An Edge-Enhanced Hierarchical Graph-to-Tree Network for Math Word Problem Solving

Qinzhuo Wu, Qi Zhang*, Zhongyu Wei
Shanghai Key Laboratory of Intelligent Information Processing,
School of Computer Science, Fudan University, Shanghai, China
(qzwu17,qz,zywei)@fudan.edu.cn

Abstract

Math word problem solving has attracted considerable research interest in recent years. Previous works have shown the effectiveness of utilizing graph neural networks to capture the relationships in the problem. However, these works did not carefully take the edge label information and the long-range word relationship across sentences into consideration. In addition, during generation, they focus on the most relevant areas of the currently generated word, while neglecting the rest of the problem. In this paper, we propose a novel Edge-Enhanced Hierarchical Graph-to-Tree model (EEH-G2T), in which the math word problems are represented as edge-labeled graphs. Specifically, an edge-enhanced hierarchical graph encoder is used to incorporate edge label information. This encoder updates the graph nodes hierarchically in two steps: sentence-level aggregation and problem-level aggregation. Furthermore, a tree-structured decoder with a split attention mechanism is applied to guide the model to pay attention to different parts of the input problem. Experimental results on the MAWPS and Math23K dataset showed that our EEH-G2T can effectively improve performance compared with state-of-the-art methods. [1]

1 Introduction

Math word problem solving is an important natural language processing (NLP) task that has recently been attracting increasing research interests. Math word problems are narrative text that describe a scene with several math variables and ask a question about an unknown quantity. A simple example is illustrated in Figure 1. Based on the given problem, the target is to infer the difference between the number of boxes of apples and pears.

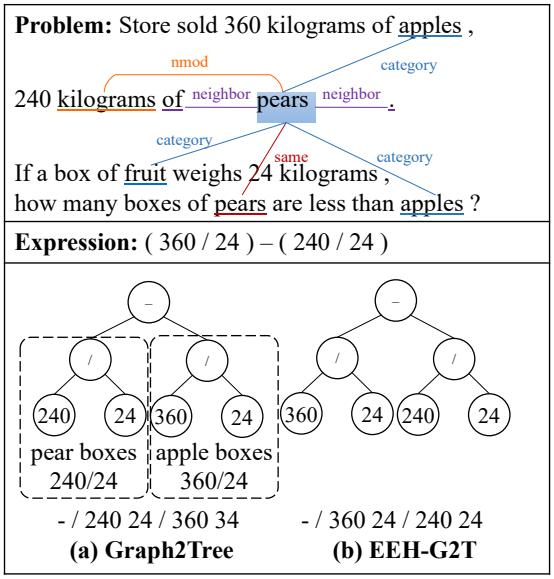
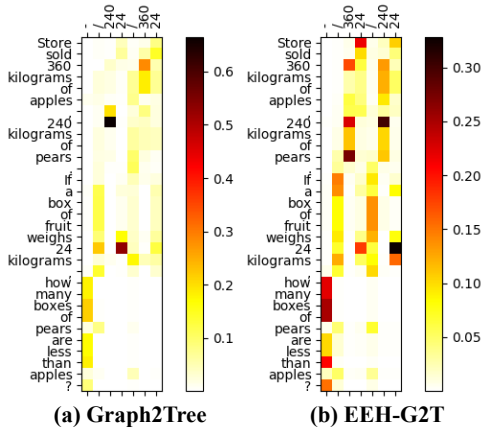


Figure 1: An example of a math word problem. The top part of the figure shows the different types of edges connected to the word "pear" in the graph. The bottom part of the figure shows the expressions generated by Graph2Tree (Zhang et al., 2020b) and EEH-G2T.

Previous works (Wang et al., 2017; Huang et al., 2018; Wang et al., 2019) used sequence-to-sequence (seq2seq) methods with an attention mechanism (Bahdanau et al., 2014) to generate math expression sequences from math word problems. To capture the structural information of math expressions, many works (Liu et al., 2019; Xie and Sun, 2019; Zhang et al., 2020a) treat math expressions as binary trees and propose several sequence-to-tree (seq2tree) frameworks. These tasks are designed to obtain the pre-order sequence of the expression tree, and they generate the current node based on its parent node and sibling node at each time step. Some works that represent problems as graphs also show better performance. Graph2Tree (Zhang et al., 2020b) connects each number in the problem with its nearby nouns to enrich the quantity representations. KA-S2T (Wu

* Corresponding author.
[1]Code is available at https://github.com/qinzhuowu/EEH_G2T



(a) Graph2Tree **(b) EEH-G2T**

Figure 2: The attention matrices of Graph2Tree and EEH-G2T. Each row corresponds to a word in the problem, and each column corresponds to a word in the expression. The darker areas of the matrix indicate higher attention scores.

et al., 2020) connects words with its category in the external knowledge base to capture common sense information.

Although these methods report promising results, several challenges still remain. 1) Long-range word relationships across sentences should be taken into consideration. As shown in Figure 1, the word "pear" in the second sentence should be associated to the word "pear" in the last sentence. Without long-range relationships, it is difficult for the model to connect these two words that are 15 steps apart. 2) Previous methods did not carefully take the edge label information into consideration. In figure 1, the label on the edge between "kilograms" and "pear" is nmod (noun compound modifier), while the label "category" on the edge between "apples" and "pears" means they belong to the same category in the external knowledge base. Such edge labels can also provide rich syntactic and semantic information. 3) When generating expressions, previous methods tend to focus on the areas in the problem that are most relevant to the currently generated words, and ignore the semantic clues provided by the rest of the problem. As shown in Figure 2, to generate "360" instead of "240" at time step 3, the model needs to pay attention to the entire problem to obtain important clues that the current sub-expression "/360 24" is the number of apple boxes and 360 is the weight of the apples. However, previous methods focused on the problem areas that are most relevant to the currently generated word (i.e., the number 360 itself), without noticing the rest of the problem.

To tackle these challenges, we propose a novel Edge-Enhanced Hierarchical Graph-to-Tree framework (EEH-G2T) for math word problem solving. EEH-G2T represents each math word problem as a graph in which the nodes are connected by labeled edges. To obtain the edge-aware problem representations, we propose an edge-enhanced hierarchical graph encoder that explicitly incorporates edge label information. In addition, the hierarchical encoder updates the nodes in two steps: sentence-level aggregation and problem-level aggregation. This hierarchical structure can first capture the local relations between words within the sentence and then capture the long-range dependencies between words across sentences. Further, we use a split attention mechanism to guide the decoder to pay attention to different parts of the entire input problem, not just the most relevant part of the currently generated word.

The main contributions of this paper can be summarized as follows:

- We propose an edge-enhanced hierarchical graph encoder to incorporate edge label information. Additionally, the encoder updates the graph nodes in two steps, namely sentence-level aggregation and problem-level aggregation.
- We propose a split attention mechanism to guide the decoder to pay attention to different parts of the entire input problem during the generation.
- We conducted experiments on two commonly used math word problem solving datasets, MAWPS and Math23K. Experimental results prove that our approach can effectively improve the performance compared with state-of-the-art methods.

2 Models

2.1 Problem Formulation

In this work, we focus on generating math expressions for the given math word problems. We denote the text of a math word problem as a sequence of words and number symbols. X=$(x_1, x_2, \ldots, x_m)$ is a math word problem with m words. Our model aims to generate a math expression Y=$(y_1, y_2, \ldots, y_T)$. Here, Y is a pre-order traversal

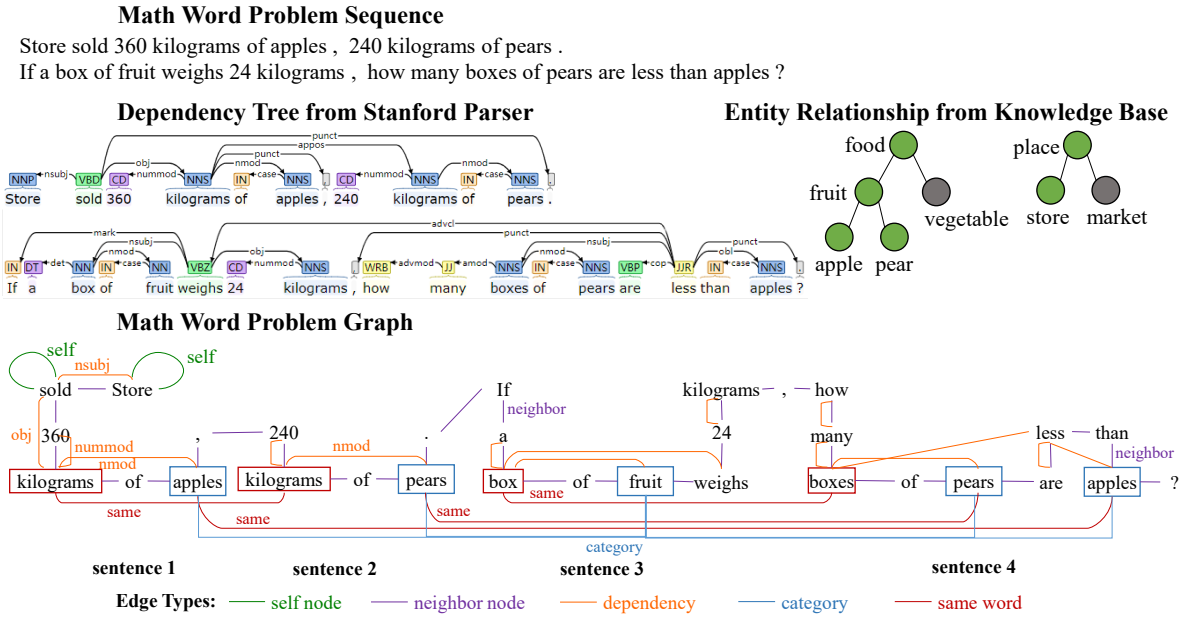


Figure 3: The procedure for construction of a edge-labeled graph is described here. For brevity, we omit some self-node edges and the labels of some neighbor edges and dependency edges. Given a math word problem, we first use the Stanford Corenlp toolkit to parse it into a dependency tree, and extract the relationships between nouns from the external knowledge bases. Based on these, we construct the edge-labeled graph, as shown in the bottom part of the figure (See Section 2.2 for more details).

sequence of a math expression tree, which can be executed to generate the answer to problem X.

Formally, math word problem X can be represented by a graph G = (V, E), where V and E are the set of nodes x_i and the set of edges e_{ij}. Here, each node in the graph is associated with a word x_i in the problem. $e_{ij} \in$ E denotes that there is an edge between the node pair (x_i, x_j). $L(e_{ij})$ denotes the label of edge e_{ij} (e.g., self-node, category, neighbor), see section 2.2 for more details.

2.2 Edge-labeled Graph

2.2.1 Graph Construction

This section introduces how to construct an edge-labeled graph that contains both the local relations between nodes within a sentence and the long-range relations between nodes across sentences. Our model extracts these relations from the problem's dependency tree and external knowledge base. We use the Stanford Corenlp toolkit [2] (Manning et al., 2014) to parse each math word problem into a dependency tree. The toolkit analyzes the grammatical structure of a sentence and establishes relationships between "head" words and words which modify those heads. In addition, inspired by Wu et al. (2020), we collected word category information from external knowledge bases. An illustrative example is shown in Figure 3. Specifically, given a math word problem X, its dependency tree, and word category information, our model constructs a graph according to the following steps.

- **Self node & Neighbor:** We define each word x_i in the problem X as a node. Each word node x_i is connected to its adjacent word nodes (x_{i-1}, x_{i+1}) in the problem. These edges are labeled as "neighbor". Also, to incorporate the node's own information into the problem representations, we connect each node to itself and label the edge as "self node".

- **Dependency (edges within sentences):** The dependency tree is a structured representation that contains various grammatical relationships between word pairs. Following Zhang et al. (2020b), we prune the output dependency tree to remove unimportant components, that is, remove edges connected to conjunctions, prepositions or punctuation. Based on the dependency tree, we establish relationships between nodes within the sentence, and keep the edge labels (e.g., nmod, nummod, appos). For example, "360"

[2] https://stanfordnlp.github.io/CoreNLP/

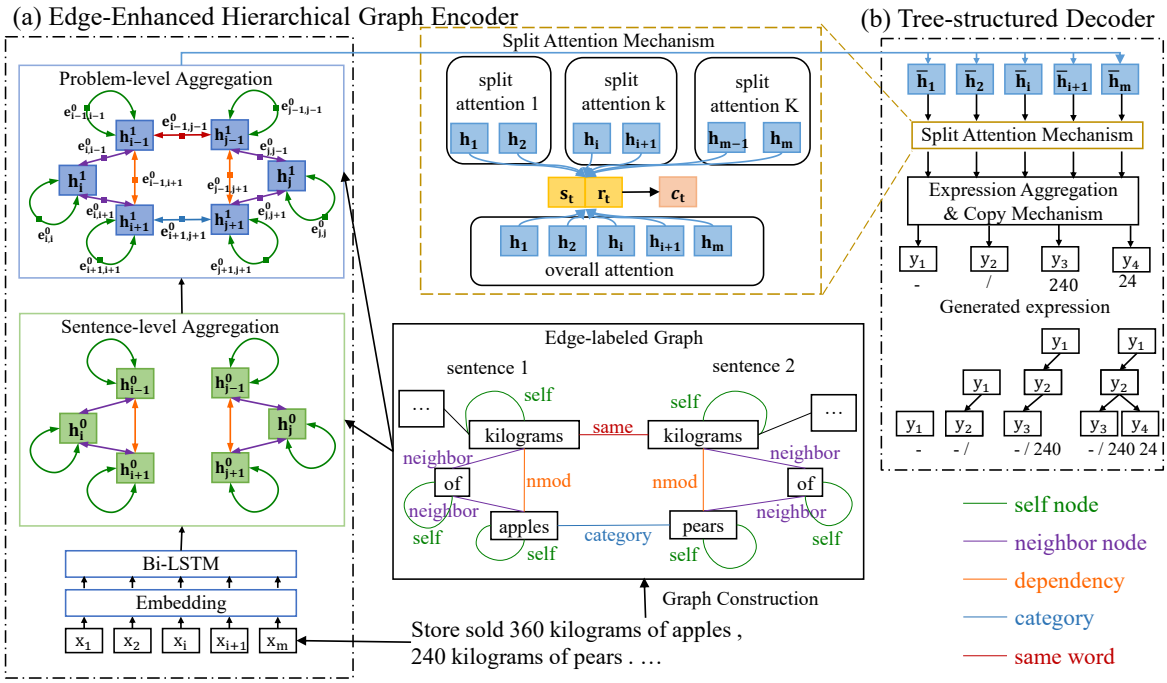


Figure 4: Main structure of our proposed EEH-S2T model. In Section 2.2, we first introduce how to construct and initialize an edge-labeled graph for a math word word problem. The left side of this figure shows (a) an edge-enhanced hierarchical graph encoder that updates the graph nodes in two steps, namely sentence-level aggregation and problem-level aggregation (Section 2.3). The right side of this figure shows (b) a tree-structured decoder. The decoder uses a split attention mechanism to guide the decoder to pay attention to different parts of the entire input problem during generation (Section 2.4).

and "kilograms" are connected by the edge "nummod" in Figure 3.

- **Same & Category (edges across sentences):** To further capture the connection across sentences, if the same word exists in two sentences and it is a noun, then we connect these two nodes and label the edge as "same". If two words belong to the same category in the external knowledge base, we also add a connection for their nodes and label the edge as "category". For example, "apples" and "pears" are connected by the edge "category" in Figure 3.

2.2.2 Graph Initialization

To initialize the node representations of the graph, we use a BiLSTM (Hochreiter and Schmidhuber, 1997) to encode the words in the math word problem X=$(x_1, x_2, \ldots, x_m)$. Here, $\mathbf{H^0} = (\mathbf{h_1^0}, \mathbf{h_2^0}, \ldots, \mathbf{h_m^0}) \in \mathbb{R}^{m*d}$ is the initial node representations of its graph G, where m is the number of nodes and d is the dimension of the node representation. The representation $\mathbf{h_i^0}$ of node x_i is calculated as follows:

$$\mathbf{h_i^0} = \text{BiLSTM}(\text{Embed}(x_i), \mathbf{h_{i-1}^0}), \quad (1)$$

where $\text{Embed}(\cdot)$ is an embedding layer.

For each edge e_{ij}, we initialize the edge representation $\mathbf{e_{ij}^0}$ based on the edge embedding and its neighbor node representations $\mathbf{h_i^0}, \mathbf{h_j^0}$:

$$\mathbf{e_{ij}^0} = \text{W}_\text{e}[\text{Embed}(e_{ij}) : \mathbf{h_i^0} : \mathbf{h_j^0}], \quad (2)$$

where W_e is a weight matrix and [:] is the concatenation operation.

2.3 Edge-Enhanced Hierarchical Graph Encoder

After initializing the graph, EEH-G2T uses an edge-enhanced hierarchical graph encoder to obtain the edge-aware problem representations. It hierarchically updates the nodes in two steps: sentence-level aggregation and problem-level aggregation. We divide math word problems into short sentences based on commas and periods. For example, the problem in Figure 1 has four sentences.

Sentence-level Aggregation.

To capture the local relations between words, EEH-G2T first recursively aggregate the node representation with its related nodes within the sentence. Let A denote the local relationship matrix, where $\mathrm{A_{ij}} \in \{0, 1\}$ denotes whether there is an edge between x_i and x_j. Formally, $\mathrm{A_{ij}} = 1$ if $e_{ij} \in \mathrm{E}$ and x_i, x_j in the same sentence, otherwise $\mathrm{A_{ij}} = 0$. The initial node representations $\mathbf{H^0} = (\mathbf{h_1^0}, \mathbf{h_2^0}, \ldots, \mathbf{h_m^0})$ are aggregated with a two-layer graph convolutional network (GCN) (Kipf and Welling, 2017). The aggregation functions are as follows:

$$\mathbf{H^1} = \sigma(\mathrm{A}\ \mathbf{H^0} \mathrm{W_g}). \tag{3}$$

Here, $\mathrm{W_g}$ is a weight matrix and σ is a relu activate function. After sentence-level aggregation, we obtain the node representations $\mathbf{H^1} = (\mathbf{h_1^1}, \mathbf{h_2^1}, \ldots, \mathbf{h_m^1})$.

Problem-level Aggregation.

Then, EEH-G2T use an attentive problem-level aggregation to capture long-range dependencies across sentences. Inspired by GAT (Veličković et al., 2018), we use the multi-head attention in GAT with M independent attention mechanisms:

$$\begin{aligned} \beta_{ij} &= \sigma(w_a^T[\mathrm{W_a}\mathbf{h_i^l} : \mathrm{W_b}\mathbf{h_j^l} : \mathrm{W_c}\mathbf{e_{ij}^0}]), \\ \alpha_{ij} &= \frac{\exp(\beta_{\mathrm{ij}})}{\sum_{e_{ij}\in \mathrm{E}} \exp(\beta_{\mathrm{ij}})}, \\ \mathbf{h_i} &= \Big\Vert_{1,\ldots,\mathrm{M}} \sum_{e_{ij}\in \mathrm{E}} \alpha_{ij} \mathrm{W_j} \mathbf{h_j^1}. \end{aligned} \tag{4}$$

Here, w_a^T, $\mathrm{W_a}$, $\mathrm{W_b}$, $\mathrm{W_c}$, $\mathrm{W_j}$ are weight vector and matrices. σ is a LeakyRelu activate function (Xu et al., 2015). $||$ is the concatenation operation. α_{ij} is the normalized attention weight of the node x_j for node x_i via the softmax function. After problem-level aggregation, we obtain the final problem representations $\mathbf{H} = (\mathbf{h_1}, \mathbf{h_2}, \ldots, \mathbf{h_m})$.

2.4 Tree-structured Decoder

The structure of the decoder is similar to other state-of-the-art Seq2Tree models (Xie and Sun, 2019; Zhang et al., 2020b; Wu et al., 2020). The decoder is an attention-based Gated Recurrent Unit (GRU) (Chung et al., 2014) whose goal is to generate pre-order traversal of expression trees. The hidden state $\mathbf{s_t}$ is updated as follows:

$$\mathbf{s_{t+1}} = \mathrm{BiLSTM}([\mathrm{Embed}(y_t) : \mathbf{c_t} : \mathbf{r_t}], \mathbf{s_t}). \tag{5}$$

At time step 1, we use the last problem representations $\mathbf{h_m}$ to initialize the decoder hidden state s_1. Here, $\mathrm{Embed}(y_t)$ denotes the embedding of the last generated word y_t; $\mathbf{c_t}$ denotes the context state of the problem representations, and $\mathbf{r_t}$ denotes the context state of the currently generated expression.

Split Attention Mechanism.

Figure 4 shows the input of our proposed split attention mechanism, which is the final problem representations of the graph encoder. EEH-G2T first uses an attention mechanism to compute the overall attention vector $\hat{\alpha}$ on the problem representations. Then, EEH-G2T divides the input math word problem into K parts, conducts attention operations on each part, and obtains K split attention vectors $(\alpha^1, \alpha^2, \ldots, \alpha^K)$. The size of each split attention vector is $\mathbb{R}^{(m/K)}$. In Figure 1, when the decoder generates y_3=360, EEH-G2T notices that the word most relevant to the current decoder state is "360" in the first sentence. At the same time, EEH-G2T obtains crucial semantic clues from the other parts, that is, the problem asks how many pear boxes are less than the apple boxes. Based on K attention vectors, the problem context state $\mathbf{c_t}$ is calculated as follows:

$$\begin{aligned} \hat{\alpha}_{ti} &= \mathrm{softmax}(\mathrm{W_s}[\mathbf{s_t} : \mathbf{r_t}] + \mathrm{W_h}\mathbf{h_i}), \\ \alpha_{ti}^k &= \mathrm{softmax}(\mathrm{W_s}[\mathbf{s_t} : \mathbf{r_t}] + \mathrm{W_h}\mathbf{h_{i+k\frac{m}{K}}}), \\ \mathbf{c_t} &= \sum_{i=1}^{m} \hat{\alpha}_{ti}\mathbf{h_i} + \sum_{k=1}^{K}\sum_{i=1}^{m/K} \alpha_{ti}^k \mathbf{h_{i+k\frac{m}{K}}}, \end{aligned} \tag{6}$$

where $\mathrm{W_s}$, $\mathrm{W_h}$ are the weight matrices. α_{ti}^k denotes the attention distribution on the k-th part of the problem representations at time step t.

Expression Aggregation Mechanism.

Following (Wu et al., 2020), we use a state aggregation mechanism to compute the expression context state $\mathbf{r_t}$:

$$\mathbf{r_{t+1}} = \sigma(\mathrm{W_r}[\mathbf{r_t} : \mathbf{r_{t,p}} : \mathbf{r_{t,l}} : \mathbf{r_{t,r}}]), \tag{7}$$

σ is a sigmoid function and $\mathrm{W_r}$ is a weight matrix. At time step 1, we use the decoder state s_1 to initialize expression context state r_1. For each node in the currently generated expression tree, $\mathbf{r_{t,p}}$, $\mathbf{r_{t,l}}$ and $\mathbf{r_{t,r}}$ represent the expression context state of the parent node, left child node, and right child node of the current node. If the current node does not have parent or child node at this time step, we pad it with a PAD vector.

Finally, we use a copying mechanism (Gulcehre et al., 2016) so that the model either generate a word from the vocabulary or copy a word from

the input problem X. At time step t, based on the decoder state $\mathbf{s_t}$, the problem context state $\mathbf{c_t}$ and the expression context state $\mathbf{r_t}$, EEH-G2T calculates a copy gate value $g_t \in (0, 1)$ to determine whether the word y_t is generated or copied:

$$\begin{aligned} g_t &= \sigma(\mathrm{W_s}\mathbf{s_t} + \mathrm{W_c}\mathbf{c_t} + \mathrm{W_r}\mathbf{r_t}), \\ \mathbf{P_c}(y_t) &= \sum_{y_t=\mathrm{x_i}} \hat{\alpha}_{ti}, \\ \mathbf{P_g}(y_t) &= \mathrm{softmax}(\mathrm{W_g}[\mathbf{s_t} : \mathbf{c_t} : \mathbf{r_t}]), \\ \mathbf{P}(y_t|y_{<t},\mathrm{X}) &= g_t\mathbf{P_c}(y_t) + (1-g_t)\mathbf{P_g}(y_t). \end{aligned} \tag{8}$$

$\mathrm{W_s}, \mathrm{W_c}, \mathrm{W_r}$ and $\mathrm{W_g}$ are weight matrices. $\hat{\alpha}_{ti}$ is the overall attention vector in the split attention mechanism. The probability distribution $\mathbf{P}(y_t|y_{<t},\mathrm{X})$ of generating y_t is calculated over the copy distribution $\mathbf{P_c}(y_t)$ and generate distribution $\mathbf{P_g}(y_t)$.

2.5 Training

We train the model with the cross-entropy loss, defined as:

$$L = -\sum_{t=1}^{T} \log \mathbf{P}(y_t|y_{<t},\mathrm{X}). \tag{9}$$

During the inference, we use beam search to generate final expression. At time step t, if y_t is an operator, the current node is an internal node, and the model continues to generate its child nodes. If y_t is a number, it represents a leaf node with no child node. Once the children of all the internal nodes have been generated, the generated expression sequence $\mathrm{Y} = \{y_1, y_2, \ldots, y_T\}$ is transformed into an expression tree, and the decoding process is terminated.

3 Experiments

3.1 Datasets

We evaluated our model on two commonly used math word problem datasets, MAWPS (Koncel-Kedziorski et al., 2016) with 2,373 problems and Math23K (Wang et al., 2017) with 23,162 problems. We adopt the data preprocessing provided by Wu et al. (2020). Following previous studies (Xie and Sun, 2019; Li et al., 2020; Wu et al., 2020), we use the same data split for the train/dev/test set.

The Stanford CoreNLP toolkit is used for dependency parsing. Hownet (Dong et al., 2010) and Cilin (Mei, 1985) are used as external knowledge bases. We choose words that appear more than 5 times in the training set or appear as edge labels to build a vocabulary, and replace words that are not in the vocabulary with a UNK token. We use answer accuracy as the evaluation metric.

3.2 Implementation Details

We used Pytorch for our implementation [3]. We used 300-dimensional Glove word embeddings (Pennington et al., 2014). The hidden size is 512. The batch size is 64. The number of heads M in problem-level aggregation is 8. The number K of split attention vectors is 2. We set the learning rate of the Adam optimizer (Kingma and Ba, 2014) to 0.001, and the dropout is 0.5.

During training, it took 120 epochs to train the model. During decoding, we used a beam search with a beam size of 5. We used the same parameter settings for both Math23K and MAWPS datasets. The hyper-parameters are tuned on the valid set.

3.3 Baselines

We compare the performance of our model with the following baselines: **DNS** (Wang et al., 2017) is a seq2seq model that consists of a two-layer GRU encoder and a two-layer LSTM decoder. **Math-EN** (Wang et al., 2018) is a seq2seq model with a bidirectional LSTM encoder and an attention mechanism. **Recu-RNN** (Wang et al., 2019) uses recursive neural networks on the predicted tree structure templates. **Tree-Dec** (Liu et al., 2019) is a seq2tree model with a tree-structured decoder, which generates each node based on its parent and sibling node. **GTS** (Xie and Sun, 2019) is a seq2tree model that generates expression trees in a goal-driven manner. It generates each node based on its parent node and its left sibling subtree embedding. **KA-S2T** (Wu et al., 2020) is a graph-to-tree model with commonsense knowledge from the external knowledge base. It uses a state aggregation mechanism to recursively aggregate neighbors of each node in the expression tree. **Graph2Tree** (Zhang et al., 2020b) is a graph-to-tree model that leverages the nouns nearby the numbers to enrich the quantity representations in the problem.

3.4 Results Analysis

Table 1 summarizes the performance of our EEH-G2T in comparison with other baselines. We

[3] https://pytorch.org/

Models	MAWPS	Math23K
DNS	59.5%	58.1%
Math-EN	69.2%	66.7%
Recu-RNN	66.8%	66.9%
Tree-Dec	-	69.0%
GTS	82.6%	75.6%
KA-S2T	-	76.3%
Graph2Tree	83.7%	77.4%
EEH-G2T	**84.8%**	**78.5%**

Table 1: Answer accuracy of EEH-G2T and other state-of- the-art models on the MAWPS and Math23K datasets.

Models	Math23K
EEH-G2T (full model)	**78.5%**
only sentence-level aggregation	77.4%
only problem-level aggregation	77.8%
remove graph structure	76.5%
remove edge label information	78.1%
remove split attention mechanism	77.7%

Table 2: Ablation analysis of edge-enhanced hierarchical graph encoder and split attention mechanism used in EEH-G2T.

can observe that: 1) Two graph-to-tree model, KA-S2T and Graph2Tree, performed significantly better than the Seq2Tree model GTS, showing that the graph structure in the encoder is effective in enriching the problem representations. 2) Our proposed EEH-G2T outperformed all the other baselines, which proved the effectiveness of using an edge-enhanced hierarchical graph encoder and split attention mechanism.

3.5 Ablation Study

Effect of Hierarchical Graph Encoder.

As shown in Table 2, we estimate the effectiveness of the proposed hierarchical graph encoder. From the results, both sentence-level aggregation and problem-level aggregation improve the performance. Removing the sentence-level aggregation reduces answer accuracy by 1.1%, and removing the problem-level aggregation reduces answer accuracy by 0.7%. When we remove the both aggregation mechanisms and use the initial node representations as the final problem representations, the answer accuracy decreases by 2.0%. We believe that the superior performance of the hierarchical graph encoder is because it captures both the local relations between words within a sentence and the long-range relations between words across sentences.

Models	Math23K
EEH-G2T (full model)	**78.5%**
- self node	64.2%
- neighbor node	77.4%
- dependency	76.9%
- category	77.6%
- same word	76.0%

Table 3: Ablation analysis on reducing the edge categories used in EEH-G2T.

Num	Math23K
K=0	77.7%
K=1	78.1%
K=2	78.5%
K=3	77.5%
K=4	76.2%
K=5	74.8%

Table 4: The performance of EEH-G2T with different number of split vectors on the Math23K valid set.

Effect of Edge Label Information and Split Attention Mechanism.

To prove the effectiveness of edge label information and split attention mechanism in the proposed EEH-G2T, we conduct ablation experiments on the Math23K dataset as shown in Table 2. We observe a slight accuracy drop by 0.4% after removing the edge label information, demonstrating that edge labels provides syntactic and semantic information to enrich the problem representations. Moreover, removing the split attention mechanism leads to a drop by 0.8%, which verifies the effectiveness of using a split attention mechanism.

Effect of Different Edge Categories.

Table 3 shows the performance when removing one edge category at a time. We can see that all the edge categories have positive effects on the model performance. The performance of the model without "self node" edges drops the most, because "self node" allows the model to keep the information of the node itself. Additionally, removing "category" and "neighbor node" edges will slightly reduce model performance. Without "dependency" and "same word" edges, model accuracy will drop to 76.9 % and 76.0%.

Split Number in Split Attention Mechanism.

To explore the impact of the number K of split vectors, we conduct the parameter experiment on

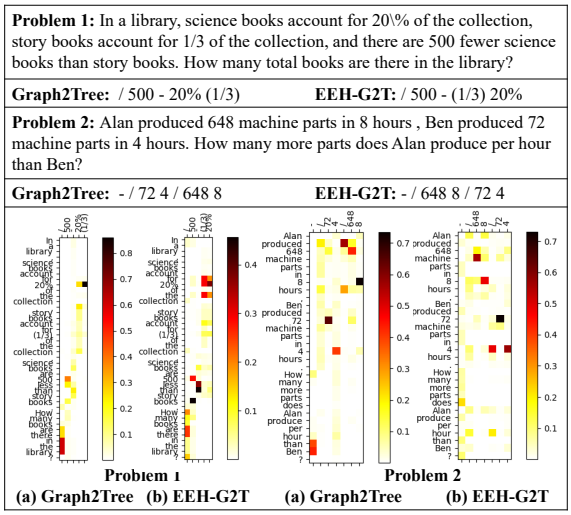


Figure 5: Two examples of generated expressions by Graph2Tree (Zhang et al., 2020b) and EEH-G2T.

the Math23K valid set by varying the split number K from 0 to 5. As shown in Table 4, when the number K increases from 0 to 2, noticeable improvements are remarked on answer accuracy. These result once again confirms the effectiveness of the split attention mechanism because it allows the model to pay attention to different parts of the input problem. The performance starts to drop since K $\geq$ 3. This is probably because more splits means that the problem is split into more parts, so that the model can obtain more information. However, too many splits may break the problem into small fragments, leading to noise. We set the number K of split vectors to 2 in other experiments.

3.6 Case Study

Figure 5 lists two examples generated by Graph2Tree and our EEH-G2T model. In Problem 1, Graph2Tree missed the information that there are fewer science books than story books, and incorrectly generated "- 20% (1/3)". With split attention mechanism, EEH-G2T can better capture this information from the enatire problem. In Problem 2, Graph2Tree incorrectly uses Ben's production speed to subtract Alan's production speed. With hierarchical graph encoder, EEH-G2T can build long-range relations across sentences and therefore generate correct results.

4 Related Work

Math Word Problem Solving: Solving math word problems has long been a very popular task and various methods have been proposed in the past few years (Ling et al., 2017; Wang et al., 2017, 2018). Previous methods usually treated the math word problem as a sequence, and use the same linear encoder to encode math word problems (Liu et al., 2019; Xie and Sun, 2019). Recently, many works that treat math word problems as graphs have shown better performance. Zhang et al. (2020b) connects each number in the problem with nearby nouns to enrich the problem representations. Wu et al. (2020) connects words that belong to the same category in the external knowledge base to capture common sense information. Li et al. (2020) construct an input graph from both the math problem and its corresponding dependency tree to incorporate structural information. However, these methods only capture the local neighbor information of nodes as additional features to enrich the problem representations and ignore the long-range relations across sentences.

In this paper, we propose an edge-enhanced hierarchical graph encoder that captures both the local relations between words within a sentence and the long-range relations between words across sentences. To further guide the decoder to pay attention to different parts of the entire input problem, we propose a split attention mechanism.

Graph Neural Networks: Many works on graph neural networks (GNNs) have been applied to a variety of tasks in recent years, such as node classification (Veličković et al., 2018; Klicpera et al., 2019), relation extraction (Zhang et al., 2018; Sahu et al., 2019), and code summarization (Zügner et al., 2021; Liu et al., 2021). Sahu et al. (2019) proposed a labeled edge graph convolutional neural network model on a document-level graph for inter-sentence relation extraction. (Cui et al., 2020) simultaneously exploits syntactic structure and typed dependency labels to improve neural event detection. Inspired by such works, we also leverage edge label information to enrich the problem representations.

5 Conclusion

In this study, we proposed a novel edge-enhanced hierarchical graph-to-tree model called EEH-G2T for the math word problem solving task. We used an edge-enhanced hierarchical graph encoder that updates the graph nodes in two steps, namely sentence-level aggregation and problem-level aggregation. Additionally, edge label information was incorporated into the model to enrich the

problem representations. We proposed a split attention mechanism to guide the decoder to pay attention to different parts of the entire input problem during generation. Experimental results confirmed that the proposed model, EEH-G2T, outperformed other state-of-the-art models.

Acknowledgments

The authors wish to thank the anonymous reviewers for their helpful comments. This work was partially funded by China National Key R&D Program (No. 2018YFB1005104), National Natural Science Foundation of China (No. 62076069, 61976056), Shanghai Municipal Science and Technology Major Project (No.2021SHZDZX0103).

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*.

Shiyao Cui, Bowen Yu, Tingwen Liu, Zhenyu Zhang, Xuebin Wang, and Jinqiao Shi. 2020. Edge-enhanced graph convolution networks for event detection with syntactic relation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2329–2339, Online. Association for Computational Linguistics.

Zhendong Dong, Qiang Dong, and Changling Hao. 2010. HowNet and its computation of meaning. In *Coling 2010: Demonstrations*, pages 53–56, Beijing, China. Coling 2010 Organizing Committee.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 140–149, Berlin, Germany. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9:1735–80.

Danqing Huang, Jin-Ge Yao, Chin-Yew Lin, Qingyu Zhou, and Jian Yin. 2018. Using intermediate representations to solve math word problems. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 419–428, Melbourne, Australia. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

Johannes Klicpera, Stefan Weiß enberger, and Stephan Günnemann. 2019. Diffusion improves graph learning. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California. Association for Computational Linguistics.

Shucheng Li, Lingfei Wu, Shiwei Feng, Fangli Xu, Fengyuan Xu, and Sheng Zhong. 2020. Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2841–2852, Online. Association for Computational Linguistics.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 158–167, Vancouver, Canada. Association for Computational Linguistics.

Qianying Liu, Wenyv Guan, Sujian Li, and Daisuke Kawahara. 2019. Tree-structured decoding for solving math word problems. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2370–2379, Hong Kong, China. Association for Computational Linguistics.

Shangqing Liu, Yu Chen, Xiaofei Xie, Jing Kai Siow, and Yang Liu. 2021. Retrieval-augmented generation for code summarization via hybrid {gnn}. In *International Conference on Learning Representations*.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Jiaju Mei. 1985. *Tongyi ci cilin*. Shangai cishu chubanshe.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Sunil Kumar Sahu, Fenia Christopoulou, Makoto Miwa, and Sophia Ananiadou. 2019. Inter-sentence relation extraction with document-level graph convolutional neural network. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4309–4316, Florence, Italy. Association for Computational Linguistics.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations*.

Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. 2018. Translating a math word problem to a expression tree. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1064–1069, Brussels, Belgium. Association for Computational Linguistics.

Lei Wang, Dongxiang Zhang, Jipeng Zhang, Xing Xu, Lianli Gao, Bing Tian Dai, and Heng Shen. 2019. Template-based math word problem solvers with recursive neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:7144–7151.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854, Copenhagen, Denmark. Association for Computational Linguistics.

Qinzhuo Wu, Qi Zhang, Jinlan Fu, and Xuanjing Huang. 2020. A knowledge-aware sequence-to-tree network for math word problem solving. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7137–7146, Online. Association for Computational Linguistics.

Zhipeng Xie and Shichao Sun. 2019. A goal-driven tree-structured neural model for math word problems. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 5299–5305. International Joint Conferences on Artificial Intelligence Organization.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

Jipeng Zhang, Roy Ka-Wei Lee, Ee-Peng Lim, Wei Qin, Lei Wang, Jie Shao, and Qianru Sun. 2020a. Teacher-student networks with multiple decoders for solving math word problem. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4011–4017. International Joint Conferences on Artificial Intelligence Organization. Main track.

Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim. 2020b. Graph-to-tree learning for solving math word problems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3928–3937, Online. Association for Computational Linguistics.

Yuhao Zhang, Peng Qi, and Christopher D. Manning. 2018. Graph convolution over pruned dependency trees improves relation extraction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2205–2215, Brussels, Belgium. Association for Computational Linguistics.

Daniel Zügner, Tobias Kirschstein, Michele Catasta, Jure Leskovec, and Stephan Günnemann. 2021. Language-agnostic representation learning of source code from structure and context. In *International Conference on Learning Representations*.