Research Group Social Computing
Department of Informatics
Technical University of Munich

GraphWalker

A Novel Approach For Knowledge Graph-based Question Answering

Baris Sen ✉, **Can Arisan** ✉, and **Demir Senturk** ✉

Department of Informatics, Technical University of Munich

✉ baris.sen@tum.de
✉ can.arisan@tum.de
✉ demir.senturk@tum.de

August 10, 2021

Abstract — The task of question answering (QA) aims to find the answer to a question formulated in natural language. To fulfill this task, a knowledge base is required, where the answer can be queried. Knowledge graphs (KG) are one such form of knowledge base, structured as a relational graph, with nodes representing entities and edges representing relations between them. We will be focusing on the task of knowledge graph-based question answering (KGQA), where the goal is to find the node in the graph that represents the answer entity, starting from the node that represents the source entity in the question text. A problem that occurs during the KGQA task is the case of a large number of hops between the source and target entities in the graph. In such cases, the amount of nodes that need to be considered increases significantly. In our paper, we introduce a new approach and an implementation, **Graphwalker**, which abstracts the task from the number of hops between the source and target.

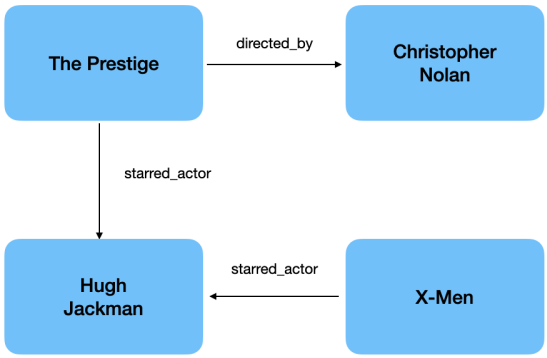


Figure 1 A subgraph from the dataset MetaQA. As an example, the question "*Who is the director of a movie, who stars an actor from the movie X-Men?*", has the source entity "*X-Men*" and a possible target entity "*Christopher Nolan*". The path between the source and target corresponds to a 3-hop traversal, and could be seen as how the KG helps with the task of QA.

1 Problem Definition

A KG is a multi-relational graph composed of entities represented as nodes and relations represented as different types of edges[1]. Each edge is represented as a triple of the form `(head entity, relation, tail entity)`, indicating that the head and tail entities are connected by the relation. As an example for such a triple, (*The Imitation Game, directed_by, Morten Tyldum*) can be given. The task of KGQA uses these triples to navigate through the graph and find the target entity representing the answer to the natural language question, which also contains the source entity in itself. Figure 1 has an example subgraph from the dataset MetaQA[2], which we also selected to use in our research.

One of the main problems that arise during KGQA is the case of multi-hop traversal for a large number of hops. At each node, the dependencies to every neighbor of the current node should be considered for inference with a KG. As the number of hops increase, the number of nodes that should be considered grows exponentially. This makes considering these nodes during training and inference time computationally infeasible. To tackle this issue, a basic solution could be to bound the possible number of hops to a maximum value, however, this would decrease the accuracy for dependencies between nodes that are further away from each other than this maximum value.

2 Related Work

There are several existing approaches that focus on KGQA. Factoid-based QA using embeddings [3], [4], sub-graph extraction for the specific question [5], [6], using a similarity function between the question and the triple [7], mapping the question text to logical forms [8, 9] are some methods that have been implemented before. Furthermore, neural network-based

methods to learn the scoring function [10, 11, 12, 13] have also been used. The methods above are, however, are not suited to be used for multi-hop KGQA directly.

In our paper, we particularly focus on KGQA for questions that require multi-hop traversals on the graph to reach the answer. In their research, Sun et al.[14] provide a method and model, PullNet, which uses an iterative process to construct a question-specific subgraph with only the information relevant to the particular question. Also, a text corpus is incorporated as an additional knowledge source, which is particularly helpful for multi-hop reasoning on sparse KGs.

However, relevant text corpora are not always available and are hard to identify. This limits the extensibility and makes the application of this method to a broader domain difficult. Also, the neighborhood size has to be limited to prevent the exponentially growing number of entities for multiple hops. Another method that addresses these issues is proposed by Saxena et al.[15]. They introduce the notion of KG embeddings to perform missing link prediction on sparse graphs and reach state-of-the-art performance for the domain they use.

3 Methodology

3.1 Multi-hop Walk on KG

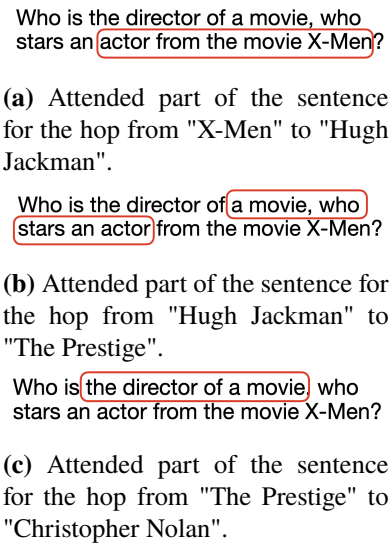
In our research, we focus on multi-hop reasoning for the KGQA task. As we discussed before, as the path between source and target entities gets longer, the number of entities that need to be considered also increases, but at an exponential rate. This is a result of the number of neighbors that need to be considered at each individual hop getting multiplied for every additional step and causes the computation of a direct traversal for an arbitrary hop count infeasible at execution time. As we do not want to lose accuracy by limiting the hop count to a maximum value, we propose another approach that would allow a traversal on the graph for an arbitrary hop count.

The main idea of our method is to abstract selecting a neighbor on an individual hop from the whole traversal from the source to the target entity. Our model does not aim to reach the target entity from the beginning and search for it on the graph indefinitely by expanding nodes but aims to make the best individual decision at each hop to navigate towards the goal. To enable the model to achieve this, we utilized two key concepts: having a stop signal to terminate the search instead of making a new hop and using attention to improve decision making for each hop.

3.1.1 Stop Signal

In the basic case of searching for the target entity starting from the source entity, we do not terminate the search until the target is reached or there is no node to expand to, which would mean that the whole graph may be traversed at worst case. As we abstract making an individual hop from the whole traversal from source to target, another issue would arise, which is when to terminate the search. In our approach, we allow the model to stop traversing the graph at any time. At any step, instead of selecting a neighbor of a node to hop to, the model can decide that the search is finished and send a stop signal, which will terminate the traversal. The node which sent the stop signal would be the prediction of the model as the answer to the question. In this way, the model can learn when to stop on itself and does not require knowledge of an hop count.

3.1.2 Attention



(a) Attended part of the sentence for the hop from "X-Men" to "Hugh Jackman".

(b) Attended part of the sentence for the hop from "Hugh Jackman" to "The Prestige".

(c) Attended part of the sentence for the hop from "The Prestige" to "Christopher Nolan".

Figure 2 Attention mechanism visually depicted on the example from Figure 1. The subfigures (a)-(c) correspond to the hops on the subgraph given.

Another key concept for our idea was attention. As we train our model to make individual decisions at each hop, it needs to focus on different parts of the question. This lies in parallel to the notion of using attention. Attention mechanism was first introduced by Bahdanau et al.[16] and had significant applications in natural language processing such as the work of Vaswani et al.[17]. It reflects the importance of

an annotation in the input with respect to the previous state, in deciding the next state, and serves generally as an alignment between the source sentence and the corresponding target sequence. In our case, the question words would be aligned, such that specific parts of the sentence would correspond to edges on the graph, therefore would be more important in making particular decisions for hops. A visual example of how the attention mechanism relates to our case is given in Figure 2.

3.2 Shortest Path As Ground Truth

To train our model to make correct decisions along the path from the source to the target entity, we need a ground truth to compare the prediction with. We assume that the most optimal path from the source entity to the target entity would be the shortest path between them on the graph. We further assume that any of the shortest paths would be a valid one, in case of the existence of multiple ones, to be used during training. In the dataset that we used, we trained our model on the 3-hop vanilla data. For this selection, all the shortest paths from source to target actually correspond to the 3-hop paths, which indicates that our assumptions are suited to our training setup.

To compute the shortest paths, we perform breadth first search (BFS)[18] for each question-answer pair, starting from the source entity and terminating the search as the target is reached. For questions with multiple valid answers, we choose one to be used in training and compute the shortest path, but during validation, we accept any of these answers as a correct prediction.

Similar to arguments presented for the issue of exponentially increasing computational effort for increasing hop count, a BFS performed during execution time also has a high computational effort. However, we perform it once for every data sample before the training as a pre-processing step to create these ground truth labels as an additional input. Therefore, it does not affect the runtime complexity of the training.

3.3 Teacher Forcing

While the model makes predictions for each individual hop during training, any wrong decision might cause the model to deviate from the path leading to the target entity. This error for one individual hop would accumulate for the following hops as well, and the training process would be affected negatively. To prevent this, we use the notion of teacher forcing. Teacher forcing in general can be explained as correcting the error of the model at each time step after calculating the loss of its prediction[19].

In our case, this is implemented as the following: we first let the model make a prediction for a hop and compare it with the ground truth that was computed during the BFS to calculate the loss. Then, even if the prediction deviated from the ground truth, we choose the correct ground truth node as the next step on the traversal, thus guiding the model towards the goal by correcting its deviation. This method makes the model converge faster and increases stability. We apply this during training only and make no corrections during validation.

4 Implementation

4.1 Dataset

We used the dataset MetaQA[2], which consists of more than 400k questions with a domain of movies. This dataset was created using another knowledge base, WikiMovies[20], and has questions from 1-hop to 3-hop. For our implementation, we used the 3-hop questions of the "vanilla" version of the dataset. It consists of 43233 entities and 142744 question-answer pairs with the split provided in Table 1.

train	dev	test
114,196	14,274	14,274

Table 1 The train/dev/test split of the question-answer pairs of the MetaQA dataset

4.2 Graph Creation

To initialize the graph that we used to perform the BFS and the traversal of our model, we used the python framework NetworkX[21]. We use the triples of the form $(entity_1, relation, entity_2)$ we get from our knowledge base and add the entities to the graph if they were not already added, and the relation between them as a labeled directed edge. The resulting graph is a multi-edged directed graph (MultiDiGraph in NetworkX), as two entities might have multiple relations between them. For example, the director of a movie can also star as an actor in it. NetworkX allows us to access the neighbors of nodes and the corresponding relations efficiently, which is important for the performance of the pre-processing step.

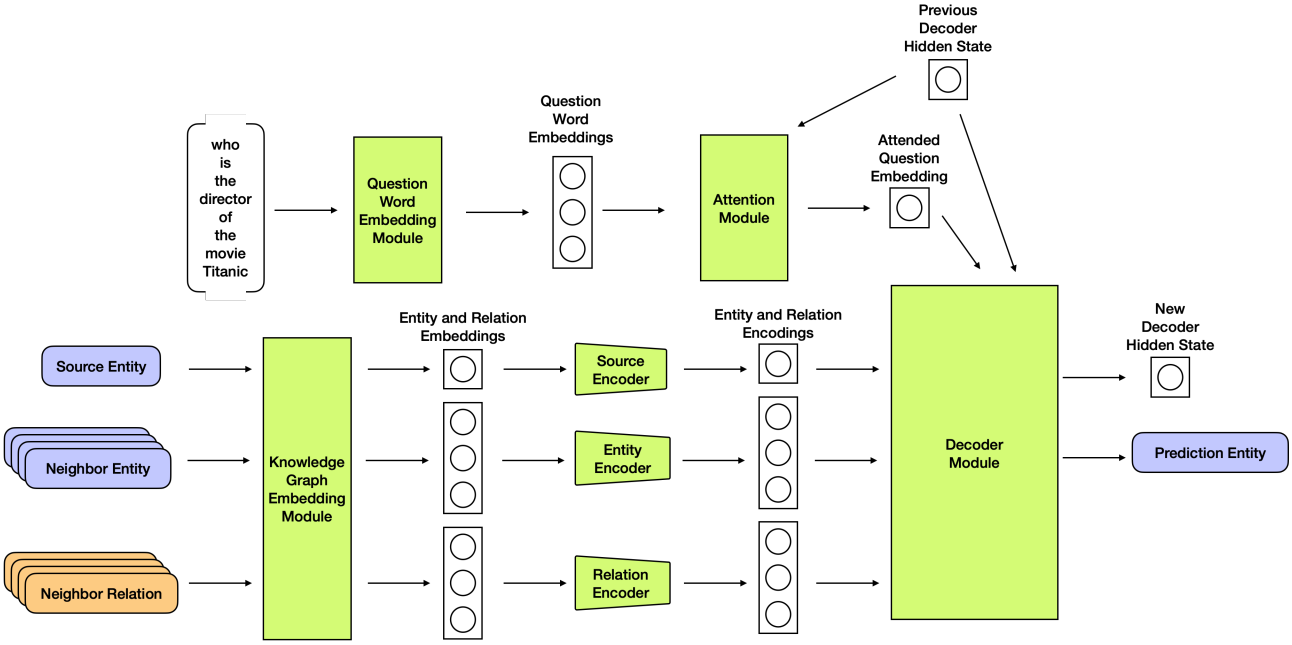


Figure 3 Architecture of GraphWalker.

4.3 Architecture

The architecture is composed of two main parts: Encoder and Decoder modules. The Encoder module consists of the Knowledge Graph Embedding module, the Question Word Embedding module, Source Encoder, Entity Encoder, and Relation Encoder modules.

First, our Knowledge Graph Embedding module takes as input the source entity, neighbor entity, and neighbor relation and gives the corresponding entity and relation embeddings as its output. The entity and relation embeddings are then fed into the source, entity, and relation encoder modules. As output, we get the entity and relation encodings. On the other hand, the question, our other initial input, is the input of the Question Word Embedding module. It outputs the corresponding question word embeddings. These mentioned parts of the architecture constitute the Encoder model of our architecture.

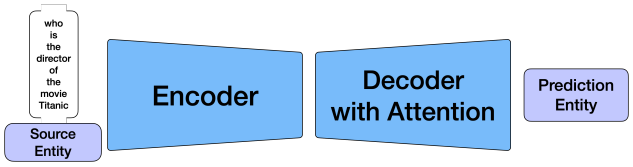


Figure 4 Overview of one unit in RNN.

Furthermore, the Encoder modules are followed by the Decoder model. The Decoder model is enhanced with the Attention module, which uses Luong attention[22]. The Attention module takes as input the question word embeddings and produces the attended question embeddings. The Decoder module takes then the entity and relation encodings, the attended question embedding, and gives the new decoder hidden state and a prediction entity as its output. The previous decoder hidden state is then fed into the attention module.

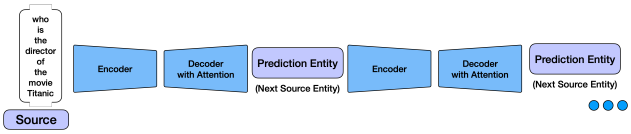


Figure 5 Unfolded RNN Architecture.

Essentially, the architecture can be seen as an encoder-decoder architecture enhanced with attention, which takes the question and source entity as input and outputs the prediction entity. As the model consists of an RNN architecture, the recurrent application of this encoder-decoder model is used, where the prediction entity of the previous decoder is the next input source entity.

5 Experiments

In this section, we are interested in investigating the model's performance in terms of accuracy from different aspects. We focus on the experiments we conducted, where our aim is to tune the hyperparameters and find the best model. First, we provide experiments

with and without regularization and using different batch sizes. Furthermore, we add more linear layers to the model and compare the accuracy. Besides the changes in the model, we trained our model with different max neighbor counts. Lastly, we delineate the experimental results and compare the model's performance with different parameters.

5.1 Regularization

This experiment focuses on the influence of regularization on the model's performance. Thus, we use our model with and without L2 Regularization and compare the experimental results.

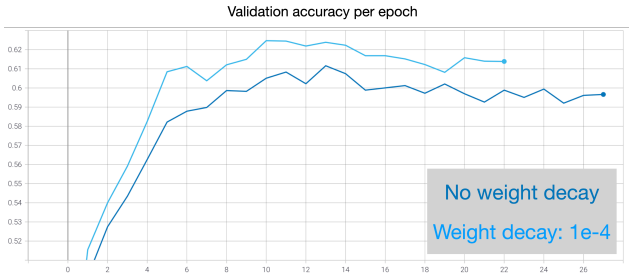


Figure 6 Accuracy results with and without L2 Regularization.

We achieved higher accuracy by using a weight decay of 1e-4. The underlying reason for this improvement can be the reduced generalization error, as the regularization improves generalization and reduces overfitting[23].

5.2 Batch Size

In this experiment, we use two different settings with a batch size of 128 and 512. In Figure 7, it can be observed that a higher batch size does not help significantly. Thus, we decide to proceed with experimenting with a batch size of 128.

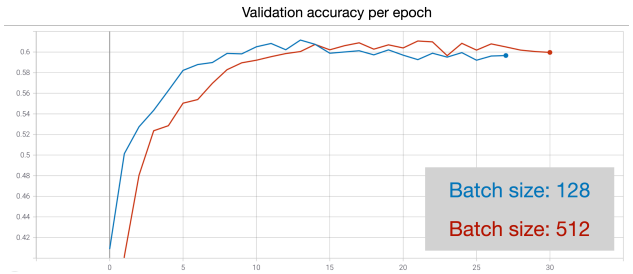


Figure 7 Validation accuracy results with a batch size of 128 and 512.

5.3 Network Size

In this subsection, we examine the network size and change the number of linear layers of our model. The results have shown that the accuracy results are higher with more linear layers compared to the model with less linear layers. Consequently, we choose the model with more linear layers.

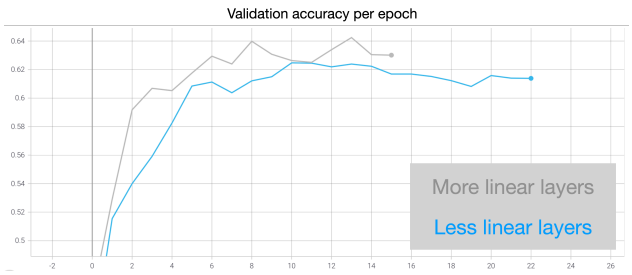


Figure 8 Comparison of results for more and less number of layers in terms of validation accuracy.

5.4 Max Neighbor Count

In addition to the previous experiments, we are interested in the model's behavior with different max neighbor counts (MNC). MNC indicates the maximum number of considered neighbors of a particular node. In this experiment, we compare the accuracy results for varying max neighbor counts. The maximum neighbor count is set to 150, 200, and 600 before each run. Our model achieved better results for lower MNC. As the MNC increases, it could be harder for the model to learn. Furthermore, the KG has a very high concentration of nodes with less than 200 neighbors. Thus, the model can learn with a lower MNC better and achieve higher accuracy results with a lower MNC.

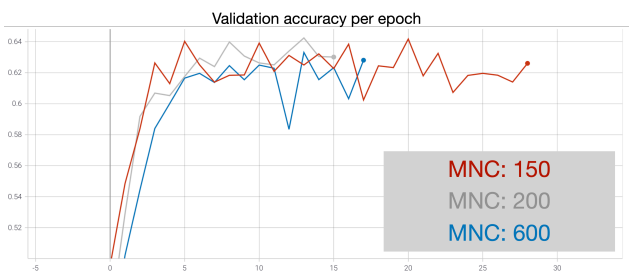


Figure 9 Accuracy results for different max neighbor counts.

6 Conclusions

6.1 Current Status

6.1.1 Best Model

Based on the experimental results from the previous section, our model has the following configuration:

Max neighbor count:	150
Batch size:	128
Weight decay:	1e-4

Table 2 Configuration of the best model.

With this configuration, the following end accuracy results on validation and test dataset are reported:

Validation:	64.17%
Test:	63.47%

Table 3 End accuracy on validation and test dataset.

6.1.2 Results & Comparison

We propose **GraphWalker**, a novel method for KGQA. Our method consists of an encoder-decoder model utilizing attention. Our model can not overperform the state-of-the-art on the MetaQA dataset but shows comparable results with other approaches. The advantage of our model against the other models is that it can work on arbitrary-sized knowledge graphs. Furthermore, it can handle any number of hops, whereas for most approaches it is prohibitively expensive to work with more than 3-hops, due to the large size of the candidate answer subgraph[15].

Model	MetaQA Accuracy (%)
KV-Mem	48.9
VRN	62.5
GraphWalker (Ours)	**63.5**
GraftNet	77.7
PullNet	91.4
EmbedKGQA	94.8

Table 4 Comparison with other methods on the MetaQA dataset

6.2 Future Work

6.2.1 Multiple Answers/Ambiguous Path to Target

The first challenges are the multiple answers to the questions and ambiguous paths to the answers. A question can have more than one valid answer. Further, there can be more than one valid path from the source to the target. In our method, one answer from the valid answers is selected and one shortest path to the selected answer is then used as the ground truth during the model training. After that, during test time, all valid answers are accepted. When all the valid answers and their potential paths are considered during training, it is hard to define an optimal loss for the model.

6.2.2 Greedy Decisions

The second challenge is the greedy decisions made by our model. In each step, our model considers the potential target nodes and chooses the one with the highest probability. By an optimal approach, every path from the source node to the end node can be evaluated and the most likely one among them can be chosen. But this is not possible, due to the large number of paths in the KG. A possible improvement for this problem is to apply beam search. In each step, the best k candidate target nodes can be stored. In this way, the model can keep exploring the best k paths. This improvement could be easier in validation but hard in training time, since it is hard to define a loss for k paths.

6.2.3 Incomplete Graphs

In real-world scenarios, the knowledge graphs can be incomplete. As our method considers the next target node among the connected nodes in each step, it relies on the complete graphs. In the case of missing relations, it would be challenging for our model to find the correct prediction entity. A possible improvement for this problem is a graph completion step. By a graph completion, the relations that might be missing can be predicted and added to the graph as new edges. In this way, our model is likely to perform better on the new graph, as it consists of more relations. This step can be done in a self-supervised way, where the relations can be removed from the existing graph and the removed relations can be predicted. This step can be done as a pre-processing step before the GraphWalker training or can be done end-to-end with the GraphWalker training.

References

[1] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[2] Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J Smola, and Le Song. Variational reasoning for question answering with knowledge graph. In *AAAI*, 2018.

[3] Dingcheng Li, Jingyuan Zhang, and Ping Li. Representation learning for question classification via topic sparse autoencoder and entity embedding. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 126–133, 2018.

[4] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 2787–2795, Red Hook, NY, USA, 2013. Curran Associates Inc.

[5] Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. pages 1321–1331, 01 2015.

[6] Junwei Bao, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. Constraint-based question answering with knowledge graph. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2503–2514, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee.

[7] Antoine Bordes, Jason Weston, and Nicolas Usunier. Open question answering with weakly supervised embedding models, 2014.

[8] Min-Chul Yang, Nan Duan, Ming Zhou, and Hae-Chang Rim. Joint relational embeddings for knowledge-based question answering. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 645–650, Doha, Qatar, October 2014. Association for Computational Linguistics.

[9] Min-Chul Yang, Do-Gil Lee, So-Young Park, and Hae-Chang Rim. Knowledge-based question answering using the semantic embedding space. *Expert Systems with Applications*, 42(23):9086–9104, 2015.

[10] Zihang Dai, Lei Li, and Wei Xu. Cfo: Conditional focused neural question answering with large-scale knowledge bases, 2016.

[11] Li Dong, Furu Wei, Ming Zhou, and Ke Xu. Question answering over Freebase with multi-column convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 260–269, Beijing, China, July 2015. Association for Computational Linguistics.

[12] Yanchao Hao, Yuanzhe Zhang, Kang Liu, Shizhu He, Zhanyi Liu, Hua Wu, and Jun Zhao. An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 221–231, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[13] Denis Lukovnikov, Asja Fischer, Jens Lehmann, and Sören Auer. Neural network-based question answering over knowledge graphs on word and character level. WWW '17, page 1211–1220, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[14] Haitian Sun, Tania Bedrax-Weiss, and William W. Cohen. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. *CoRR*, abs/1904.09537, 2019.

[15] Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4498–4507, Online, July 2020. Association for Computational Linguistics.

[16] Dzmitry Bahdanau, Kyunghyun Cho, and Y. Bengio. Neural machine translation by jointly

learning to align and translate. *ArXiv*, 1409, 09 2014.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[18] E.F. Moore. *The Shortest Path Through a Maze*. Bell Telephone System. Technical publications. monograph. Bell Telephone System., 1959.

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

[20] Alexander H. Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *CoRR*, abs/1606.03126, 2016.

[21] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[22] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015.

[23] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.