input concepts
Latent concepts
Hidden vectors (knowledge state)
Feature vectors (ex: concept embedding)

Figure 2: Comparison of GKT with previous methods.

Table 1: Examples of concept tags

Dataset	Concept name
ASSISTments	Absolute Value Greatest Common Factor Proportion
KDDCup	Calculate area of overlap Identify GCF Measure acute angle

4 EXPERIMENTS

4.1 Datasets

For the experiments, we used two open datasets of student math exercise logs: ASSISTments 2009-2010 “skill-builder” provided by the online educational service ASSISTments[1] (hereinafter called “ASSISTments”) and Bridge to Algebra 2006-2007 [19] used in the Educational Data Mining Challenge of KDDCup (hereinafter called “KDDCup”). In both datasets, human-predefined knowledge concept tags were assigned to each exercise. We provide examples of the existing concept tags in each dataset in Table 1.

We preprocess each dataset using certain conditions. For ASSISTments, we combine simultaneous answer logs into one, subsequently extract the logs associated with the named concept tags, and finally extract the logs associated with the concept tags answered at least 10 times. For KDDCup, we consider the combination of *problem* and *step* as one answer, then extract the logs associated with the concept tags that are named and are not dummies, and finally extract the logs associated with the concept tags answered at least 10 times. Combining simultaneous answer logs into one set prevents an unfairly high prediction performance because of the frequently co-occurring tags. Excluding concept tags that are unnamed or dummies removes the noise. Thresholding the logs with the number of times each concept tag was answered secures a sufficient number of logs to remove the noise. After preprocessing the datasets using the conditions above, we obtained 62, 955 logs consisting of 1, 000 students and 101 skills for the ASSISTments dataset and 98, 200 logs consisting of 1, 000 students and 211 skills for the KDDCup dataset.

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

4.2 Implementation Details

For each dataset, we divide the students into training:validation:test = 8:1:1. We train the model using the training students’ data and adjust the hyperparameters using the validation students’ data.

DKT We searched the hyperparameters following Piech et al. [17]. The size of the hidden layer was 200 and we used a GRU 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, and we used Adam [11] as an optimizer with a learning rate of 0.001.

DKVMN We searched hyperparameters following Zhang et al. [25]. The size of the memory slot was 20 for the ASSISTments dataset and 50 for the KDDCup dataset. The size of the hidden vector was 32 for the ASSISTments dataset and 128 for the KDDCup dataset. The batch size was 32, and we used Adam as an optimizer with a learning rate of 0.001.

GKT The size of all hidden vectors and the embedding matrix was 32. For the MLPs in the model, we applied a dropout from the hidden vectors to the output vectors with a drop rate of 0.5, and applied batch normalization [10] for the output layers. The batch size was 16, and we used Adam as an optimizer with a learning rate of 0.01. 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: incoming edges and outgoing edges.

Table 2: Comparison of prediction performance.

Method		AUC	
		ASSISTments	KDDCup
Baseline	DKT	0.709	0.751
	DKVMN	0.710	0.753
Statistics-based	Dense graph	0.722	0.762
	Transition graph	0.721	**0.769**
	DKT graph	**0.723**	0.764
Learning-based	PAM	0.719	0.762
	MHA	**0.723**	0.766
	VAE	0.722	**0.769**

4.3 Prediction Performance

First, we evaluate the prediction performance of GKT. We set the DKT and DKVMN as baselines and compare the area under the curve (AUC) score of GKT with them. We list the results in Table 2. The highest scores are denoted in bold, for each dataset. In both the datasets, GKT indicated the highest AUC score. This suggests that GKT can trace the student knowledge state better than the previous methods, which do not consider the knowledge graph structure. In the statistics-based approaches, the transition or DKT graph outperformed the simple dense graph. This indicates that representing sparse relationships between nodes in a certain manner enables GKT to perform better, although the degree of improvement was small. In the learning-based approaches, the MHA or VAE, which estimate edge information using neural networks based on node features, performed better than the PAM, as the latter simply optimizes the adjacency matrix. However, the best performance for both the approaches was almost similar, and the methods that performed the best differed between the two datasets. Therefore, further experiments over various datasets are warranted.

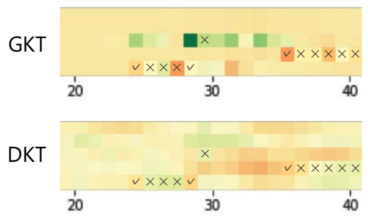
4.4 Interpretability of the Prediction

Next, we visualize how the model predicts the student knowledge state changes over time and evaluate the interpretability of its prediction. This visualization helps students and teachers recognize the former's knowledge state, efficiently and intuitively; consequently, its interpretability is of importance. Here, we evaluate interpretability based on the following two points: 1) Whether the model updates only the related concepts to the answered concept at each time step. 2) Whether the update is reasonable with the given graph structure. Although previous works [17, 25] performed a similar analysis, our study extends this approach as follows to analyze the temporal transition more precisely:

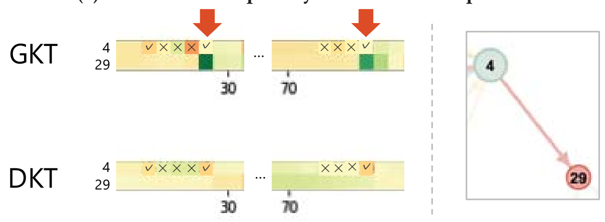
(1) Randomly sample a student log until time step T.
(2) Remove bias vectors from the output layers in a trained model.
(3) Input the student answer logs $\mathbf{x}_{t\leq T}$ to the trained model and stack the output $\mathbf{y}_{t\leq T}$.
(4) Normalize the output values at each time step from 0 to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

We randomly sampled a student and depict the student knowledge state change to a subset of concepts in Figures 3a and 3b. 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 the 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 Figure 3a, 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 related concepts. In addition, Figure 3b shows that although concept 29 is not answered, its knowledge state is clearly updated at $t = 28$ and $t = 75$. In 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.



(a) A subset of frequently answered concepts.



(b) Subset of concepts neighboring the graph.

Figure 3: Visualization of transition of predicted knowledge states for concept subsets.

4.5 Network Analysis

Finally, we extract the learned graph structure from the trained GKT model and analyze it. In the learning-based approaches, GKT learns the graph structure that helps predict student performance. Thus, the graph extracted from the model that demonstrated a high prediction performance can provide insights into a good knowledge structure.

The networks are depicted in Figure 4, where the left-hand side shows the network overview and the right-hand side shows the local connection of the graphs. The color of the nodes is graded from blue to red, where the earlier an exercise is answered, the bluer is the shade. The size of the nodes is proportional to their out-degree, implying that larger nodes affect more nodes.

First, in the DKT graph, which is visualized for comparison, similarly colored nodes are connected with each other, thereby generating clusters. As DKT models the hidden states of all concepts with the same single hidden vector, modeling the long-time

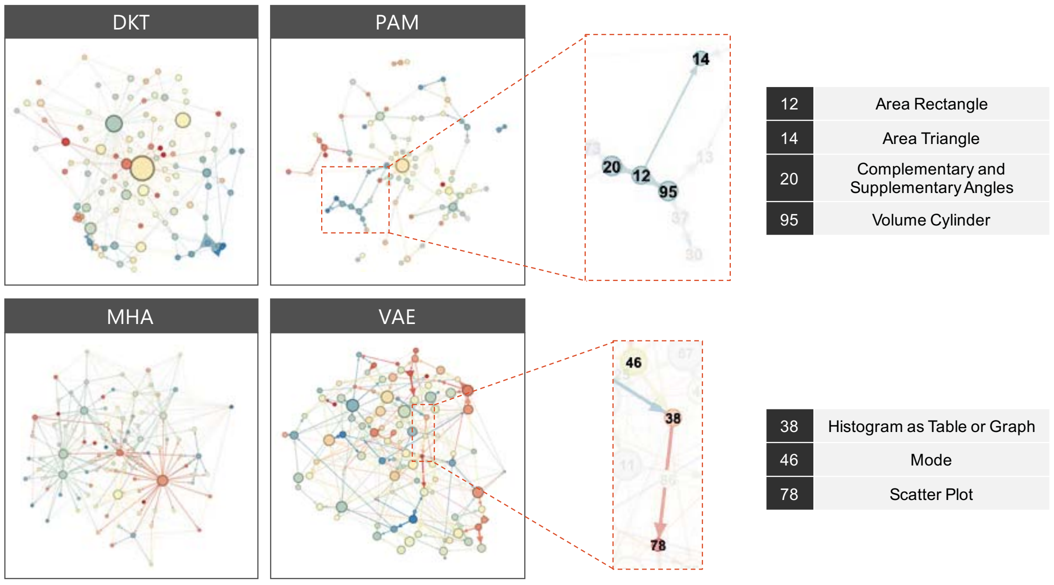


Figure 4: Network visualizations of graphs derived from DKT and GKT.

dependency between concepts is difficult. Thus, the model tends to learn the dependencies between nodes that are answered in temporally close order. The graph extracted from the PAM exhibited a structure similar to that of the DKT graph, where clusters are constructed; from the top right of the figure, we can perceive that some geometrical concepts are connected. The graph extracted from the MHA shows several outcoming edges from certain nodes. Although it is possible that the model might have learned some special dependencies between concepts that differ from those in the other graphs, its prediction can be biased. Hence, we must evaluate the effect of such a structure on the prediction performance. The graph extracted from the VAE differed from the other graphs in that it formed a dense graph, where several nodes were connected with one another. Although many of these connections are difficult to interpret, from the bottom right of the figure, we can identify that some statistical concepts are connected.

5 DISCUSSION

5.1 Differences Between Learning-based Approaches

In this study, to solve the implicit graph structure problem, we proposed two implementation approaches and also developed three methods, namely, PAM, MHA, and VAE, in the learning-based approaches. In the following, we discuss the difference between them.

The difference between PAM and the other two methods is in the identification of whether or not the edge feature estimation can be conditioned. In PAM, the adjacency matrix is directly optimized, and no condition exists for estimating the edge features. Whereas, in MHA and VAE, features to condition the prediction of edge features can be selected; herein, we selected concept embeddings $\mathbf{E}_c$ as the input so that a single stable concept graph invariant to students could be learned, and it is also the most identical and simple setting for knowledge tracing.

The difference between MHA and VAE is in the method of calculating the edge weights. In MHA, when a concept i is answered, attention scores are calculated and normalized across all the neighboring concepts. Furthermore, MHA can learn k edge weights for each edge using multi-head attention. Whereas, in VAE, based on the pairs of answered concepts and each of its neighboring concepts, each edge feature is calculated independently; the Gumbel Softmax function renders only one of the k edge weights near 1 and the others near 0. In addition, VAE can define some prior distributions such that the sparsity of the whole graph can be defined.

Consequently, each approach learned different graph structures, as shown in Section 4.5; however, we found minor differences in their prediction performances. It is possible that incorporating some constraints into the edge estimation may result in a difference in prediction performance. Therefore, the effect of the choice from the learning-based approaches must be investigated as a part of future work.

5.2 Dataset Generalizability

In this study, to validate the performance of our model, we used a subset of the original datasets, where students and concepts with low data are excluded to reduce noise, as described in Section 4.1. Although the experiments demonstrate the potential of our model in terms of improving the prediction performance and its interpretability, we must loosen the constraints to demonstrate that the benefits of our model can be shown more clearly under the same settings of the previous studies.

In addition, we must validate the applicability of our method in different subject datasets. The subjects of the datasets we used are limited to math, as in the case of Piech et al. [17]. Given that there exist reports of DKT applications to programming education [22] and

GKT is a generic algorithm that completes the previous algorithms such as DKT, GKT can also be effective in various subjects. However, different subjects can exhibit different latent graph structures; thus, we must compare the effect of subjects to our model, e.g., what type of graph is appropriate for predicting student performance or can be obtained from the learning-based approaches.

5.3 Incorporating Richer GNN Architectures

We proposed the first GNN-based knowledge tracing method and validated relatively simple architectures. In the following, we discuss three directions to improve our model.

One is imposing 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 for each node type; therefore, the meaning for 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 limit 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 is incorporating a hidden state common to all concepts, such as that of DKT, into GKT. Although adopting only a single hidden vector to represent the student knowledge state complicated the modeling of complex interactions between concepts in DKT, adding this type of representation to GKT could improve the performance by serving as global features [2]. Global features imply the features common for each node and can represent knowledge states common across variable concepts or the student's original intelligence invariant to individual concept understandings.

The last possible solution is implementing multihop propagation. In this study, we limited the propagation to a single hop, i.e., the information of answering a certain node is propagated only to its neighboring node at one time step. However, to effectively model the human learning mechanism, using multiple hops will be more appropriate. Additionally, this can enable the model to learn the sparse connections, because the model can propagate features to distant nodes without connecting to other nodes.

6 CONCLUSION

We proposed a GNN-based knowledge tracing method called GKT, which considers the latent knowledge structure that has been ignored by the previous deep learning-based methods. Casting the knowledge structure as a graph, we reformulated the knowledge tracing task as an application of the GNN. Empirical validations on two open datasets indicated that our method could potentially improve the prediction of student proficiency and demonstrated highly interpretable predictions than those of the previous methods. These results confirmed the potential of our method in enhancing knowledge tracing performance and the possibility of its application to real educational environments. We believe this work could help improve the learning experience of students in diverse settings.

REFERENCES

[1] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. 2016. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*. 4502–4510.

[2] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).

[3] Zhengdao Chen, Lisha Li, and Joan Bruna. 2018. Supervised Community Detection with Line Graph Neural Networks. (2018).

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[5] Albert T Corbett and John R Anderson. 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-adapted Interaction* 4, 4 (1994), 253–278.

[6] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. 2017. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems*. 6530–6539.

[7] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212* (2017).

[8] Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, Vol. 2. IEEE, 729–734.

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[10] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[11] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[12] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[13] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. 2018. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687* (2018).

[14] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[16] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. *arXiv preprint arXiv:1611.00712* (2016).

[17] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. 2015. Deep knowledge tracing. In *Advances in Neural Information Processing Systems*. 505–513.

[18] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. 2018. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242* (2018).

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

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 5998–6008.

[21] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* 1, 2 (2017).

[22] Lisa Wang, Angela Sy, Larry Liu, and Chris Piech. 2017. Deep knowledge tracing on programming exercises. In *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale*. ACM, 201–204.

[23] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. 2018. Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7794–7803.

[24] Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation* 1, 2 (1989), 270–280.

[25] Jiani Zhang, Xingjian Shi, Irwin King, and Dit-Yan Yeung. 2016. Dynamic Key-Value Memory Network for Knowledge Tracing. *arXiv preprint arXiv:1611.08108* (2016).