Dynamics-Aware Gated Graph Attention Neural Network for Student Program Classification and Knowledge Tracing

Tiancheng Jin[1,2,3(✉)], Liang Dou[2], Guang Yang[2], Aimin Zhou[1,2], Xiaoming Zhu[3], and Chengwei Huang[3]

[1] Shanghai Institute of AI for Education, East China Normal University, Shanghai, China
52205901026@stu.ecnu.edu.cn, amzhou@cs.ecnu.edu.cn
[2] School of Computer Science and Technology, East China Normal University, Shanghai, China
ldou@cs.ecnu.edu.cn
[3] Zhejiang Lab, Hangzhou, China
{zhuxiaoming,huangchengwei}@zhejianglab.com

Abstract. Faced with a large number of questions on the programming OJ (Online Judge) system, students are usually mindless when choosing questions, which is not conducive to helping students quickly improve their programming ability. Programming Knowledge Tracing (PKT) is a technology that dynamically traces students' programming knowledge states using their historical learning data including submitted programs. Relying on PKT, OJ can find students' unmastered knowledge points, and recommend questions examining these knowledge points to students, so as to help students overcome their weakness. However, existing program analysis modules in PKT models ignore dynamic information of program. Therefore, this paper proposes Dynamics-Aware Gated Graph Attention Neural Network (DGGANN), which inputs test cases of questions into program, obtains call frequency coefficients of every node in Abstract Syntax Tree (AST) through code coverage statistical tool, and introduces such call frequency information into process of program analysis. This paper applies DGGANN to two tasks in our experiments: classifying programs by functionalities and PKT. Experimental results show that our approach can achieve higher performance than the state-of-the-art models in both tasks on datasets of two well-known OJ systems named CodeForces and Libre.

Keywords: Programming Knowledge Tracing · Dynamic Program Analysis · Online Judge · Artificial Intelligence in Education

1 Introduction

OJ (Online Judge) is a kind of teaching assistant system for programming, which can provide students with programming questions, judge correctness of the program submitted by students online according to pre-designed test cases, and feed back the judging results to students for helping them find and consolidate unmastered knowledge points.

X. Yuan et al. (Eds.): ICEKIM 2023, AHCS 13, pp. 1640–1652, 2023.
https://doi.org/10.2991/978-94-6463-172-2_182

At present, answering questions autonomously on OJ has become a common learning method of programming, but the excessive questions in OJ and lack of teacher guidance make students waste too much valuable learning time in choosing suitable questions.

Knowledge Tracing (KT) is a technology that dynamically traces students' knowledge states using their historical learning data. Relying on KT, OJ can find students' unmastered knowledge points, and recommend questions examining these knowledge points to students, so as to help students overcome their weakness and im-prove programming ability. Existing KT models usually mine the correlation between questions through knowledge points examined by questions, but quite a few questions are not labeled with knowledge point in some well-known OJ systems, for example, more than 30% of questions in Libre and Luogu do not have knowledge point labels. If there are a large number of questions without knowledge point labels, the performance of KT model will be reduced. Since code implementations of programming questions with the same knowledge points are similar, some researchers utilize program analysis method to mine knowledge point information contained in code implementations of questions and integrates such information into KT model for solving the problem of "lacking knowledge point labels" and improving prediction accuracy, such kind of KT models are named Programming Knowledge Tracing (PKT).

Affected by Natural Language Processing (NLP), researches on program analysis in early stage regard programming language as natural language and apply NLP based technology to program analysis, but different from natural language, syntax specification of programming language is more stringent. Therefore, some researchers proposes program analysis models based on Abstract Syntax Tree (AST), which converts program into AST for analysis. However, existing AST based program analysis models only consider the static information of program, including syntax structure, control flow or data flow, and ignore dynamic information, that is, real behavior of program when it is executed. This kind of methods can not strengthen influence of most frequently called core code blocks and weaken blocks that are rarely called in program on analysis results, which may make results inaccurate. This paper proposes Dynamics-Aware Gated Graph Attention Neural Network (DGGANN), which inputs test cases of questions into program, obtains call frequency coefficients of every node in AST through code coverage statistical tool, and introduces such call frequency information into process of program analysis. DGGANN is a generic architecture, and is applied to two tasks in our experiments: classifying programs by functionalities and KT. It outperforms state-of-the-art methods in both tasks.

In general, our contributions can be summarized as follows: (1) we propose a Dynamic Abstract Syntax Tree (DAST), which integrates call frequency coefficients of every node into AST to characterize dynamic information from program. (2) we improve the GGANN [1] model by introducing dynamic information to the attention mechanism of learning each neighbor node's weight in the program graph and aggregated representation of the whole program. (3) We evaluate the performance of pro-posed work by comparative experiments on program classification and KT.

2 Related Work

2.1 Program Analysis

In early stage, most related works [2, 3] of program analysis pay attention to processing program with NLP models. However, programs contain abundant and explicit structural information. To capture structural information, Mou et al. [4] propose Tree-Based Convolutional Neural Network (TBCNN) to transform ASTs into distributed vectors, which can preserve structural information. Li et al. [5] think that AST is a weakened graph, and propose Gated Graph Neural Network (GGNN) to process AST. Gated Graph Attention Neural Network (GGANN [1]) adds data flow and function call information of program into AST, introduces attention mechanism of learning each neighbor node's weight in the program graph and aggregated representation of the whole program.

2.2 Knowledge Tracing

Knowledge learning is a long-term process and follows principle of gradual improvement, so how to trace changes of students' knowledge mastery level in real time is of great significance for providing students with personalized learning guidance services. Considering above issue, Corbett et al. [6] declare the concept of Knowledge Tracing (KT), which is a technology aiming to trace changes of students' knowledge mastery level through exercising history, and propose Bayesian Knowledge Tracing (BKT) to predict students' performance in the future. Deep Knowledge Tracing (DKT [7]) estimates students' knowledge states based on Gated Recurrent Unit (GRU). Dynamic Key-Value Memory Network (DKVMN [8]) introduces two memory matrices to represent knowledge and learners' mastery level for each knowledge respectively. Exercise-correlated Knowledge Proficiency Tracing (EKPT [9]) uses time interval between students' learning and number of repeated learning to define memory and forgetting factor, so as to model students' learning and forgetting behaviors. Separated Self-Attentive Neural Knowledge Tracing (SAINT [10]) applies the encoder decoder structure to the knowledge tracking task for the first time, separating the question sequence and response sequence, so that the model can capture complex relationship between question and response through the deep self-attention mechanism. Programming Knowledge Tracing has applied KT to programming education. PKT-sequences [11] evaluates students' knowledge state by calculating tree edit distance between the program submitted by student and shortest correct program, and utilize such knowledge state to predict whether a student can solve the next exercise with the repeated submitted program to the current question, or can solve the current question at the next submission. ASTNN-attn [12] adopt an attention-based KT model to learn the features reflecting the multiple programming knowledge from the program submitted by student. Code-DKT [13] use code2vec model to learn a meaningful representation of student code, and combine this with DKT to track student progress.

3 Methodology

This section mainly presents the architecture of proposed work, as illustrated in Fig. 1, which consists of three main components: (1) constructing Dynamic Abstract Syntax Tree (DAST) by AST with Call Frequency Coefficient (CFC), Data Flow Graph (DFG) and Function Call Graph (FCG); (2) structural details of DGGANN model; (3) DGGANN's application on program classification and KT.

3.1 Construction of DAST

The construction of DAST includes three steps:

Firstly, input test case of the question to program submitted by student and get Call Numbers (CN) of each line in source code through code coverage statistics tool gcov. Figure 2 shows code coverage result of inputting test case "N = 5" to the C++ program for "finding the sum of 1 to N", where LN is line number.

Then, convert program into AST and obtain line number of each AST's node in the program by srcML, as shown in Fig. 3.

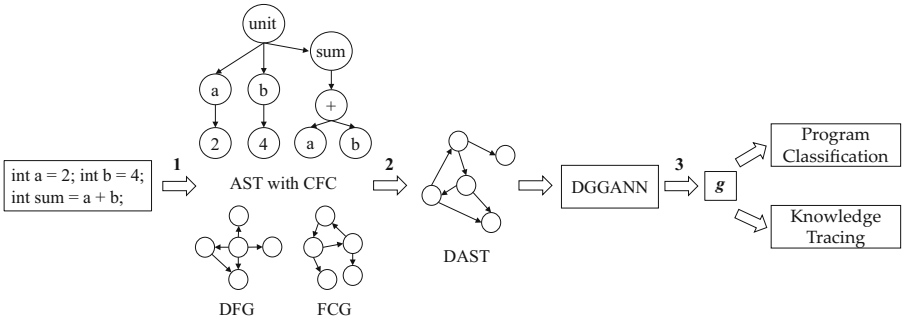

Fig. 1. The architecture of proposed work

```
CN LN
 1   1   int i, total, N;
 1   2   total = 0;
 1   3   scanf("%d",&N);
 1   4   if (N < 1)
 0   5     printf("illegal input\n");
 11  6   for (i=1; i<=maxNum; i++)
 10  7     total += i;
 1   8   printf("%d\n", total);
```

Fig. 2. Code coverage result example

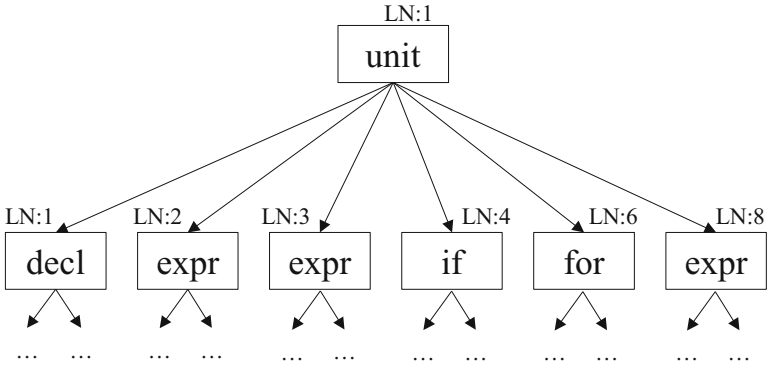

Fig. 3. AST example

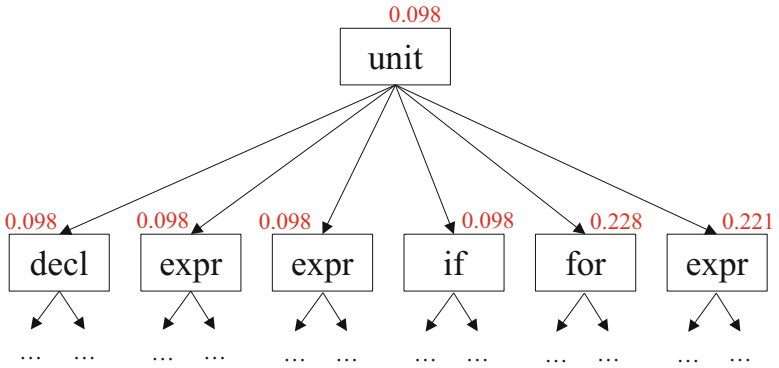

Fig. 4. DAST example

Finally, initialize Call Frequency Coefficient (CFC) of each node through Eq. (1):

$$CFC_i^{(0)} = \frac{\log(CN_{L_i} + 2)}{\sum_{L \in S} \log(CN_L + 2)} \tag{1}$$

where $CFC_i^{(0)}$ is initial CFC of node i, S is source code of program, L_i is the line which node i is in, CN_{L_i} is CN of L_i. Fuse CFC and AST to generate DAST, as shown in Fig. 4, where red number is initial CFC of node.

In addition, following GGANN, this paper extends DAST through Data Flow Graph (DFG) and Function Call Graph (FCG).

3.2 The DGGANN Model

This paper defines DAST as a directed graph $g = (V, E)$, where V and E are the sets of nodes and directed edges respectively. Node type set $VT = \{vt_1, vt_2, ..., vt_y\}$ is the set of different types of nodes, such as “unit”, “decl”, “expr”, “if” and “for” in Fig. 4.

Edge type set $ET = \{et_1, et_2, ..., et_z\}$ is the set of different types of relations between nodes, including parent-child relation in AST, function call relation and five types of data flow relations ("LastUse", "Compute", "Formal", "Return" and "Operand"). Node i is denoted as v_i, a directed edge from i to j is denoted as e_{ij}. Relations among nodes in graph g is expressed by a connection matrix $\mathbf{A} \in \mathbb{R}^{|V|\times|V|}$, the element at row i and column j of $\mathbf{A}$ is expressed as $\mathbf{A_{ij}}$. $\mathbf{A_{ij}}$ is a $d_v \times d_v$ matrix for representing e_{ij}, where d_v is the representation dimension.

GGANN initializes each node i's representation according to its type. We first set it to be a one-hot encoding $\mathbf{v_i^{'}} \in \{0, 1\}^y$, then introduce the embedding matrix $\mathbf{W_v} \in \mathbb{R}^{d_v \times y}$ to transfer it into a low-dimensional vector $\mathbf{v_i^{(0)}} \in \mathbb{R}^{d_v}$ by Eq. (2):

$$\mathbf{v_i^{(0)}} = \mathbf{W_v v_i^{'}} \tag{2}$$

For e_{ij}'s representation, get one-hot encoding $\mathbf{e_{ij}} \in \{0, 1\}^z$ in term of its type firstly. After that, transfer $\mathbf{e_{ij}}$ into $\mathbf{A_{ij}} \in \mathbb{R}^{d_v \times d_v}$ with tensor $\mathbf{W_e} \in \mathbb{R}^{d_v \times d_v \times z}$ by Eq. (3):

$$\mathbf{A_{ij}} = \mathbf{W_e e_{ij}} \tag{3}$$

DGGANN updates each node's representation by aggregating it and its neighbors' representations for T iterations with Eq. (4) and Eq. (5):

$$\mathbf{m_i^{(t)}} = \sum_{j \in N_i} \alpha_{ij}^{(t)} \cdot \mathbf{A_{ij}} \cdot \mathbf{v_j^{(t)}} \tag{4}$$

$$\mathbf{v_i^{(t)}} = GRU(\mathbf{v_i^{(t-1)}}, \mathbf{m_i^{(t)}}) \tag{5}$$

where N_i is the set of node i's neighbors, $\mathbf{m_i^{(t)}} \in \mathbb{R}^{d_v}$ is node i's context representation, $\alpha_{ij}^{(t)} \in (0, 1)$ is node j's importance coefficient for node i, which is calculated by Eq. (6) and Eq. (7):

$$\alpha_{ij}^{(t)} = \beta \cdot sigmoid[(\mathbf{v_i^{(t)}})^{\mathbf{T}} \mathbf{W_{att}} \mathbf{v_j^{(t)}} + b_{att}] + (1 - \beta) \cdot CFC_j^{(t)} \tag{6}$$

$$CFC_j^{(t)} = sigmoid(w_c^{(t)} \cdot CFC_j^{(t-1)} + b_c^{(t)}) \tag{7}$$

where $\mathbf{W_{att}} \in \mathbb{R}^{d_v \times d_v}$ is weight matrix, $b_{att} \in \mathbb{R}$ is bias term, $sigmoid[(\mathbf{v_i^{(t)}})^{\mathbf{T}} \mathbf{W_{att}} \mathbf{v_j^{(t)}} + b_{att}]$ means comparing the similarity between node i and node j. $w_c^{(t)} \in \mathbb{R}$ is weight parameter, $b_c^{(t)} \in \mathbb{R}$ is bias term, $sigmoid(w_c^{(t)} \cdot CFC_j^{(t-1)} + b_c^{(t)})$ is used to calculate dynamic factor of node j for node i in t-th iteration, which neighbor is called more frequently, the value of dynamic factor and importance coefficient are larger. $\beta \in (0, 1)$ is a weight parameter for balancing similarity and dynamic factors.

The representation of DAST graph can be calculated after obtaining each node's embedding vector, as shown in Eq. (8).

$$\mathbf{g} = \sum_{i \in V} [\gamma \cdot f_1(\mathbf{v_i^{(T)}}, \mathbf{v_i^{(1)}}) + (1 - \gamma) CFC_i^{(T)}] \cdot f_2(\mathbf{v_i^{(T)}}) \tag{8}$$

where f_1 represents a neural network to implement attention mechanism, the output of f_1 is probability (a scalar) of node i being fused. $\gamma \in (0, 1)$ is a weight parameter for balancing fusion probability and dynamic factors. f_2 means a neural network to learn embedding vector of node. f_1 and f_2 use sigmoid and tanh to activate outputs respectively. $\mathbf{g} \in \mathbb{R}^{d_v}$ is vector representation of DAST.

3.3 Program Classification

After vector representation of DAST graph is obtained, probabilities $\mathbf{p} \in \mathbb{R}^{|PT|}$ (PT is the set of program types) that the program associated with DAST graph be-longs to each category can be predicted with Eq. (9):

$$\mathbf{p} = softmax(\mathbf{W_g g} + \mathbf{b_g}) \tag{9}$$

where $\mathbf{W_g} \in \mathbb{R}^{|PT| \times d_v}$ is weight matrix, $\mathbf{b_g} \in \mathbb{R}^{|PT|}$ is bias vector.

To optimize program classification model, we update the parameters using gradient descent by minimizing cross entropy loss between predicted probability and true label of the programs' category.

$$L_{PC} = -\sum_{g \in G} \sum_{pt \in PT} y_{pt} \log(p_{pt}) \tag{10}$$

3.4 Knowledge Tracing

Suppose there are $|U|$ students, $|Q|$ questions and $|K|$ knowledge points in an OJ system. Student u's historical interaction sequence is $I^u = \{(q_1, l_1), (q_2, l_2), ..., (q_T, l_T)\}$, where q_t ($q_t \in Q$) is a question that the student u solved at time t. l_t is system judgment result, which $l_t = 1$ and $l_t = 0$ mean judgement result is correct and wrong respectively. KT aims at predicting whether student is able to answer the next question q_{t+1} correctly.

In this paper, we also apply DGGANN to Knowledge Tracing (KT). The KT model using DGGANN to encode the program of questions is called DGGANN-KT. In addition to question description, knowledge point and difficulty, DGGANN-KT also takes optimal solution of question (the correct and shortest running program in submission records corresponding to the question) and program submitted by students as features of questions.

For question q_t's optimal solution and program submitted by students, DGGANN-KT converts them into DAST g_t and g_t^s respectively, input them to DGGANN for obtaining vector representation of program $\mathbf{g_t}$ and $\mathbf{g_t^s}$, this process is shown in Eq. (2) to Eq. (8).

For the word sequence $c_t = \{w_1, w_2, ..., w_M\}$ of question q_t's description, DGGANN-KT uses Word2Vec to generate vector presentation $\mathbf{w_i} \in \mathbb{R}^{d_w}$ of each word w_i. Input $\mathbf{w_i}$ into LSTM to get embedding vector of question description $\mathbf{c_t} \in \mathbb{R}^{d_c}$, as shown in Eq. (11).

$$\mathbf{c_t} = LSTM(\mathbf{w_1}, \mathbf{w_2}, ..., \mathbf{w_M}; \Phi_{LSTM}) \tag{11}$$

where Φ_{LSTM} is LSTM's related parameters.

For knowledge points k_t examined in question q_t, DGGANN-KT obtains its multi-hot representation $\mathbf{k_t}^{'} \in \{0, 1\}^{|K|}$, and uses the parameter matrix $\mathbf{W_k} \in \mathbb{R}^{d_k \times |K|}$ to convert it into a low dimensional embedding vector $\mathbf{k_t} \in \mathbb{R}^{d_k}$ of knowledge points k_t, as shown in Eq. (12):

$$\mathbf{k_t} = \frac{\mathbf{W_k}\mathbf{k_t}^{'}}{|k_t|} \tag{12}$$

DGGANN-KT selects the student's answer error rate r_t^a and submission error rate r_t^s as features to evaluate the difficulty of question q_t. The student's answer error rate r_t^a is the proportion of students who answer q_t incorrectly among students who try to solve q_t. Submission error rate r_t^s is the proportion of the submission records with wrong judgment results in the submission records of q_t.

In summary, DGGANN-KT chooses above vectors and values to encode question as $\mathbf{x_t}$ with Eq. (13):

$$\mathbf{x_t} = \mathbf{g_t} \oplus \mathbf{g_t^s} \oplus \mathbf{c_t} \oplus \mathbf{k_t} \oplus [r_t^a, r_t^s] \tag{13}$$

After obtaining embedding vector $\mathbf{x_t}$ of question q_t, according to judgement result l_t given by OJ, the vector representation of question considering judgement result is acquired by using Eq. (14):

$$\mathbf{lx_t} = \begin{cases} \mathbf{x_t} \oplus \mathbf{0} & if l_t = 1 \\ \mathbf{0} \oplus \mathbf{x_t} & if\ l_t = 0 \end{cases} \tag{14}$$

where $\mathbf{0}$= [0, 0,..., 0] is a zero vector with the same dimensions of $\mathbf{x_t}$, $\oplus$ is vector concatenation operator.

Then DGGANN-KT uses $\mathbf{lx_t}$ as input to GRU for estimating student knowledge hidden states:

$$\mathbf{h_t} = GRU(\mathbf{h_{t-1}}, \mathbf{lx_t}; \Phi_{GRU}) \tag{15}$$

where Φ_{GRU} is GRU's related parameters.

DGGANN-KT inputs embedding vectors of student knowledge hidden states $\mathbf{h_t}$ and questions $\mathbf{x_{t+1}}$ into a fully connected layer for predicting whether student is able to answer the question q_{t+1} correctly:

$$p_{t+1} = sigmoid(\mathbf{W_q^T}(\mathbf{h_t} \oplus \mathbf{x_{t+1}}) + b_q) \tag{16}$$

where $\mathbf{W_q} \in \mathbb{R}^{d_h + d_x}$ is a weight matrix, $b_q \in \mathbb{R}$ is a bias term, p_{t+1} is the predicted probability that the student can answer question q_{t+1} correctly.

To optimize DGGANN-KT, we update the parameters using gradient descent by minizing the cross entropy loss between predicted probability of answering correctly and true label of student's answer:

$$L_{KT} = \sum_{u \in U} \sum_{t=1}^{T} [l_t \log p_t + (1 - l_t) \log(1 - p_t)] \tag{17}$$

Table 1. Basic information of datasets

Dataset	Number of Questions	Number of Students	Number of Submissions
CodeForces	7008	26787	1048575
Libre	2538	24657	1382200

4 Result

This section evaluates the effectiveness of DGGANN from the perspective of pro-gram classification and KT.

4.1 Dataset

This paper collects experimental data from two well-known OJ systems named CodeForces and Libre. Both datasets contain question information and user submission records. Each data in question information represents information of a question, including following fields: question ID, title, question description and knowledge point label (possibly missing). Each data in user submission records contains fields such as submission ID, user ID, question ID, system judgement result (correct/wrong), program, programming language type, running time and submission stamp. The basic information of CodeForces and Libre is shown in Table 1.

For program classification task, this paper randomly selects 40 questions with more than 500 solutions from CodeForces and Libre (In order to exclude the influence of programming languages, this paper only considers the programs written in C++), then randomly chooses 500 solutions from these questions to form two standard datasets containing 20000 data samples respectively. Each dataset is divided into training set, verification set and test set according to the ratio of 3:1:1.

For knowledge tracing task, 80% of the students are randomly selected from each dataset to train KT model, 10% of the students are used for verification, and 10% of the students are used for testing.

4.2 Experiment Setup

The proposed method in this paper is implemented by PyTorch, the optimizer adopts Adam, batch size bs is 128, learning rate lr is 0.001, dimension of program's embedding vector d_v, dimension of word's embedding vector d_w, dimension of question description's embedding vector d_c, dimension of knowledge point's embedding vector d_k and dimension of knowledge hidden state's embedding vector d_h are set to 256, 64, 128, 16 and 128.

4.3 Experiment Results

This subsection mainly analyzes the student program classification and knowledge tracing performance associated with the proposed DGGANN model.

Table 2. The accuracy of program classification models

Model	Test Accuracy	
	CodeForces	**Libre**
TBCNN (2016)	0.9413	0.8679
GGNN (2016)	0.9608	0.8824
GGANN (2021)	0.9725	0.9016
DGGANN	**0.9861**	**0.9253**

4.3.1 Program Classification

The experiments for student program classification are used to verify whether the proposed work can successfully learn the grammatical structure and semantic information from source codes, i.e., whether the submitted codes implemented by different students for the same programming task can be classified into the same category.

Our experiment applies DGGANN to classify program in the OJ system. The target label of a program is one of 40 questions (represented as an ID). To evaluate the performance of DGGANN, this paper selects TBCNN, GGNN, GGANN as the baseline model. Table 2 shows the accuracy of four models applied on two test sets. From Table 2, we can infer that due to considering dynamic information of running program, DGGANN's performance in program classification task is better than baseline model.

4.3.2 Knowledge Tracing

In order to evaluate the effectiveness of DGGANN-KT, this paper selects DKT, DKVMN, SAINT, ASTNN-attn, Code-DKT, TBCNN-KT (replacing DGGANN in DGGANN-KT with TBCNN), GGNN-KT (replacing DGGANN in DGGANN-KT with GGNN) and GGANN-KT (replacing DGGANN in DGGANN-KT with GGANN) as baseline models for knowledge tracing task, and conducts comparative experiments with DGGANN-KT. The experimental results are shown in Table 3.

It can be seen from Table 3 that since ASTNN-attn, Code-DKT, TBCNN-KT, GGNN-KT, GGANN-KT and DGGANN-KT are able to mine the correlation between questions through programs, their AUC values are higher than DKT, DKVMN and SAINT. DGGANN-KT considers the dynamic information of the program, so its performance is better than ASTNN-attn, Code-DKT, TBCNN-KT, GGNN-KT and GGANN-KT. Futhermore, it should be noted that the AUC values of DKT, DKVMN and SAINT on Libre have decreased by about 6% compared with CodeForces, while the AUC values of ASTNN-attn, Code-DKT, TBCNN-KT, GGNN-KT, GGANN-KT and DGGANN-KT have decreased by about 3%, which are relatively smaller. According to analysis result, the reason for this phenomenon is that 34.08% of questions in Libre dataset do not have knowledge point labels. If the knowledge tracing model does not take the program as feature, it will be difficult to mine the correlation among these questions without knowledge point labels and the performance of knowledge tracing model will be reduced. In

Table 3. The AUC of knowledge tracing models

Model	AUC	
	CodeForces	**Libre**
DKT (2015)	0.7198	0.6581
DKVMN (2017)	0.7235	0.6627
SAINT (2020)	0.7446	0.6703
ASTNN-attn (2022)	0.7534	0.7294
Code-DKT (2022)	0.7485	0.7151
TBCNN-KT	0.7680	0.7356
GGNN-KT	0.7752	0.7433
GGANN-KT	0.7803	0.7490
DGGANN-KT	**0.7897**	**0.7618**

the CodeForces dataset, only 4.62% of the questions do not have knowledge point labels, making the impact of not using the programs of questions smaller.

4.3.3 Impact Analysis of Important Hyperparameter

This section studies the impact of different program embedding dimensions on performance of program classification and KT. The value set of this hyperparameter is {16, 32, 64, 128, 256, 512}. Figure 5 and Fig. 6 show variation curve of test accuracy on program classification and AUC on KT under different program embedding dimensions.

From Fig. 5 and Fig. 6, it can be seen that with the increase of dimensions, test accuracy and AUC of model also increases, reaching the best at 256. When dimension

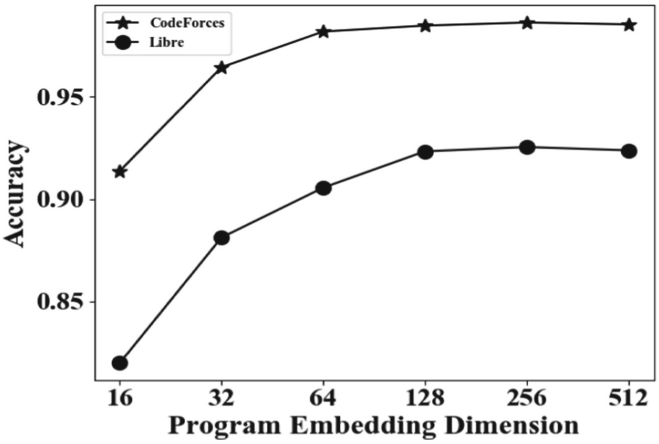

Fig. 5. Test Accuracy of Setting Different Program Embedding Dimensions

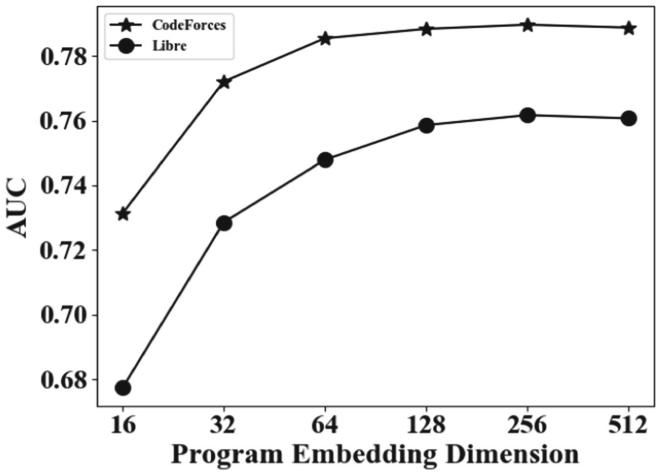

Fig. 6. AUC of Setting Different Program Embedding Dimensions

is increased to 512, the performance of model decreases, because oversized dimension will lead to overfitting and reduce generation ability of the model.

5 Conclusion

In order to solve the problems that quite a few OJ questions lack knowledge point labels and the current AST based program analysis model does not consider dynamic information of program, this paper proposes the DGGANN model, which inputs test cases of questions into program, obtains call frequency coefficients of every node in AST through code coverage statistical tool, and introduces such call frequency information into process of program analysis. We apply DGGANN to two tasks in our experiments: classifying programs by functionalities and KT. It outperforms state-of-the-art methods in both tasks.

Although this paper has tackled the problem of lacking knowledge point labels to some extent, there are still many elements in OJ that can have negative impacts on the adaptive leaning model. For example, when students encounter exercises that they can not solve, they often refer and clone codes submitted by other students or codes in blogs about the exercises. Cloning behavior may cause KT model to incorrectly evaluate students' knowledge state. In future, we plan to propose a KT model, which can detect and reduce the impact of students' code cloning behavior on knowledge state assessment.

Acknowledgments. This work was supported by the Young Scientists Fund of the National Natural Science Foundation of China (No. 61907015) and the Shanghai Committee of Science and Technology, China (No. 20511102502).

References

1. Lu M, Wang Y, Tan D, et al. Student Program Classification Using Gated Graph Attention Neural Network[J]. IEEE Access, 2021, 9: 87857–87868.
2. Allamanis M, Peng H, Sutton C. A convolutional attention network for extreme summarization of source code[C]//International conference on machine learning. PMLR, 2016: 2091–2100.
3. Lu Y, Li G, Zhao Z, et al. Learning to infer API mappings from API documents[C]//International Conference on Knowledge Science, Engineering and Management. Springer, Cham, 2017: 237–248.
4. Mou L, Li G, Zhang L, et al. Convolutional neural networks over tree structures for programming language processing[C]//Thirtieth AAAI conference on artificial intelligence. 2016.
5. Li Y, Tarlow D, Brockschmidt M, et al. Gated graph sequence neural networks[C]//Proceedings of the 4th International Conference on Learning Representations. Puerto Rico, OpenReview, 2016.
6. Corbett A T, Anderson J R. Knowledge tracing: Modeling the acquisition of procedural knowledge[J]. User modeling and user-adapted interaction, 1994, 4(4): 253–278.
7. Piech C, Bassen J, Huang J, et al. Deep knowledge tracing[C]//Proceedings of the Advances in Neural Information Processing Systems. Quebec, MIT Press, 2015: 505–513.
8. Zhang J, Shi X, King I, et al. Dynamic key-value memory networks for knowledge tracing[C]//Proceedings of the 26th international conference on World Wide Web. 2017: 765–774.
9. Huang Z, Liu Q, Chen Y, et al. Learning or forgetting? a dynamic approach for tracking the knowledge proficiency of students[J]. ACM Transactions on Information Systems (TOIS), 2020, 38(2): 1–33.
10. Choi Y, Lee Y, Cho J, et al. Towards an appropriate query, key, and value computation for knowledge tracing[C]//Proceedings of the Seventh ACM Conference on Learning@ Scale. 2020: 341–344.
11. Jiang B, Wu S, Yin C, et al. Knowledge tracing within single programming practice using problem-solving process data[J]. IEEE Transactions on Learning Technologies, 2020, 13(4): 822–832.
12. Zhu M, Han S, Yuan P, et al. Enhancing Programming Knowledge Tracing by Interacting Programming Skills and Student Code[C]//LAK22: 12th International Learning Analytics and Knowledge Conference. 2022: 438–443.
13. Shi Y, Chi M, Barnes T, et al. Code-DKT: A Code-based Knowledge Tracing Model for Programming Tasks[J]. arXiv preprint arXiv:2206.03545, 2022.