Graph-Induced Transformers for Efficient Multi-Hop Question Answering

Giwon Hong[1] **Jeonghwan Kim**[1] **Junmo Kang**[2]* **Sung-Hyon Myaeng**[1]
[1]School of Computing, KAIST
[2]School of Interactive Computing, Georgia Institute of Technology
{giwon.hong, jeonghwankim123, myaeng}@kaist.ac.kr
junmo.kang@gatech.edu

Abstract

A graph is a suitable data structure to represent the structural information of text. Recently, multi-hop question answering (MHQA) tasks, which require inter-paragraph/sentence linkages, have come to exploit such properties of a graph. Previous approaches to MHQA relied on leveraging the graph information along with the pre-trained language model (PLM) encoders. However, this trend exhibits the following drawbacks: (i) sample inefficiency while training in a low-resource setting; (ii) lack of reusability due to changes in the model structure or input. Our work proposes the Graph-Induced Transformer (GIT) that applies graph-derived attention patterns directly into a PLM, without the need to employ external graph modules. GIT can leverage the useful inductive bias of graphs while retaining the unperturbed Transformer structure and parameters. Our experiments on HotpotQA successfully demonstrate both the sample efficient characteristic of GIT and its capacity to replace the graph modules while preserving model performance.

1 Introduction

Graphs are a widely employed data structure from fake news detection in social media (Monti et al., 2019) to molecular graph generation in biomedicine (Jin et al., 2018). More recently, graphs have come to manifest themselves in many subdomains of natural language processing (NLP) for its topological properties that provide useful connectivity inductive bias for model reasoning. Particularly, some models in the multi-hop question answering (MHQA) tasks like HotpotQA (Yang et al., 2018) and MuSiQue (Trivedi et al., 2022) that require textual reasoning through multiple paragraphs (i.e., multi-hop) make use of such graph structures to model the internal structural information within the given document. The external graphs used in these studies have demonstrated the effectiveness of harnessing the information from textual units such as paragraphs and sentences to perform multi-hop reasoning in the QA setting.

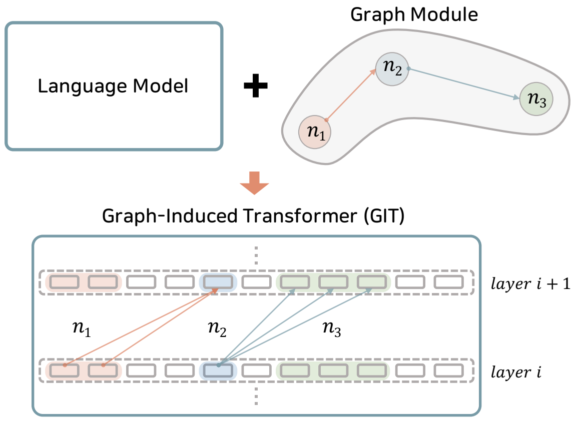

Figure 1: Existing graph-based MHQA model vs. Graph-Induced Transformer (GIT).

The graphs used in MHQA models like HGN (Fang et al., 2020), SAE (Tu et al., 2020) and DFGN (Qiu et al., 2019) are typically placed on top of the pre-trained language models (PLM) like BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019). These graphs use graph neural network's (GNN) message propagation mechanism to update the node and edge parameters, learning to extract useful inductive, structural bias from the graph. These features are then fused with the contextualized representations from the PLM to leverage the information derived from this connectivity (e.g., paragraph-to-sentence) of these graphs. Despite the merits of these graphs, such graph modules require numerous samples (i.e., *sample inefficiency*) to acquire the necessary inductive bias, increasing the overall cost in training. This significantly hinders the model's use of graph connectivity information, especially in a low-resource scenario.

To address the sample inefficient training while preserving the structural information of graphs,

*Work was done while working at KAIST.

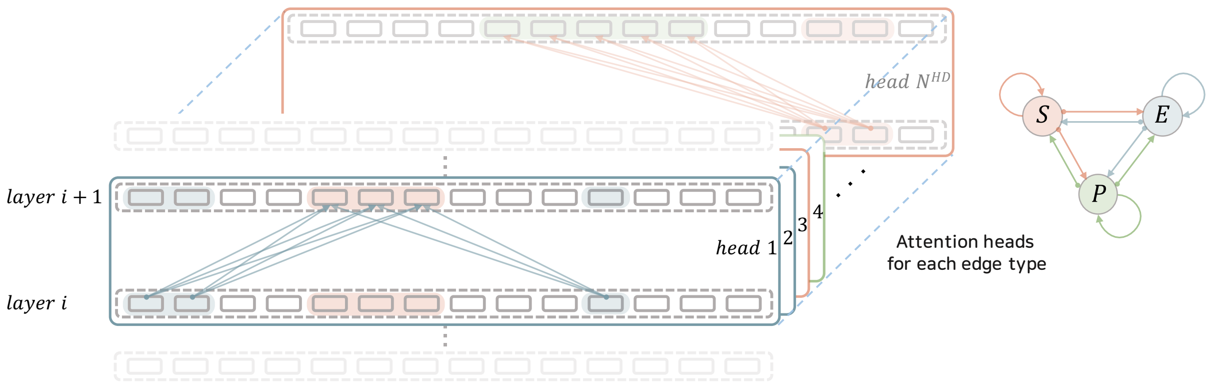

Figure 2: Detailed illustration of Graph-Induced Transformer (GIT). The colors represents different node types, with each token group corresponding to the nodes of the same color. Each edge type is assigned to a separate head.

we propose the Graph-Induced Transformer (GIT). GIT is a simple, intuitive approach to leverage graphs in MHQA tasks without changing the PLM architecture or using additional graph modules. By injecting attention patterns derived from text-based graphs, GIT is able to take advantage of the graphs without the extra modules to fuse graph representations with the text (Figure 1). This enables GIT to: (i) conduct sample-efficient training under a low-resource setting, and (ii) successfully replace the graph module while preserving performance. We conduct experiments to prove the advantages of using GIT instead of separate graphs and provide a range of empirical evidence to show the effectiveness of GIT on the above two aspects.

2 Related Works

MHQA models that leverage the useful connectivity inductive bias of graphs have shown substantial performance gains. With DFGN (Qiu et al., 2019) using entity node graphs, SAE (Tu et al., 2020) using sentence nodes and HGN (Fang et al., 2020) leveraging the hierarchical, paragraph-sentence-entity nodes for its graph, the use of instance-specific graphs has led the development of graph-based MHQA models. Nevertheless, this has led to sample inefficient training that prevents the graph modules from sufficiently acquiring the ability to leverage the interconnected node and edge information, especially in low-resource settings.

Meanwhile, some works have tried to modify the Transformer architecture for graph modeling. Graph-BERT (Zhang et al., 2020) added node-specific positional embeddings and modified the encoder to solve the suspended animation problem of graph neural networks (GNN). Other studies incorporated heterogeneous graphs by converting them into meta-paths (Yun et al., 2019) or using relative temporal positional encodings (Hu et al., 2020) in Transformers. Their modified architectures, however, are specific to graph tasks only, preventing their direct use in the MHQA tasks (and other downstream NLP tasks in general) because they preclude the use of text as input and do not reuse the pre-trained parameters like those of BERT. GIT, in contrast, is designed to handle the MHQA tasks by integrating these graphs into PLMs while preserving the PLM architecture for generalization.

3 Graph-Induced Transformer

The core idea behind GIT is to represent the node connections in the form of masked self-attention layers. Since the attention mask is an inherent part of the Transformer architecture, we can simply alter the attention mask pattern to inject the connective inductive bias of graphs. The major benefit GIT brings with its graph injected as attention masks is the natural and explicit integration of its inductive bias without the need to learn about the graph structure, thereby ensuring a sample efficient utilization of the graph information during training. The graph-derived attention masks regulate the information flow among the intermediate representations, emulating the inter-node connections within a graph (Figure 2).

In GIT, the nodes of the graph are represented as a group (or chunks) of tokens that belong to a homogeneous textual unit; if a group of tokens constitute a sentence, then the group as a whole corresponds to a single sentence node. Edges of the graph are represented as an attention connection between these two groups of tokens. The dynamically extracted attention mask selectively blocks the connections in a self-attention layer, except for

Algorithm 1 Graph-Induced Transformer

```
Input: Context $C$, Attention Mask $M$, Graph $G$ Number of Transformer layers $N^T$, Number of GNN layers $N^G$, Number of Edge types $N^E$
Output: Final Representation $H_{N^T}$
 1: $M^G \leftarrow copy\_and\_zeros(M)$
 2: for each edge e = $(i,j) \in G$ do
 3:     $t \leftarrow edge\_type(e)$        ▷ t ∈ $\{1, ..., N^E\}$
 4:     $M^G[t,i,j] \leftarrow 1$
 5: end for
 6: $M^G \leftarrow concat(M^G[: N^E+1], M[N^E+1 :])$
 7: $H_0 \leftarrow embeddings(C)$
 8: for $n = 1, ..., N^T - 1$ do
 9:     if $n < N^T - N^G$ then
10:         $H_n \leftarrow layer_n(H_{n-1}, M)$
11:     else
12:         $H_n \leftarrow layer_n(H_{n-1}, M^G)$
13:     end if
14: end for
15: $H_{N^T} \leftarrow layer_{N^T}(H_{N-1}, M)$
16: return $H_{N^T}$
```

the connections to the words that belong to the nodes that constitute the edge.

$$\hat{h}_i^{n+1} = O_h^n \overset{N^H}{\underset{k=1}{\Big\|}} (\sum_j \alpha_{ij}^{k,n} V^{k,n} H_i^n)$$

$$\text{where, } \alpha_{ij}^{k,n} = softmax(M^G \odot \frac{Q^{k,n}H_i^n \cdot K^{k,n}H_i^n}{\sqrt{d_k}})$$

In the above equation, $H_i^n \in \mathbb{R}^d$ denotes the hidden state of the i-th token at the n-th layer, where $n < N^T - N^G$ (refer to Algorithm 1 line 9), Q, K and $V \in \mathbb{R}^{d_k \times d}$ each denote query, key and value projection matrix, respectively. $O_h^n \in \mathbb{R}^{d \times d}$ is the projection matrix for the intermediate embeddings concatenated over N^{HD} heads, where the k-th head is assigned an attention mask M^G which enforces the connection between one textual unit (e.g., paragraph) to another (e.g., sentence) by zeroing out the other attentions and maintaining the remaining attention coefficients for the connection.

Since there are multiple edge types that connect one node type to another, we define an edge type as a pair of head and tail node types (e.g., paragraph-to-sentence), in addition to the edge types of the original graph. The different edge types are then applied to the Transformer encoder as a set of attention masks by assigning a head per edge type; this is based on the previous studies (Rogers et al., 2020; Jo and Myaeng, 2020) that each head in a Transformer learns different linguistic concept. Furthermore, considering that nodes in the graph are updated through l different steps given l GNN layers, we allocate l layers inside the PLM to emulate the message propagation in GNNs. To promptly adjust to different end tasks, we keep the last layer as an original fully connected self-attention layer.

4 Experiments

4.1 Settings

Dataset We use HotpotQA[1] (Yang et al., 2018), an English multi-hop question answering (MHQA) dataset for experiments. HotpotQA contains 90,564 training instances and 7,405 for development and test. All the evaluations are conducted using the development set (Distractor setting).

Models We use the following graph-based MHQA models as our baselines for which the codes are publicly available. For simplification, we denote question (Q), paragraph (P), sentence (S), and entity (E) accordingly (e.g., the edge connecting a paragraph to sentence is a P2S type).

DFGN (Qiu et al., 2019) uses an entity graph that connects an entity to another in the text (E2E).

SAE (Tu et al., 2020) uses a sentence graph that connects a sentence node with other sentence nodes (S2S) with 3 different edge types: (i) within document, (ii) across documents and (iii) when both sentences share an entity with the question.

HGN (Fang et al., 2020) uses a hierarchical graph with question, paragraph, sentence and entity nodes, with 8 different edge types: 1) Q2P, 2) Q2E, 3) P2P, 4) P2S, 5) S2P (hyperlink), 6) S2S, 7) S2E, and 8) a self-loop

Implementation We have retrained the models in the original papers in our environment for a fair comparison against GIT. For the details on implementation refer to Appendix A.

4.2 Graph Replacement with GIT

In this section, we first analyze whether GIT can be applied to different baselines and replace their graphs. Table 1 shows the result of comparing baselines under two different settings: "Graph" refers to using the original graph-based model and "GIT" refers to removing the graph and applying GIT.

[1]https://hotpotqa.github.io/

		Ans EM	Ans F1	Sup EM	Sup F1	Joint EM	Joint F1
DFGN	Graph	55.40	69.13	49.26	80.25	31.56	58.27
	GIT	**55.91**	**70.01**	49.17	**80.77**	31.75	**59.17**
SAE	Graph	66.68	80.06	60.35	86.50	43.55	71.45
	GIT	66.32	79.92	**60.93**	86.49	43.59	71.31
HGN	Graph	68.08	81.63	**63.34**	88.45	**46.19**	73.70
	GIT	67.91	81.61	62.11	88.33	45.31	73.61

Table 1: Comparison of applying graph and GIT on various MHQA graph models on HotpotQA.

SAE	Joint EM	Joint F1
Graph	43.55	**71.45**
w/o Graph	42.39	70.66
GIT	**43.59**	71.31
Graph + GIT	43.38	71.12

Table 2: Ablation study about graph and GIT for SAE on HotpotQA. "w/o Graph" is removing graph propagation among nodes and "Graph + GIT" is enabling both the graph propagation and GIT attention masking.

Data Portion	SAE	GIT
1%	9.57	**15.68** (↑ 63.8%)
2%	17.79	**24.88** (↑ 39.8%)
5%	28.05	**29.68** (↑ 5.8%)
10%	31.41	**33.02** (↑ 5.1%)
50%	40.61	**41.67** (↑ 2.6%)

Table 3: Joint EM of SAE and its GIT counterpart in low-resource scenario on HotpotQA.

The results in Table 1 show that GIT achieves performance that is on par with those of the baselines. This indicates that GIT is generalizable to various graphs with different node and edge types, and effectively replaces the external graph modules.

To evaluate whether the proposed GIT actually emulates and compensates for the role of graphs, we conduct a more thorough study as in Table 2. Here, we seek to address the equivalence of GIT to graphs from the information retention perspective. For "w/o Graph," we ascertain that removing the graph decreases model performance as the authors of SAE claim. By applying GIT, this loss of performance is successfully restored, proving the equivalence of GIT and graphs in retaining information. In addition, "Graph + GIT" does not outperform the original setting, which implies that GIT and graphs introduce features, while informative, of the same nature to the model.

4.3 Low-resource Scenario

To test the sample efficiency of GIT in training, we designed a low-resource scenario that uses only 1%, 2%, 5%, 10% and 50% of the HotpotQA training dataset. Table 3 compares the results of SAE, a MHQA graph model and its GIT counterpart; the percentage of improvement is denoted in red.

The results in Table 3 demonstrate significant performance gains by replacing the graphs with GIT layers. The main takeaway here is that, unlike the previous graph-based models, the GIT takes the GNN module out of the picture, effectively eliminating the need to train its additional parameters while taking the full advantages of already pre-trained Transformer. We can clearly see that GIT outperforms the graph-based model when the data is scarce (i.e. low-resource), especially in the 1% ($\sim$ 900 instances) case GIT improves the model performance by 63.8% compared to its original setting. Similar improvements are evidenced in 2%, 5%, 10%, and 50% cases. This result provides a conclusive evidence that GIT is very effective when dataset is scarce. On top of the GIT's robustness in data scarcity, it is apparent that even in the 50% setting where the number of instances exceeds 45K, GIT outperforms the graph-based MHQA model by a substantial margin.

5 Conclusions

This work presented GIT, the graph-induced Transformer that drastically improved MHQA models' sample efficiency and replaces graphs in the models while retaining their performance. Our empirical evidences demonstrated that models can enjoy the benefits of connective inductive bias of graphs without additional graph modules in place. The design of GIT also allowed us to reuse the parameters of PLM while incorporating the graph information. Future directions of our work may include using GIT in downstream NLP applications where the graph inductive bias is necessary and dataset is scarce.

Limitations

Our proposed method, GIT, allocates each edge type of the graph to the attention head of the Transformer. This means that if the number of edge types exceeds that of attention heads, it is difficult to apply the GIT scheme. This may be problematic in a situation 1) where the number of attention heads of the Transformer is very small, for example, in the tiny/mini/small setting suggested by Turc et al. (2019) or 2) when we have a complex graph which consists of numerous types of nodes and edges.

Also, we tested GIT only with MHQA, a QA task where graphs are proven to be useful. In practice, however, it should be generally applicable when text and its connectivity are important.

Acknowledgements

This work was supported by Institute for Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. 2013-2-00131, Development of Knowledge Evolutionary WiseQA Platform Technology for Human Knowledge Augmented Services) and (No. 2022-0-00369, (Part 4) Development of AI Technology to support Expert Decision-making that can Explain the Reasons/Grounds for Judgment Results based on Expert Knowledge)

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Yuwei Fang, Siqi Sun, Zhe Gan, Rohit Pillai, Shuohang Wang, and Jingjing Liu. 2020. Hierarchical graph network for multi-hop question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8823–8838, Online. Association for Computational Linguistics.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pages 2704–2710.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. 2018. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*, pages 2323–2332. PMLR.

Jae-young Jo and Sung-Hyon Myaeng. 2020. Roles and utilization of attention heads in transformer-based neural language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3404–3417, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M Bronstein. 2019. Fake news detection on social media using geometric deep learning. *arXiv preprint arXiv:1902.06673*.

Lin Qiu, Yunxuan Xiao, Yanru Qu, Hao Zhou, Lei Li, Weinan Zhang, and Yong Yu. 2019. Dynamically fused graph network for multi-hop reasoning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6140–6150, Florence, Italy. Association for Computational Linguistics.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in bertology: What we know about how bert works. *Transactions of the Association for Computational Linguistics*, 8:842–866.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Ming Tu, Kevin Huang, Guangtao Wang, Jing Huang, Xiaodong He, and Bowen Zhou. 2020. Select, answer and explain: Interpretable multi-hop reading comprehension over multiple documents. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 9073–9080.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. In *Advances in Neural Information Processing Systems*, pages 11960–11970.

Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. 2020. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*.

A Experiment Setting Details

We abide by the official HotpotQA evaluation metrics and the script provided[2].

Unless otherwise specified, we have used the original hyperparameters of those three models.

DFGN (Qiu et al., 2019) We have changed batch size (32 -> 8) and gradient accumulation step (1->4). For the encoder, BERT (Devlin et al., 2019) pre-trained model (*bert-base-uncased*) which contains 110M parameters, same as the original code. Also, since there is no explicit query node in DFGN, we generated connections between the query tokens and the other nodes in the GIT layers.

SAE (Tu et al., 2020) As the published SAE code is incomplete on the training side, we had to implement some parts of the code on our own. We have changed batch size (2->4) and gradient accumulation step (1->8). Also we used Roberta (Liu et al., 2019) model which contains 355M parameters (*RoBERTa-large*) pre-trained on the SQuAD dataset (Rajpurkar et al., 2018)[3], similar to the original code (As we was not able to get the pre-trained model that the authors used). Also, since there is no explicit query node in SAE, we generated connections between the query tokens and the other nodes in the GIT layers.

HGN (Fang et al., 2020) We have changed batch size (8->4), gradient accumulation step (1->2), and "fp16" option (True -> False). Also, we used RoBERTa pre-trained model (*RoBERTa-large*) which contains 355M parameters, same as the original code.

B Computing Infrastructure

Our workstation contains two NVIDIA GeForce RTX 3090s, AMD Ryzen Threadripper 3960X 24-Core Processor, and 128GB RAM. Each of the baseline used in this work was trained using a single GeForce RTX 3090, except for DFGN which requires two GPUs.

C Layer-wise Probing for GIT

To probe the layer-wise applicability of the attention masks constructed with our GIT scheme, we probe chunks of Transformer layers within the PLM (RoBERTa-large) and evaluate their performance on HotpotQA development set in the distractor setting. Our results suggest that the preservation of lower and intermediate layers is important in building a set of contextualized representations that are rich in semantic and syntactic information, while applying the attention masks in the uppermost layers prove to be most effective. This implies that the last layers can be harmlessly altered to embrace the purposefully injected inductive bias.

To determine whether additional layers that incorporate our GIT masks positively affect the PLM when scaling their parameters, we also experiment with the additional layers setting. By comparing the additional layers with and without GIT masks, we observe that the PLM with extra layers that include the GIT masks outperform the vanilla extra layers. This hints at the feasibility of scaling up the PLMs with positive graph inductive bias.

[2]https://github.com/hotpotqa/hotpot/blob/master/hotpot_evaluate_v1.py

[3]https://huggingface.co/phiyodr/roberta-large-finetuned-squad2

GIT	Joint EM	Joint F1
Layer 1-24	42.75	70.70
Layer 1-3	42.47	70.72
Layer 8-10	43.07	71.21
Layer 15-17	43.04	70.76
Layer 22-24	43.36	71.01
Layer 21-23	**43.59**	**71.31**
GIT Additional Layers		
Layer 25-27 (No GIT)	43.13	71.07
Layer 25-27	**44.09**	**71.64**

Table 4: Layer-wise application of the attention masks constructed with GIT scheme. The PLM used in this table is RoBERTa-large (24 layers). The results for the additional layers case are provided in the last two rows.