DynGL-SDP: Dynamic Graph Learning for Semantic Dependency Parsing

Bin Li[1,2], **Miao Gao** [1,2], **Yunlong Fan**[1,2],
Yikemaiti Sataer[1,2], **Zhiqiang Gao**[1,2,*], **Yaocheng Gui**[3]
[1]School of Computer Science and Engineering, Southeast University, Nanjing 210096, China
[2]Key Laboratory of Computer Network and Information Integration, Ministry of Education, Nanjing 210096, China
[3]School and Institute of Modern Posts, Nanjing University of Posts and Telecommunications, Nanjing 210003, China
{lib, miaogao, fanyunlong, yikmat, zqgao}@seu.edu.cn
guiyc@njupt.edu.cn

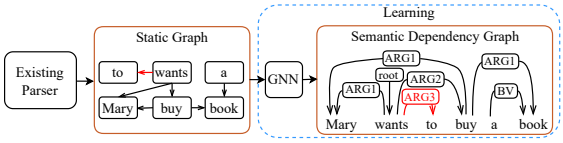
Abstract

A recent success in semantic dependency parsing shows that graph neural networks can make significant accuracy improvements, owing to its powerful ability in learning expressive graph representations. However, this work learns graph representations based on a static graph constructed by an existing parser, suffering from two drawbacks: (1) the static graph might be error-prone (e.g., noisy or incomplete), and (2) graph construction stage and graph representation learning stage are disjoint, the errors introduced in the graph construction stage cannot be corrected and might be accumulated to later stages. To address these two drawbacks, we propose a dynamic graph learning framework and apply it to semantic dependency parsing, for jointly learning graph structure and graph representations. Experimental results show that our parser outperforms the previous parsers on the SemEval-2015 Task 18 dataset in three languages (English, Chinese, and Czech).

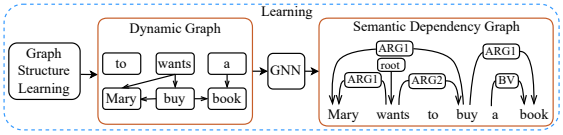
1 Introduction

Semantic dependency parsing (SDP) represents a sentence as a directed acyclic graph (DAG), also called semantic dependency graph (SDG), to capture between-word semantic relationships that are more closely related to the meaning of the sentence. SDP has been widely applied in many downstream tasks of natural language processing (NLP), including sentiment analysis (Lin et al., 2019), abstractive summarization (Jin et al., 2020a), natural language understanding (Wu et al., 2021), etc.

Several semantic dependency parsers are presented in recent years. Their parsing mechanisms are either transition-based or graph-based. A transition-based parser generates a sequence of transition actions to incrementally build an SDG (Wang et al., 2018; Kurita and Søgaard, 2019; Lindemann et al., 2020; Fernández-González and Gómez-Rodríguez, 2020). Transition-based parsers can parse a sentence efficiently. However, they are more prone to suffer from error propagation since the transition prediction stage is sequential and greedy. Graph-based parsers overcome the shortcomings of transition-based parsers. They score each edge (or combination of them) of a possible SDG and globally search for the highest-scoring SDG (Peng et al., 2017; Dozat and Manning, 2018; Wang et al., 2019; Jia et al., 2020; He and Choi, 2020; Wang et al., 2021; Li et al., 2022). A recent success in SDP is the model of Li et al. (2022), which is a graph-based model relying on graph neural networks (GNNs). Their model utilizes an existing parser to construct an initial static graph, and then use GNNs to learn node embeddings based on the static graph to predict dependency relationships between words, as shown in Figure 1(a). This model achieves state-of-the-art performance in three languages (English, Chinese, and Czech), owing to the powerful ability of GNNs in learning expressive graph representations.



(a) Static graph-based disjoint learning for SDP



(b) Dynamic graph-based joint learning for SDP

Figure 1: Static graph-based SDP and dynamic graph-based SDP for example sentence "*Mary wants to buy a book*". The edge and dependency colored red are erroneous.

Despite the promising performance of Li et al. (2022), there are still two drawbacks in their model: (1) the initial graph constructed by the existing

*Corresponding Author

Proceedings of the 29th International Conference on Computational Linguistics, pages 3994–4004
October 12–17, 2022.

parser is static and might be error-prone (e.g., noisy or incomplete), the learned node embeddings based on the static graph may lead to performance degradation to some extent; (2) graph construction stage and graph representation learning stage are disjoint, the errors introduced in the graph construction stage cannot be corrected and might be accumulated to later stages.

To address these two drawbacks mentioned above, we propose a dynamic graph learning framework and apply it to SDP, as shown in Figure 1(b), for jointly learning graph structure and graph representations in this paper. Two GNNs variants, Graph Convolutional Network (GCN) (Kipf and Welling, 2016) and Graph Attention Network (GAT) (Veličković et al., 2018) have been investigated in DynGL-SDP. Experiments are conducted on the SemEval-2015 Task 18 dataset in three languages (English, Chinese, and Czech). Experimental results demonstrate the effectiveness of DynGL-SDP, which outperforms previous studies and achieves a new state-of-the-art performance. In addition, DynGL-SDP shows more advantages with respect to parsing speed.

Contributions The major contributions of this paper are summarized as follows: (1) we propose a dynamic graph learning framework for jointly learning graph structure and graph representations; (2) we apply the framework in SDP to propose a graph-based semantic dependency parser; (3) we conduct sufficient experiments and show that our parser achieves a new state-of-the-art result in three languages. Our code is publicly available at `https://github.com/LiBinNLP/DynGL-SDP`.

2 Related Work

In this section, the studies of SDP and dynamic graph learning will be summarized as follows.

2.1 Semantic Dependency Parsing

Several SDP models are presented in recent years. Their parsing mechanisms are either transition-based or graph-based.

A transition-based parser predicts a sequence of transition actions to incrementally build an SDG. Wang et al. (2018) presented a neural transition-based parser using *Bi-LSTM Subtraction* and *Incremental Tree-LSTM* to represent the key components of the transition system, using a variant of list-based arc-eager transition algorithm for dependency graph parsingto better capture the semantics of segments and internal sub-graph structures. Kurita and Søgaard (2019) presented a transition-based parser, using reinforcement learning algorithm to iteratively apply the syntactic dependency parser to build a DAG structure sequentially. Lindemann et al. (2020) developed a transition-based parser for *Apply-Modify* dependency parsing. They introduced the *stack-pointer* model to predict transitions. Fernández-González and Gómez-Rodríguez (2020) presented a transition-based parser, using *pointer network* to choose a transition among *ROOT*, *Attach-p*, and *Shift*.

Transition-based parsers can parse a sentence efficiently using a linear or quadratic number of transitions. However, they are more prone to suffer from error propagation since the transition prediction stage is sequential and greedy, an erroneous action can affect future predictions.

Graph-based parsers overcome the shortcomings of transition-based parsers, therefore they draw more attention. A graph-based parser scores each edge (or combination of them) of a possible SDG and searches for the highest-scoring SDG. Peng et al. (2017) developed a graph-based parser which explored two multitask learning approaches with a parameterization and factorization that implicitly to model the relationship between multiple formalisms. Dozat and Manning (2018) presented a biaffine attention-based parser, which extended the syntactic parser of Dozat et al. (2017) to train on and generate an SDG. Wang et al. (2019) extended the parser of Dozat and Manning (2018) and added the second-order information for scoring each dependency edge. Either mean-field variational inference or loopy belief propagation is utilized for approximate decoding. Jia et al. (2020) presented a semi-supervised model based on conditional random field autoencoder to learn a dependency graph. He and Choi (2020) significantly improved the performance by introducing contextual string embeddings (called Flair) in the basis of Dozat and Manning (2018). Wang et al. (2021) utilized reinforcement learning algorithm to find better concatenations of embeddings, and then fed it into the parser of Dozat and Manning (2018).

Recently, Li et al. (2022) presented a GNN-based parser. They used an existing parser to construct an initial SDG, and then utilized GNNs to learn node embeddings to predict dependencies,

achieving state-of-the-art performance. However, the initial SDG constructed by the existing parser is static and might be error-prone (e.g., noisy or incomplete), this may leads to error accumulation and performance degradation.

2.2 Dynamic Graph Learning

Graph learning is the process of learning the representations of a graph, GNNs are the most prominent approaches for graph learning. GNNs take in the original feature and adjacency matrix, and output node embeddings as graph representations (Hamilton et al., 2017; Kipf and Welling, 2016; Veličković et al., 2018). Due to the powerful ability in learning graph representations, GNNs have been applied to various downstream tasks, including node prediction (Hamilton et al., 2017), link prediction (Teru et al., 2020), and graph classification (Ying et al., 2018).

Despite GNNs' powerful ability in graph learning, unfortunately, they can only be used when graph-structured data is available. Many NLP tasks may only have sequential data, there is no graph structure available. To address this limitation, several dynamic graph learning frameworks are presented, for jointly learning graph structure and graph representations (Chen et al., 2020; Jin et al., 2020b; Fu and He, 2021).

Inspired by the ideas of these frameworks, we propose a dynamic graph learning framework, and then apply it to SDP, to generate an SDG from word sequence rather than an initial static graph.

3 DynGL-SDP

DynGL-SDP is a graph-based parser using the dynamic graph learning framework. An overview of DynGL-SDP is shown as Figure 2. Given sentence s with n words $[w_1, w_2, \ldots, w_n]$, there are four stages to parse it as an SDG: (1) contextualized representation learning—a bidirectional long short-term memory network (BiLSTM) is used to learn the contextualized representation of each word; (2) graph structure learning—a graph structure learning module is used to learn the adjacency matrix of a potential SDG; (3) graph representation learning—the contextualized representations and the learned adjacency matrix will be fed into the GNNs, to learn expressive node embeddings; (4) dependency relationship learning—the concatenation of node embeddings and contextualized representations are fed into biaffine attention-based parser, to learn dependency relationships between words.

3.1 Contextualized Representation Learning

We concatenate word and feature embeddings, and feed them into a BiLSTM to obtain contextualized representations.

$$x_i = e_i^{(word)} \oplus e_i^{(feat)} \quad (1)$$

$$c_i = BiLSTM(x_i) \quad (2)$$

where x_i is the concatenation ($\oplus$) of the word and feature embeddings of word w_i, c_i is the contextualized representation of w_i.

Word Embedding 100-dimensional word embeddings from GloVe (Pennington et al., 2014) are used for English; 300-dimensional word embeddings from fasttext (Grave et al., 2018) are used for Chinese and Czech.

Feature Embedding Four types of feature embeddings are used, their dimensions (denoted as d) are equal: (1) Part-of-speech (POS) tag: POS tag embedding $E^{(pos)}$ is randomly generated, $E^{(pos)} \in R^{n \times d}$, where n is the number of POS tags; (2) Lemma: lemma embedding $E^{(lemma)}$ is also randomly generated. $E^{(lemma)} \in R^{l \times d}$, where l is the number of lemmas; (3) Character: character embedding is generated using Char-LSTM (Kim et al., 2016) that convolved over three-character embeddings at each time step; (4) BERT: BERT embedding (Kenton and Toutanova, 2019) is extracted from the pretrained BERT$_{\text{base}}$ model.

3.2 Graph Structure Learning

The purpose of the graph structure learning module is to learn the adjacency matrix A of a potential SDG. Two multi-layer perceptrons (MLP) are used to capture head and dependent representations of each word, as Equation 3 and 4:

$$h_i^{(adj-head)} = MLP^{(adj-head)}(c_i) \quad (3)$$

$$h_i^{(adj-dep)} = MLP^{(adj-dep)}(c_i) \quad (4)$$

We can then use biaffine classifier (as Equation 5), to compute the score of a possible edge between w_i and w_j, as Equation 6:

$$Biaff(x_1, x_2) = x_1^T U x_2 + W(x_1 \oplus x_2) + b \quad (5)$$

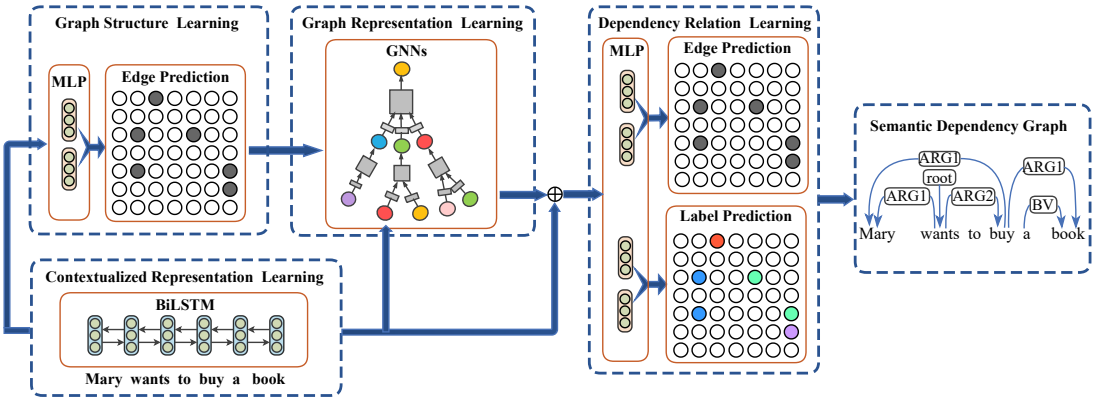


Figure 2: Overall architecture of the proposed DynGL-SDP.

$$s_{i,j}^{(adj)} = Biaff^{(adj)}(h_i^{(adj-dep)}, h_j^{(adj-head)}) \tag{6}$$

A directed edge from w_i to w_j exists when A_{ij} is 1. A_{ij} in A is computed as Equation 7:

$$\hat{A}_{i,j} = \begin{cases} 1, & s_{i,j}^{(adj)} > 0 \\ 0, & s_{i,j}^{(adj)} \leq 0 \end{cases} \tag{7}$$

3.3 Graph Representation Learning

The graph representation learning module utilizes GNNs to learn the each word's representation (i.e. node embedding) that contains graph structure information. K-layer GNNs are employed, which take in the contextualized representations and the learned adjacency matrix A and output the embedding matrix of final layer as node embeddings R. $R^{(k)}$ in k^{th}-layer is computed as Equation 8:

$$R^{(k)} = GNNLayer^{(k-1)}(R^{(k-1)}, \hat{A}) \tag{8}$$

When *GNNLayer* is implemented in *GCN*, the representation of node i in k^{th} layer $r_i^{(k)}$ is computed as Equation 9:

$$r_i^{(k)} = \sigma\left(W \sum_{j \in N(i)} r_j^{(k-1)} + B r_i^{(k-1)}\right) \tag{9}$$

where W and B are parameter matrices; $N(i)$ are neighbors of node i; σ is active function; $r_i^{(0)} = c_i$.

When *GNNLayer* is implemented in *GAT*, $r_i^{(k)}$ is computed as Equation 10:

$$r_i^{(k)} = \sigma\left(W \sum_{j \in N(i)} a_{ij}^{(k-1)} r_j^{(k-1)} + B r_i^{(k-1)}\right) \tag{10}$$

where $a_{ij}^{(k-1)}$ is attention coefficient of node i to its neighbour j at $(k-1)^{th}$ layer.

3.4 Dependency Relationship Learning

The dependency relationship learning module follows the biaffine attention-based parser (Dozat and Manning, 2018), which has two components: edge prediction and label prediction. For each word w_i, the node embedding r_i and the contextualized representation c_i are concatenated to represent it, as shown in Equation 11. For each of the two components, we use MLP to split the final word representation z_i into two parts—a head representation, and a dependent representation, as shown in Equation 12 – 15:

$$z_i = r_i \oplus c_i \tag{11}$$

$$h_i^{(edge-head)} = MLP^{(edge-head)}(z_i) \tag{12}$$

$$h_i^{(label-head)} = MLP^{(label-head)}(z_i) \tag{13}$$

$$h_i^{(edge-dep)} = MLP^{(edge-dep)}(z_i) \tag{14}$$

$$h_i^{(label-dep)} = MLP^{(label-dep)}(z_i) \tag{15}$$

Two biaffine classifiers are used to predict edges and labels, as Equation 16 and 17 :

$$s_{i,j}^{(edge)} = Biaff^{(edge)}(h_i^{(edge-dep)}, h_j^{(edge-head)}) \tag{16}$$

$$s_{i,j}^{(label)} = Biaff^{(label)}(h_i^{(label-dep)}, h_j^{(label-head)}) \quad (17)$$

where $s_{i,j}^{(edge)}$ and $s_{i,j}^{(label)}$ are scores of the edge and label between the word w_i and w_j. U, W, and b are learned parameters.

For edge prediction component, U will be $(d \times 1 \times d)$-dimensional, so that $s_{i,j}^{(edge)}$ will be a scalar. An edge between w_i and w_j exists where $s_{i,j}$ is positive. For label prediction component, U will be $(d \times c \times d)$-dimensional, where c is the number of labels, so that $s_{i,j}^{(label)}$ is a vector that represents the probability distribution of each label. The most probable label will be assigned to the edge between w_i and w_j.

$$\hat{y}_{i,j}^{(edge)} = \{s_{i,j}^{(edge)} > 0\} \quad (18)$$

$$\hat{y}_{i,j}^{(label)} = \arg\max s_{i,j}^{(label)} \quad (19)$$

3.5 Learning

We can train the system by summing the losses from the three modules, back propagating error to the parser. Cross entropy function is used as the loss function, which is computed as Equation 20:

$$CE(p, q) = -\sum_x p(x) \log q(x) \quad (20)$$

We define the loss function of graph structure learning module (as Equation 21), edge prediction module (as Equation 22) and label prediction module (as Equation 23):

$$\mathcal{L}^{(adj)}(\theta_1) = CE(\hat{A}_{i,j}, A_{i,j}) \quad (21)$$

$$\mathcal{L}^{(edge)}(\theta_2) = CE(\hat{y}_{i,j}^{(edge)}, y_{i,j}^{(edge)}) \quad (22)$$

$$\mathcal{L}^{(label)}(\theta_3) = CE(\hat{y}_{i,j}^{(label)}, y_{i,j}^{(label)}) \quad (23)$$

where θ_1, θ_2, and θ_3 are the parameters of three modules.

Then the Adaptive Moment Estimation (Adam) (Kingma and Ba, 2015) method is used to optimize the summed loss function $\mathcal{L}$:

$$\mathcal{L} = \alpha\mathcal{L}^{(adj)} + \beta\mathcal{L}^{(edge)} + (1-\alpha-\beta)\mathcal{L}^{(label)} \quad (24)$$

where α and β are two tunable interpolation constants, where $(\alpha + \beta) \in (0, 1)$.

4 Experiments

4.1 Dataset and Evaluation Metrics

We conduct experiments on the SemEval-2015 Task 18 dataset, which covers three languages (English, Chinese, and Czech) and contains three different formalisms (DELPH-IN MRS (DM) (Flickinger et al., 2012), Predicate-Argument Structure (PAS) (Miyao and Tsujii, 2004), and Prague Semantic Dependencies (PSD) (Hajic et al., 2012)). Three formalisms (DM, PAS, and PSD) are all available for English; only PAS formalism is available for Chinese; only PSD formalism is available for Czech.

Dataset Split The dataset split for three languages is the same as Li et al. (2022), which is shown in Appendix A.1.

Evaluation Metric Labeled F-measure score (LF1) (including ROOT edges) is used as the metric to evaluate our parser's performance on the ID and OOD test sets for each formalism as well as the macro-average over the three of them.

4.2 Hyperparameters

The hyperparameter configuration for our final system is given in Appendix A.2. Following Wang et al. (2019), Adam method is used for optimizing our model, annealing the learning rate by 0.5 for every 10,000 steps, and switched the optimizer to AMSGrad (Reddi et al., 2019) after 5,000 steps without improvement. We train the model for 100,000 iterations with batch sizes of 6,000 tokens and terminated training early after 10,000 iterations with no improvement on the development set.

4.3 Baseline Approaches

We compare DynGL-SDP with previous state-of-the-art approaches. We group them into three groups: transition-based models, graph-based models, and hybrid models.

Transition-based models Turku is from Kanerva et al. (2015). WCGL (Wang et al., 2018) is a neural transition-based model. SemPointer (Fernández-González and Gómez-Rodríguez, 2020) is a transition-based model using Pointer Network. Lindemann et al. (2019) and Lindemann et al. (2020) are compositional semantic parser for SDP and abstract meaning representation.

Models	English							
	DM		PAS		PSD		Avg	
	ID	OOD	ID	OOD	ID	OOD	ID	OOD
Peking (2015)	89.1	81.8	91.3	87.2	75.7	73.3	85.3	80.8
Lisbon (2015)	88.2	81.8	90.9	86.9	76.4	74.8	85.2	81.2
PTS17 (2017): Basic	89.4	84.5	92.2	88.3	77.6	75.3	86.4	82.7
PTS17 (2017): Basic	90.4	85.3	92.7	89.0	78.5	76.4	87.2	83.6
WCGL (2017): Basic	90.3	84.9	91.7	87.6	78.6	75.9	86.9	82.8
D&M (2018): Basic	91.4	86.9	93.9	90.8	79.1	77.5	88.1	85.0
MF (2019): Basic	93.0	88.4	94.3	91.5	80.9	78.9	89.4	86.3
LBP (2019): Basic	92.9	88.4	94.3	91.5	81.0	78.8	89.4	86.2
Lindemann et al. (2019): Basic	91.2	85.7	92.2	88.0	78.9	76.2	87.4	83.3
SemPointer (2020): Basic	92.5	87.7	94.2	91.0	81.0	78.7	89.2	85.8
GNNSDP(GCN) (2022): Basic	93.3	88.0	94.8	91.1	85.6	83.6	91.2	87.6
GNNSDP(GAT) (2022): Basic	93.0	87.9	94.8	91.6	85.4	83.3	91.1	87.6
DynGL-SDP(GCN) Basic	**93.7**	**89.3**	**94.9**	**91.7**	**85.9**	**84.1**	**91.5**	**88.5**
DynGL-SDP(GAT) Basic	**93.8**	**89.2**	**95.1**	**92.0**	**85.9**	**83.8**	**91.6**	**88.3**
D&M (2018): +Char+Lemma	93.7	88.9	93.9	90.6	81.0	79.4	89.5	86.3
MF (2019): +Char+Lemma	94.0	89.7	94.1	91.3	81.4	79.6	89.8	86.9
LBP (2019): +Char+Lemma	93.9	89.5	94.2	91.3	81.4	79.5	89.8	86.8
Jia et al. (2020): +Lemma	93.6	89.1	-	-	-	-	-	-
SemPointer (2020): +Char+Lemma	93.9	89.6	94.2	91.2	81.8	79.8	90.0	86.9
GNNSDP(GCN) (2022): +Char+Lemma	94.2	**90.1**	94.9	91.4	86.4	84.9	91.8	88.8
GNNSDP(GAT) (2022): +Char+Lemma	94.4	89.9	**95.0**	91.8	86.2	84.6	91.9	88.8
DynGL-SDP(GCN) +Char+Lemma	**95.0**	**90.1**	**95.0**	**92.0**	**86.6**	**85.0**	**92.2**	**89.0**
DynGL-SDP(GAT) +Char+Lemma	**94.9**	**90.5**	**95.3**	**92.1**	**86.7**	**85.0**	**92.3**	**89.2**
Lindemann et al. (2019): +BERT$_{large}$	94.1	90.5	94.7	92.8	82.1	81.6	90.3	88.3
Lindemann et al. (2020): +BERT$_{large}$	93.9	90.4	94.7	92.7	81.9	81.6	90.2	88.2
SemPointer (2020): +Char+Lemma+BERT$_{base}$	94.4	91.0	95.1	93.4	82.6	82.0	90.7	88.8
He et al. (2020): +Char+Lemma+BERT$_{base}$+Flair	94.6	90.8	96.1	94.4	86.8	79.5	92.5	88.2
ACE-Fine-tune (2021) +AutoConcat	95.6	92.6	95.8	**94.6**	83.8	83.4	91.7	90.2
GNNSDP(GCN) (2022): +Char+Lemma+BERT$_{base}$	95.1	91.1	95.7	93.2	**87.7**	**87.3**	92.8	90.5
GNNSDP(GAT) (2022): +Char+Lemma+BERT$_{base}$	95.3	91.9	96.0	94.3	87.0	86.7	92.8	91.0
DynGL-SDP(GCN) +Char+Lemma+BERT$_{base}$	**95.8**	**92.7**	**96.2**	94.2	**87.8**	87.0	**93.3**	**91.3**
DynGL-SDP(GAT) +Char+Lemma+BERT$_{base}$	**95.9**	**92.7**	**96.2**	94.3	**87.7**	87.2	**93.3**	**91.4**

Table 1: Comparison of labeled F1 scores achieved by our model and previous parsers on English dataset. Jia et al. (2020) only reports the full-supervised result on DM formalism. The F1 scores of Baseline and our model are averaged over 5 runs. ID denotes the in-domain (Wall Street Journal Corpus) test set and OOD denotes the out-of-domain (Brown Corpus) test set. +Char, +Lemma, +BERT, +Flair, and +AutoConcat mean augmenting the token embeddings with character-level, lemma embeddings, BERT embeddings, Flair embeddings, and automated concatenation of 11 types of pretrained embeddings.

Graph-based models Lisbon is from Almeida and Martins (2015). PTS17 (Peng et al., 2017) is a multitask learning based parser across three formalisms. D&M (Dozat and Manning, 2018) is a biaffine attention-based parser. MF and LBP (Wang et al., 2019) are a second-order model using mean field variational inference or loopy belief propagation. Jia et al. (2020) is a semi-supervised parser, only the full-supervised result on DM formalism is reported in their paper. He and Choi (2020) uses not only BERT but also contextual string embeddings (called Flair). ACE-Fine-tune (Wang et al., 2021) adds automated concatenation of 11 types of pretrained embeddings to the biaffine attention-based parser. GNNSDP (Li et al., 2022) is a GNN-based parser, which is the previous state-of-the-art parser.

Hybrid models Peking is a hybrid model from Du et al. (2015). Riga is from Barzdins et al. (2015).

4.4 Main Results

To perform a fair comparison, we group SDP models in three blocks according to the embeddings provided to the models: (1) just basic pre-trained word embeddings and POS tag embeddings (Basic), (2) character and pre-trained lemma embeddings augmentation (+Char+Lemma) and (3) pretrained BERT embeddings augmentation (+Char+Lemma+BERT).

4.4.1 Results on English

Table 1 shows the comparison of DynGL-SDP and previous studies on the SemEval-2015 Task 18 English dataset. From the result, we have the following observations:

- DynGL-SDP implemented with two GNN variants outperforms all existing parsers on three formalisms of English dataset in three embedding settings.
- Compared to the previous best one (GNNSDP (2022)) in each embedding setting, the best performing DynGL-SDP makes 0.4%, 0.4%, and 0.5% averaged LF1 improvements on in-domain test sets, 0.9%, 0.4%, and 0.4% averaged LF1 improvements on out-of-domain test sets.
- The performances of most parsers have been generally improved when the token embeddings are augmented with more feature embeddings.
- The performances of two DynGL-SDP variants implemented with GCN and GAT are relatively close.
- We note that ACE-Fine-tune (2021) performs better than DynGL-SDP in out-of-domain test set of PAS formalism. A reasonable explanation is that 11 types of pretrained embeddings are used in their model, improving the model's generalization ability.

4.4.2 Results on Chinese and Czech

Table 2 shows the comparison of DynGL-SDP and previous studies on SemEval-2015 Task 18 Chinese and Czech test sets. From the result, we have observed that:

- DynGL-SDP outperforms the previous parsers on Chinese and Czech. The best performing DynGL-SDP makes 0.33% averaged LF1 improvement on Chinese in-domain test set in three embedding settings, 0.28% averaged LF1 improvement on Czech in-domain and out-of-domain test sets in three embedding settings.
- The LF1 scores of two DynGL-SDP variants implemented with GCN and GAT are relatively close on Chinese and Czech.

Models	Chinese PAS ID	Czech PSD ID	Czech PSD OOD
Turku(2015)	79.6	75.3	63.7
Riga(2015)	82.5	75.3	61.3
Peking(2015)	83.4	78.5	64.4
Lisbon(2015)	82.0	79.3	63.5
D&M(2018): Basic	87.4	86.9	77.8
GNNSDP(GCN)(2022): Basic	88.3	88.2	**79.1**
GNNSDP(GAT)(2022): Basic	88.0	87.8	78.9
DynGL-SDP(GCN): Basic	**88.8**	**88.7**	78.9
DynGL-SDP(GAT): Basic	**88.9**	**88.9**	79.0
D&M (2018): +Char+Lemma	87.8	87.6	78.9
GNNSDP(GCN) (2022): +Char+Lemma	**88.5**	88.8	80.2
GNNSDP(GAT) (2022): +Char+Lemma	88.3	88.9	80.2
DynGL-SDP(GCN): +Char+Lemma	**88.5**	**89.7**	**80.3**
DynGL-SDP(GAT): +Char+Lemma	88.3	**90.0**	80.0
GNNSDP(GCN) (2022): +Char+Lemma+BERT	90.1	89.6	**80.7**
GNNSDP(GAT) (2022): +Char+Lemma+BERT	90.4	89.3	80.4
DynGL-SDP(GCN): +Char+Lemma+BERT	**90.8**	**90.1**	80.4
DynGL-SDP(GAT): +Char+Lemma+BERT	**90.8**	**90.1**	80.4

Table 2: Comparison of labeled F1 scores achieved by DynGL-SDP and previous studies on Chinese and Czech datasets. Only the PAS formalism and in-domain (ID) test set are available for Chinese, the PSD formalism, in-domain and out-of-domain (OOD) test sets are available for Czech.

- On the Chinese dataset, the LF1 score of DynGL-SDP has degraded on the contrary when the token embeddings are augmented with character-level and lemma embeddings. The reason is that the character and lemma are not available for Chinese.

In summary, DynGL-SDP outperforms the previous parsers in three languages and three semantic dependency formalisms. Outstanding performances of DynGL-SDP demonstrate that the proposed dynamic graph learning framework is able to learn the expressive graph representations without depending on an initial static graph.

5 Analysis

5.1 Performance on Different Sentence Lengths

Here we want to investigate the performances of DynGL-SDP (ours) and GNNSDP (the previous best one) on different sentence lengths. We split the ID and OOD test sets of DM formalism into 6 and 7 groups (one group with 10 tokens) and evaluate DynGL-SDP and GNNSDP on them. The GNN module in these two parsers is implemented with GAT. The results in different groups are shown in

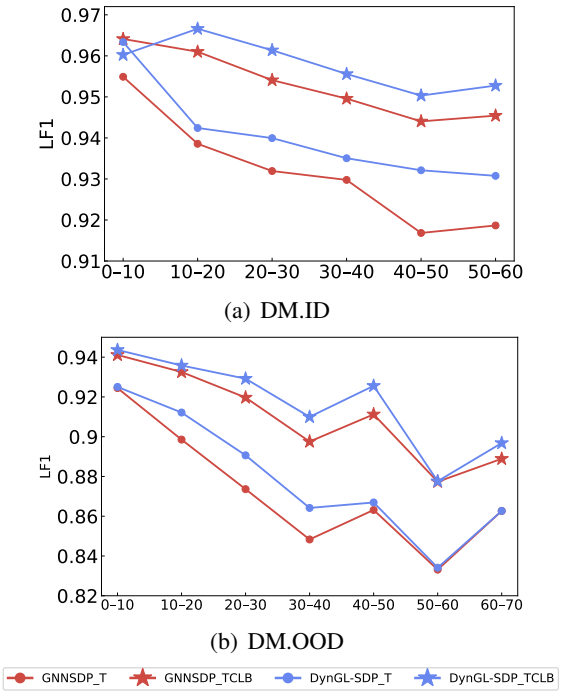


Figure 3: LF1 scores of different sentence lengths in DM formalism on English dataset. *_T represents that only the POS tag embedding is used. *_TCLB represents that the POS tag, character-level, lemma, and BERT embeddings are used.

Figure 3.

From the result, we can see that DynGL-SDP outperforms GNNSDP in different groups and two embedding settings. Furthermore, the performances of both parsers degrade as sentence length gets longer, highly suggesting that parsing longer sentences is still a challenge.

5.2 Parsing Speed

Not only accuracy but also parsing speed determine whether a parser can be applied to downstream tasks. Therefore we compare DynGL-SDP and GNNSDP with respect to parsing speed on an Nvidia GeForce RTX2080Ti server.

To avoid the influence of preprocessing stage, the annotated tokens, POS tags, and lemmas in the dataset are directly used without preprocessing. The result is shown in table 3.

From the result, we can see that:

- DynGL-SDP performs better than GNNSDP with respect to parsing speed in three embedding settings.
- The parsing speed of two parsers slows down when more features are added.

Models	EN DM	CHS PAS	CZ PSD
GNNSDP(GCN):Basic	1153	1032	997
DynGL-SDP(GCN):Basic	**1974**	**1922**	**1828**
GNNSDP(GCN):+Char+Lemma	821	819	775
DynGL-SDP(GCN):+Char+Lemma	**1551**	**1543**	**1411**
GNNSDP(GCN):+Char+Lemma+BERT	346	297	276
DynGL-SDP(GCN):+Char+Lemma+BERT	**677**	**559**	**554**

Table 3: Parsing speed (sentences/second) of DynGL-SDP (ours) and GNNSDP (the previous best one) on English (EN), Chinese (CHS) and Czech (CZ). Parsing speed of each parser is averaged over 5 runs.

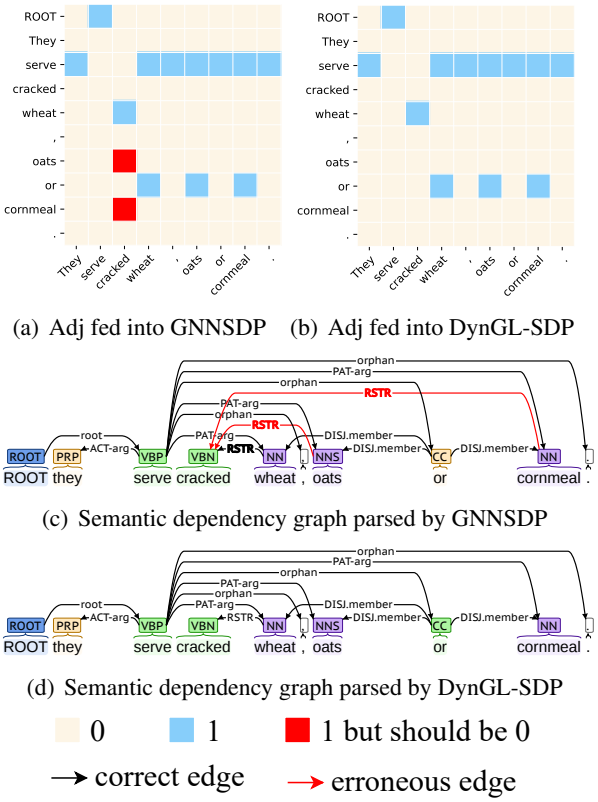


Figure 4: 4(a) and 4(b) are two adjacency matrices (Adj) fed into two parsers. The words in the left are head words, words in the bottom are dependent words. 4(c) and 4(d) are parsing results of two parsers.

5.3 Case Study

We provide a parsing example to show why DynGL-SDP can outperform GNNSDP using dynamic graph learning. Figure 4(a) and Figure 4(b) represent the adjacency matrices fed into GNNSDP and DynGL-SDP. Figure 4(c) and Figure 4(d) are parsing results of GNNSDP and DynGL-SDP for the English sentence "*They serve cracked wheat, oats or cornmeal.*" (sent_id = 40504062, in OOD test set of PSD formalism). The two parsers are implemented with GCN and trained in the basic

embedding setting.

From Figure 4(a), we can see that there are two erroneous values (red square) in this adjacency matrix, indicating that the graph structure input into GNNSDP is noisy. Using learned node embeddings based on the noisy graph leads to two erroneous dependent edges in SDG parsed by GNNSDP (red edge labeled RSTR).

Benefiting from the dynamic graph learning framework, the learned graph structure input into DynGL-SDP is correct. Therefore DynGL-SDP produces a correct SDG.

6 Conclusions

In this paper, we propose a dynamic graph learning framework and apply it in semantic dependency parsing. Experimental results show that our model achieves a new state-of-the-art performance on the SemEval-2015 Task 18 dataset in three languages (English, Chinese, and Czech). The outstanding performance of our model demonstrates that the proposed dynamic graph learning framework is able to learn the expressive graph representations without depending on an initial static graph.

Acknowledgments

This research was sponsored by the Foundation of Jiangsu Provincial Double-Innovation Doctor Program (Grant No. JSSCBS20210507) and NUPTSF (Grant No. NY220176 and NY221106).

References

Mariana SC Almeida and André FT Martins. 2015. Lisbon: Evaluating turbo semantic parser on multiple languages and out-of-domain data. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 970–973.

Guntis Barzdins, Peteris Paikens, and Didzis Gosko. 2015. Riga: from framenet to semantic frames with c6. 0 rules. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 960–964.

Yu Chen, Lingfei Wu, and Mohammed Zaki. 2020. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. *Advances in Neural Information Processing Systems*, 33.

Timothy Dozat and Christopher D Manning. 2018. Simpler but more accurate semantic dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 484–490.

Timothy Dozat, Peng Qi, and Christopher D Manning. 2017. Stanford's graph-based neural dependency parser at the conll 2017 shared task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30.

Yantao Du, Fan Zhang, Xun Zhang, Weiwei Sun, and Xiaojun Wan. 2015. Peking: Building semantic dependency graphs with a hybrid parser. In *Proceedings of the 9th international workshop on semantic evaluation (SemEval 2015)*, pages 927–931.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2020. Transition-based semantic dependency parsing with pointer networks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7035–7046.

Dan Flickinger, Yi Zhang, and Valia Kordoni. 2012. Deepbank. a dynamically annotated treebank of the wall street journal. In *Proceedings of the 11th International Workshop on Treebanks and Linguistic Theories*, pages 85–96.

Dongqi Fu and Jingrui He. 2021. Sdg: A simplified and dynamic graph neural network. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2273–2277.

Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. 2018. Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Jan Hajic, Eva Hajicová, Jarmila Panevová, Petr Sgall, Ondřej Bojar, Silvie Cinková, Eva Fucíková, Marie Mikulová, Petr Pajas, Jan Popelka, et al. 2012. Announcing prague czech-english dependency treebank 2.0. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 3153–3160.

William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035.

Han He and Jinho Choi. 2020. Establishing strong baselines for the new decade: Sequence tagging, syntactic and semantic parsing with bert. In *The Thirty-Third International Flairs Conference*.

Zixia Jia, Youmi Ma, Jiong Cai, and Kewei Tu. 2020. Semi-supervised semantic dependency parsing using crf autoencoders. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6795–6805.

Hanqi Jin, Tianming Wang, and Xiaojun Wan. 2020a. Semsum: Semantic dependency guided neural abstractive summarization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8026–8033.

Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020b. Graph structure learning for robust graph neural networks. In *2020 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Jenna Kanerva, Juhani Luotolahti, and Filip Ginter. 2015. Turku: Semantic dependency parsing as a sequence classification. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 965–969.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of Annual Conference of the North American Chapter of the Association for Computational Linguistics:Human Language Technologies (NAACL-HLT)*, pages 4171–4186.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Thirtieth AAAI conference on artificial intelligence*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*.

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

Shuhei Kurita and Anders Søgaard. 2019. Multi-task semantic dependency parsing with policy gradient for learning easy-first strategies. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2420–2430.

Bin Li, Yunlong Fan, Yikemaiti Sataer, Zhiqiang Gao, and Yaocheng Gui. 2022. Improving semantic dependency parsing with higher-order information encoded by graph neural networks. *Applied Sciences*, 12(8).

Peiqin Lin, Meng Yang, and Jianhuang Lai. 2019. Deep mask memory network with semantic dependency and context moment for aspect level sentiment classification. In *Proceedings of the 2019 International Joint Conferences on Artificial Intelligence*, pages 5088–5094.

Matthias Lindemann, Jonas Groschwitz, and Alexander Koller. 2019. Compositional semantic parsing across graphbanks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4576–4585.

Matthias Lindemann, Jonas Groschwitz, and Alexander Koller. 2020. Fast semantic parsing with well-typedness guarantees. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3929–3951.

Yusuke Miyao and Jun'ichi Tsujii. 2004. Deep linguistic analysis for the accurate identification of predicate-argument relations. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING 2004)*, pages 1392–1398.

Hao Peng, Sam Thomson, and Noah A Smith. 2017. Deep multitask learning for semantic dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2037–2048.

Jeffrey Pennington, Richard Socher, and Christopher D 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.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. 2019. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*.

Komal Teru, Etienne Denis, and Will Hamilton. 2020. Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning*, pages 9448–9457. PMLR.

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

Xinyu Wang, Jingxian Huang, and Kewei Tu. 2019. Second-order semantic dependency parsing with end-to-end neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4609–4618.

Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. 2021. Automated concatenation of embeddings for structured prediction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2643–2660.

Yuxuan Wang, Wanxiang Che, Jiang Guo, and Ting Liu. 2018. A neural transition-based approach for semantic dependency graph parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Zhaofeng Wu, Hao Peng, and Noah A Smith. 2021. Infusing finetuning with semantic dependencies. *Transactions of the Association for Computational Linguistics*, 9:226–242.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31.

A Appendix

A.1 Dataset Split

The dataset split is shown in Table 4.

Language	Train	Dev	Test	
			ID	OOD
English	33,964	1,692	1,410	1,849
Chinese	25,336	3,000	8,976	-
Czech	39,057	3,000	1,670	316

Table 4: The number of sentences contained in each divided dataset of three languages. Only the in-domain test set is available for Chinese; the in-domain and out-of-domain test sets are available for English and Czech.

- For English, 33,964 sentences from Sections 00-19 of the Wall Street Journal corpus as training data, 1,692 sentences from Section 20 as development data, 1,410 sentences from Section 21 as in-domain (ID) test data, and 1,849 sentences sampled from the Brown Corpus as the out-of-domain (OOD) test data.

- For Chinese, the top 3,000 sentences as the development data, the remaining 25,336 sentences as the training data, and 8,976 sentences as ID test data.

- For Czech, the top 3,000 sentences as the development data, the remaining 39,057 sentences as the training data, 1,670 sentences as the ID test data, and 316 sentences as the OOD test data.

A.2 Hyperparameter Values

The hyperparameter configuration for our final system is given in Table 5. 100-dimensional pretrained GloVe embeddings are used for English, in which the token "*unk*" represents the out-of-vocabulary tokens. 300-dimensional pretrained fasttext embeddings are used for Chinese and Czech, in which the token "*UNK*" represents the out-of-vocabulary tokens. Word embeddings of each language will be linearly transformed to be 125-dimensional. Only words or lemmas that occurred 7 times or more will be included in the word and lemma embedding matrix.

Layer	Hyper-parameter	Value
Word Embedding	English	100
	Chinese/Czech	300
Feature Embedding	POS/Lemma	100
	Char/BERT	100
LSTM	layers	3
	hidden size	400
	dropout	0.33
GNN	GCN/GAT layers	3
	GCN/GAT hidden	600
	GCN/GAT dropout	0.33
MLP	adj-head/dep hidden	600
	edge-head/dep hidden	600
	label-head/dep hidden	600
Trainer	optimizer	Adam
	learning rate	$1e^{-2}$
	Adam (β_1, β_2)	0.95
	decay rate	0.75
	decay step length	5000
	Loss(α, β)	0.2, 0.2

Table 5: Final hyperparameter configuration.