Question Decomposition Tree for Answering Complex Questions over Knowledge Bases

Xiang Huang, Sitao Cheng, Yiheng Shu, Yuheng Bao, Yuzhong Qu

State Key Laboratory for Novel Software Technology, Nanjing University, China
{xianghuang, stcheng, yhshu, yhbao}@smail.nju.edu.cn, yzqu@nju.edu.cn

Abstract

Knowledge base question answering (KBQA) has attracted a lot of interest in recent years, especially for complex questions which require multiple facts to answer. Question decomposition is a promising way to answer complex questions. Existing decomposition methods split the question into sub-questions according to a single compositionality type, which is not sufficient for questions involving multiple compositionality types. In this paper, we propose Question Decomposition Tree (QDT) to represent the structure of complex questions. Inspired by recent advances in natural language generation (NLG), we present a two-staged method called `Clue-Decipher` to generate QDT. It can leverage the strong ability of NLG model and simultaneously preserve the original questions. To verify that QDT can enhance KBQA task, we design a decomposition-based KBQA system called `QDTQA`. Extensive experiments show that `QDTQA` outperforms previous state-of-the-art methods on ComplexWebQuestions dataset. Besides, our decomposition method improves an existing KBQA system by 11% and sets a new state-of-the-art on LC-QuAD 1.0.

Introduction

Question answering (QA) is a long-standing challenge in artificial intelligence. With the emergence of knowledge bases (KBs), such as DBpedia (Auer et al. 2007) and Freebase (Bollacker et al. 2008), knowledge base question answering (KBQA) has attracted intensive attention (Lan et al. 2021). However, answering complex questions with multiple hops or constraints is still challenging. The difficulty of linking and compositing KB items (entities, relations, and constraints) grows intractably as questions become complex.

To tackle the problems brought by complex questions, recent works (Min et al. 2019; Talmor and Berant 2018; Zhang et al. 2019) put efforts into question decomposition and achieve remarkable performance. In these methods (sequence-based decomposition), a question is firstly classified into a single compositionality type, i.e., composition (with an inner question) or conjunction (with conjunctive descriptions). Then, the question is split into two sub-questions according to its compositionality type. For the exemplar question in Figure 1, sequence-based methods would classify the question as a conjunction question and split it into two sub-questions (shown in the upper part). However, we find that 39% and 31% of the questions in ComplexWebQuestions (CWQ) (Talmor and Berant 2018) and LC-QuAD 1.0 (LC) (Trivedi et al. 2017), contain more than one compositionality type, respectively. For the above example, the latter sub-question still contains an inner question, namely *"a notable professional athlete who began their career in 1997"*, but sequence-based methods do not further decompose it, preventing this question from being answered correctly. EDGQA (Hu et al. 2021) alleviates such problem by decomposing the question into an entity-centric graph through hand-crafted rules. But such graph structure is complicated and difficult to train with a neural model.

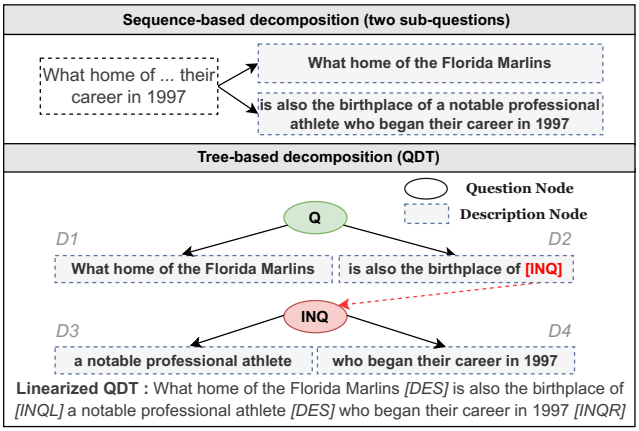

Figure 1: An example of Question Decomposition Tree (QDT) and sequence-based decomposition for the question "What home of the Florida Marlins is also the birthplace of a notable professional athlete who began their career in 1997?". [INQ] is a placeholder for an inner question.

In this paper, we focus on how to make question decomposition an effective way to answer complex questions over KBs. We propose **Question Decomposition Tree (QDT)**, a tree-based decomposition structure to better model the structure of complex questions. An example of QDT is shown in the lower part of Figure 1. The root (green oval) is a question node that represents the whole question. It points to two description nodes, *D1* and *D2* (dashed rectangles). *D2* contains a placeholder "[INQ]" which indicates a sub-

question. In this manner, a QDT can represent the decomposition structure of a question with a combination of multiple compositionality types. We also construct a decomposition benchmark named QDTrees with 6,607 QDTs.

In order to generate QDT, we build our decomposition model Clue-Decipher based on NLG model and propose linearized QDT as target format, which makes it easier to generate a tree-based structure in a neural way. Nevertheless, sometimes the generative method suffers from missing words or generating unexpected words (as shown in Table 1). Clue-Decipher addresses this issue by first generating a preliminary decomposition as a clue then inserting separators to the original question according to the clue. It leverages the strong generation ability of NLG models and simultaneously ensures that the question remains unchanged.

Moreover, we design a QDT-based KBQA system called QDTQA to evaluate the effectiveness of QDT in answering complex questions over KB. Experimental results show that QDTQA achieves state-of-the-art result on CWQ. In addition, our decomposition method improves an existing QA system by 11% and sets a new state-of-the-art on LC[1].

Question: What films featuring **Taylor Swift** have netflix_id numbers above **70068848** **SubQ1:** what films featuring **swift** **SubQ2:** have netflix_id numbers above **Ours:** What films $[DES]$ featuring Taylor Swift $[DES]$ have netflix_id numbers above 70068848
Question: What schools were attended by the characted of focus in the film "**William & Kate**" **SubQ1:** what schools were attended by [INQ] **SubQ2:** the characted of "**characted focus & the film**" **Ours:** What schools $[DES]$ were attended by $[INQL]$ the characted of focus in the film "William & Kate" $[INQR]$

Table 1: Failed cases of another generative method HSP (Zhang et al. 2019), compared with our results. HSP splits each question into two sub-questions, namely SubQ1 and SubQ2. The bold words indicate the missing tokens or unexpected tokens that are in conflict between the original question and sub-questions.

Related Work

Complex KBQA

Complex KBQA aims at answering a question involving more than a single fact over KBs. There are two mainstream methods to solve complex KBQA: information retrieval-based (IR-based) and semantic parsing-based (SP-based) methods (Lan et al. 2021). IR-based methods extract a question-specific subgraph from KB and then rank the candidate entities in the subgraph to get the final answer (Sun, Bedrax-Weiss, and Cohen 2019).

SP-based methods fall into another line and generally perform better. They parse questions into formal queries that can be executed against KB. Traditional SP-based methods follow a staged query graph generation manner, enumerating all possible query structures starting from a topic entity in limited hops, which introduces a large number of noisy candidates (Yih et al. 2015; Chen et al. 2020). Researchers have made efforts in pruning the search space. QGG (Lan and Jiang 2020) considers constraints in query generation and uses beam search to control the search space. AQG (Chen et al. 2020) predicts an abstract query graph to restrict candidate queries. Recently, with the rise of natural language generation, some works cast KBQA to a Seq2Seq task. CBR-KBQA (Das et al. 2021) uses T5 to directly transform a question into a SPARQL and outperforms previous state-of-the-art result by a large margin. RnG-KBQA (Ye et al. 2022) proposes a Rank-and-Generate approach which first adopts a contrastive ranker to rank candidate logic forms, then generates the final logic form in a Seq2Seq manner.

In this paper, we propose QDTQA following the promising SP-based paradigm. By incorporating question decomposition into KBQA system, we outperform state-of-the-art methods on both CWQ and LC datasets.

[1]Our code and dataset are available at https://github.com/cdhx/QDTQA

Question Decomposition

Question decomposition essentially provides an ungrounded query graph that handles structure disambiguation. With the help of the question's structure, the QA system can avoid inefficient traversal of relation paths (Chen et al. 2020). There are mainly three kinds of question decomposition methods:

(1) Splitting-based methods, such as SplitQA (Talmor and Berant 2018) and DecompRC (Min et al. 2019), adopt pointer network to split a question into two parts. While these methods can preserve the original questions, they are too specific to support more complex structure.

(2) Generative methods are more flexible and can be easily extended to different target formats. HSP (Zhang et al. 2019) employs a Seq2Seq model with copy mechanism to generate sub-questions. However, these methods still decompose a question into two parts and can not guarantee that the sentence semantics stay unchanged (as shown in Table 1). As a result, they may lose some tokens or generate unexpected tokens, which would corrupt the semantics of the input question and pose difficulties in evaluating the performance.

(3) Rule-based methods. A representative work is EDG (Hu et al. 2021). It iteratively transforms the constituency tree into an entity-centric graph with hand-crafted rules. Generally, this method has limited coverage and is heavily dependent on constituency parsing. Moreover, the iterative decomposition approach lacks global awareness and can lead to error cascade when further decomposing sub-questions. EDG can handle questions with multiple compositionality types, but how to generate the complex structure in a neural way is challenging.

To better model the structure of question, in this paper, we propose a tree-based decomposition structure called QDT, along with a two-staged method Clue-Decipher to generate QDT. Clue-Decipher can leverage the advantages of both splitting-based methods (questions remaining unchanged) and generative methods (strong generation ability and flexibility).

Question Decomposition Tree

In this section, we propose Question Decomposition Tree (QDT), a tree-based structure, to model the decomposition structure of complex question. We first define QDT and then present our QDT dataset, named `QDTrees`.

Definitions

Given a question q, the QDT of q is a tree containing two types of nodes: the question node and the description node. As shown in the example in the lower part of Figure 1, the root question node (green oval) indicates the original question, whose answer is constrained by two description nodes, *D1* and *D2* (dashed rectangle). Note that *D2* contains a placeholder "[INQ]", indicating the existence of an inner question. The inner question is represented by a subtree, the root of which is an inner question node (red oval) pointed by *D2*. Meanwhile, the inner question node points to two description nodes, *D3* and *D4* (dashed rectangle). The structure of QDT enables it to represent the combinations of multiple compositionality types. We also provide an equivalent linear representation of QDT under the tree illustration by introducing three separators, i.e., $[INQL]$, $[INQR]$, $[DES]$. Among them, $[INQL]$ and $[INQR]$ indicate the left and right boundaries of an inner question, while $[DES]$ splits conjunctive descriptions.

A similar work to QDT is Entity Description Graph (EDG) (Hu et al. 2021), an entity-centric graph structure. Compared to EDG, QDT has three differences. Firstly, QDT supports more kinds of questions. For example, EDG cannot represent the questions whose answers are not entities, e.g., *"Did* $[INQL]$ *Curry join the Warriors* $[INQR]$ *before* $[INQL]$ *LeBron played for the Lakers* $[INQR]$*?"*. Secondly, the structure of EDG (i.e., a graph with three types of nodes and three types of edges) is complicated, rendering it hard to train with a neural model. In comparison, QDT is more concise and has an equivalent linearized representation (as indicated in Figure 1), making it easier to generate tree-based structure in a neural way. Thirdly, QDT keeps the original question unchanged, except inserting some tags, while EDG removes some dummy words, such as *which, that*, etc.

QDT Dataset

To train a model for question decomposition and provide a benchmark, we construct a dataset called `QDTrees`, with 6,607 Question Decomposition Trees of complex questions from existing KBQA datasets.

Data Collection The questions in `QDTrees` are derived from two complex KBQA datasets: ComplexWebQuestions (CWQ) (Talmor and Berant 2018) and LC-QuAD 1.0 (LC) (Trivedi et al. 2017). For CWQ, we annotate three subsets with 2,000/500/500 questions randomly sampled from the training/validation/test sets, respectively. Following (Talmor and Berant 2018), we regard the comparative and superlative questions as conjunction question. Since LC does not provide an official validation set, we split the training set into a new training set (the first 3,200 questions) and a validation set (the last 800 questions). We annotate all complex questions of LC which have more than one triple pattern in SPARQL queries (3,307 questions). Finally, we obtain 6,607 examples in total, detailed in Table 2.

Source	Comp.	Conj.	Comp.&Conj.	Total
CWQ	1,350	2,864	1,214	3,000
Train	900	1,916	816	2,000
Dev	225	473	198	500
Test	225	475	200	500
LC	1,958	3,002	1,554	3,607
Train	1,270	1,895	983	2,313
Dev	308	487	246	576
Test	380	620	325	718
Total	3,308	6,067	2,768	6,607

Table 2: Statistics of `QDTrees`. Comp. and Conj. refer to the number of questions containing the compositionality type of composition and conjunction, respectively.

Annotation We invite four graduate students, majored in Computer Science and familiar with KBQA, to annotate `QDTrees`. Before annotation, they are informed of the detailed instructions with clear examples. The annotation consists of two phases. In the annotation phase, two annotators independently label linearized QDT for all examples. Auxiliary information, including machine questions, intermediary questions, and SPARQL queries, is provided to help understand the meaning of the question. In the validation phase, for the examples with consistent annotations (exact-match inter-annotator agreement is 0.92) in the annotation phrase, the other two annotators check if they agree with the results. For the inconsistent examples, all annotators discuss and determine the final annotation. The annotation lasts for two weeks and takes around 60 hours for each annotator.

Question Decomposition Method

To obtain the tree-based structure, we propose a two-staged decomposition method called `Clue-Decipher`. A running example of `Clue-Decipher` is shown in Figure 2. Instead of deriving the generated decomposition result directly, we regard it as a clue for inserting separators into the original question. Our method consists of ClueNet, used to generate a preliminary decomposition result as a clue, and DecipherNet, used to obtain the inserting positions of separators in the original question.

ClueNet

ClueNet aims to generate a *Clue* for a question. We use T5 (Raffel et al. 2020) as basic model. A *Clue* is a preliminary decomposition which is literally a corrupted question with some separators. In other words, other than the separators, a *Clue* may have some differences from the original question. As shown in the example in Figure 2, the *Clue* loses "Motion Picture" and part of type constrain ("child actor" is changed to "actor"). Besides, it fails to generate the relation (whose **soundtrack** is) between "the movie" and "Forrest Gump: Original Motion Picture" in the original question, but instead generates a relation (whose **child** is) that does not exist.

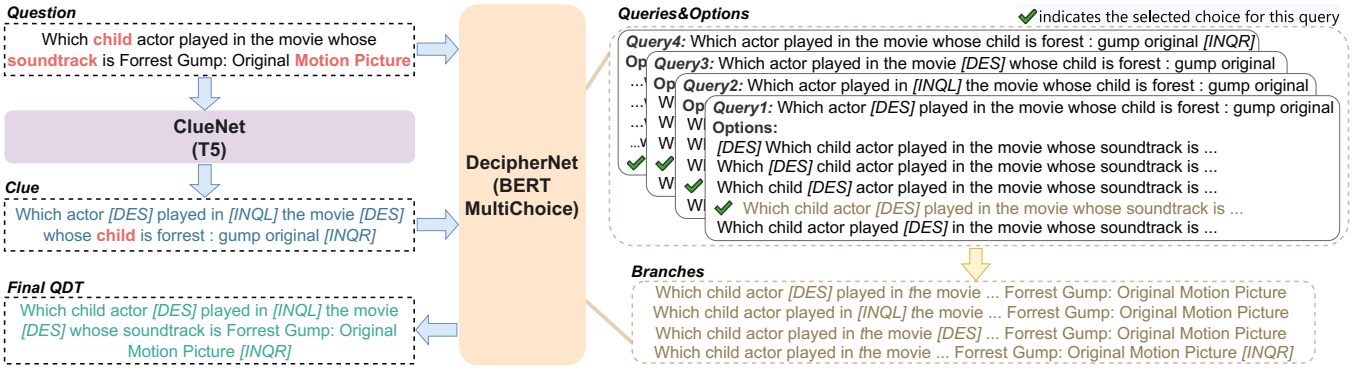

Figure 2: An overview of our two-staged decomposition method `Clue-Decipher`, consisting of ClueNet and DecipherNet. The tokens in red bold are the missing or unexpected tokens that are in conflict between question and clue.

DecipherNet

DecipherNet aims to locate the inserting position of each separator in the original question, according to the *Clue* obtained from ClueNet. Separators in the *Clue* are processed in turn. As shown in the example in Figure 2, we first break the *Clue* down into several *Queries* by preserving only one separator at a time. As a result, a *Clue* with k separators derives k *Queries*.

For each *Query*, we construct five *Options* indicating the possible positions for a separator in the original question. After that, DecipherNet selects the most possible option, called *Branch*, such process is formulated as a multiple-choice task. For *Query1* in Figure 2, we first estimate that the approximate inserting position of the first $[DES]$ is 2, according to its token position. We provide five *Options* by inserting $[DES]$ into the original question in positions from 0 to 4. Then, DecipherNet selects the best choice (*Branch*), from the *Options*. Compared to the *Query*, with a separator in a corrupted question, the *Branch* is presented by inserting a separator into the original question. After obtaining all the *Branches*, we merge them to form a QDT.

Training Data Collection

DecipherNet selects the *Branch* for each *Query* from some *Options*. Note that the only difference between *Query* and *Branch* is that a *Query* is a corrupted question with a separator, while a *Branch* is the original question with a separator. Therefore, we formulate the training data collection in three steps: Firstly, we break a golden QDT down into several golden *Branches* by retaining only one separator at a time. Secondly, we construct some negative *Options* for each *Branch* by shifting the separator to other 4 neighboring positions. Thirdly, we corrupt the *Branches* to construct the *Queries*. For each token in a *Branch*, we have 4 ways to corrupt it, the probability of each way is a predefined hyperparameter given in parentheses: (1) replace this token with a random word in this *Branch* (1%), (2) delete this token (1%), (3) add another random token in this *Branch* after this token (1%), (4) convert this token to its vocabulary id and reconvert the id to the token by the same tokenizer (97%).

Evaluation of Decomposition Method

We evaluate the decomposition quality of different decomposition methods from two aspects: tree-based evaluation and sequence-based evaluation.

Baselines

We compare our method with four baseline methods. (1) SplitQA (Talmor and Berant 2018) finds split points with a pointer network. (2) DecompRC (Min et al. 2019) formulates decomposition as a span prediction task. (3) HSP (Zhang et al. 2019) leverages a Seq2Seq model to generate sub-questions. (4) EDGQA (Hu et al. 2021) employs handcrafted rules to generate a graph to represent the question. The first three methods are sequence-based methods which are designed to split a question into two sub-questions while EDGQA is a graph-based method.

Implementation Details

Our decomposition model is based on Pytorch (Paszke et al. 2019) and Hugging Face (Wolf et al. 2020). We use T5-base with Adafactor optimizer for ClueNet, and BERT-base with SGD optimizer for DecipherNet. The batch sizes for ClueNet and DecipherNet are set to 64. We train our models for 100 epochs on an NVIDIA GeForce RTX 3090 GPU and save the best checkpoints on the validation set. All methods except EDGQA are trained and tested on `QDTrees`.

Tree-based Evaluation

Metrics We consider three metrics:

Exact Match (EM) denotes whether the predicted decomposition is completely the same as the golden one.

Tree Depth Accuracy (TDA) denotes the ratio of generated decomposition whose depth is equal to the golden ones.

Graph Edit Distance (GED)[2] is to measure the minimal transition cost from the predicted decomposition to the golden one. The lower the GED score, the better. Three edit operations (addition, deletion, and substitution) are considered, with predefined cost following (Wolfson et al. 2020).

[2]We use the implementation provided by https://networkx.org/

Method	EM	TDA	GED
EDGQA (Hu et al. 2021)	-	0.7315	0.3799
Clue-Decipher	**0.8332**	**0.9650**	**0.0554**
w/o DecipherNet	0.8130	**0.9650**	0.0558

Table 3: Tree-based Decomposition evaluation.

Method	EM	BLEU	ROUGE
SplitQA (Talmor and Berant 2018)	0.653	0.734	0.905
DecompRC (Min et al. 2019)	0.862	0.954	0.988
HSP (Zhang et al. 2019)	0.252	0.679	0.881
HSP + DecipherNet	0.793	0.935	0.983
Clue-Decipher	**0.909**	**0.970**	**0.993**
w/o DecipherNet	0.889	0.966	0.991

Table 4: Sequence-based Decomposition evaluation.

Results As shown in Table 3. Our method achieves 0.8332 on EM, and significantly outperforms EDGQA on GED and TDA. We do not report the EM of EDGQA because it cannot be trained on QDTrees and removes some dummy words, making it hard to match with our annotation.

To demonstrate the effectiveness of our DecipherNet, we perform an ablation study by removing DecipherNet from Clue-Decipher. Results show that Clue-Decipher is superior to the bare ClueNet by 2.02% on tree-based EM. This suggests that our proposed DecipherNet can further promote generative method. Besides, we also find that none of the example become worse after the incorporation of DecipherNet, which means DecipherNet is stable and safe as an external module for generative decomposition method. Since we adopt separator-insertion according to *Clue*, all generated QDTs do not change any information in the original question, which prevents downstream tasks from error propagation.

Sequence-based Evaluation

Metrics To compare with sequence-based methods, we degrade Clue-Decipher to decompose a question into only two sub-questions. We evaluate the performance from two aspects: Exact Match and Text Similarities. In order to make a comprehensive evaluation with other generative methods, we use two text similarity metrics introduced in HSP (Zhang et al. 2019), i.e., BLEU-4 (Papineni et al. 2002) and ROUGE-L (Lin 2004).

Results Table 4 shows that even with the degradation of generating only 2 parts, Clue-Decipher still consistently surpasses other methods on both Exact Match and Text Similarities. Concretely, our method achieves the highest EM score of 0.909, exceeding DecompRC by 4.7%. Besides, by removing DecipherNet, the sequence-based EM decreases from 0.909 to 0.889. Among the baselines, HSP is a generative-based decomposition method and achieves a poor performance on EM. Intuitively, our DecipherNet can be leveraged to revise its result. After using DecipherNet, the performance increases notably on all three metrics. It shows that our DecipherNet is easy to be adapted to other generative methods and reveal their potential performance.

KBQA based on QDT

In this section, we present QDTQA, following the Seq2Seq manner and employing QDT to promote the performance. The framework is shown in Figure 3. QDTQA receives the concatenation of question, QDT, and candidate entities as input and yields an executable query.

Entity Linking

To get candidate entities, we adopt off-the-shelf tools: ELQ (Li et al. 2020) and FACC1 (Gabrilovich and Subramanya 2013). ELQ is an end-to-end linking model by dense retrieval and returns candidate entities $CandEnt_{ELQ}$ with confidence scores. FACC1 is a large mapping from entity mention to Freebase ID. We utilize a NER model provided by GrailQA (Gu et al. 2021) to detect mentions and match with FACC1 through string similarity to get $CandEnt_{FACC1}$. We adopt a BERT-based sentence classification model to rank $CandEnt_{FACC1}$ and merge the result with $CandEnt_{ELQ}$ to get the final candidate entities.

Query Normalization

In order to better adapt Pre-trained Language Model (PLM) to a KBQA task, we consider **normalized S-expression** as target format of query generation model instead of SPARQL. The conversion from SPARQL to S-expression is accomplished via the script provided by (Ye et al. 2022). After that, we make some modifications to the entities and relations of S-expression to get the normalized S-expression. In detail, for each entity, the Freebase ID is mapped to its label via *rdfs:labels*. For each relation, it is converted into a word list which is closer to nature language. Furthermore, we surround each KB element, including entity, relation, and literal, with a pair of brackets to remind the language model that this is a KB element. For example, in Figure 3, the Freebase entity "m.02_x" is mapped to "[Miami Marlins]" and the relation "location.location.people_born_here" is converted to " [location, location, people born here]".

Training Query Generation Model

Following (Das et al. 2021), we employ T5-base as our backbone query generation model. Given the question Q and its golden SPARQL sq, we first convert sq to normalized S-expression $Sexp$, according to Query Normalization. After that, we extract golden entities Ent from sq, then map each entity e in Ent to its label e_l to obtain Ent_{label}. We construct the model input by concatenating Q with the linearized QDT QDT and Ent_{label}. Such input is fed to T5 encoder to obtain the representation $H_{Sexp'}$. After that, the T5 decoder decodes $H_{Sexp'}$ to a logic form $Sexp'$ token by token. We optimize the cross-entropy loss between $Sexp'$ and $Sexp$ using teacher forcing.

$$H_{Sexp'} = T5Encoder([Q; QDT; Ent_{label}]) \quad (1)$$

$$Sexp' = T5Decoder(H_{Sexp'}) \quad (2)$$

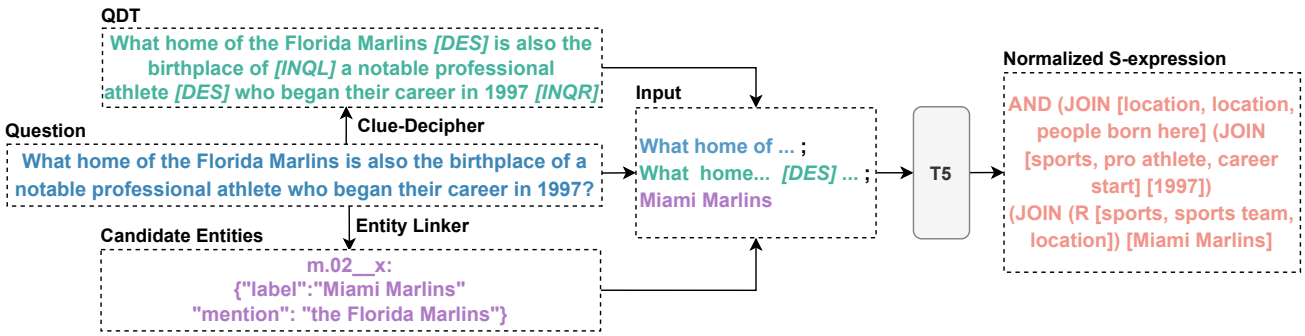

Figure 3: An overview of QDTQA.

Inference

During inference, we first conduct entity linking to get candidate entities and concatenate them with question and QDT as the input of the query generation model. After obtaining the normalized S-expression, we convert each normalized element, including the entity label, modified relation, and literal, to its KB representation to get S-expression. The S-expression is converted to an executable SPARQL query by the script provided by (Gu et al. 2021). The SPARQLs that can not be executed against KB are skipped. We regard the result of the first SPARQL that returns a non-empty result as the final answer.

Evaluation of KBQA Method

In this section, we evaluate the performance of QDTQA and apply Clue-Decipher to another KBQA system: EDGQA (Hu et al. 2021).

Implementation Details

QDTQA : We employ T5-base as the query generation model and train with AdamW optimizer. The batch size is set to 16 and the max length is set to 196. The entity disambiguation model is based on BERT-base, in which we set batch size to 16 and max length to 96.

EDGQA + Clue-Decipher : We replace the decomposition method in EDGQA with Clue-Decipher. Other settings remain the same for fair comparison.

Datasets

We conduct experiments on two popular complex question answering datasets: CWQ (for QDTQA) and LC (for EDGQA + Clue-Decipher).

ComplexWebQuestions (CWQ) (Talmor and Berant 2018) contains 34,689 complex questions over Freebase (version 2015-08-09). The train/validation/test sets contain 27,639/3,519/3,531 questions, respectively.

LC-QuAD 1.0 (LC) (Trivedi et al. 2017) contains 5,000 questions over DBpedia (2016-04), with 4,000 training and 1,000 testing questions. 72.64% questions of LC are complex questions, which means the corresponding SPARQL queries have more than one triple patterns.

Baselines

We compare QDTQA with the following three systems on CWQ. (1) (Qin et al. 2021) gradually shrinks knowledge base to a desired query graph. (2) (Huang, Kim, and Zou 2021) directly generate SPARQL query without simplification. (3) CBR-KBQA (Das et al. 2021) generates logical forms by retrieving relevant cases.

EDGQA + Clue-Decipher is compared with: (1) EDGQA (Hu et al. 2021) generates sub-queries node-by-node following a decomposition, and then composes the final query. (2) NSQA (Kapanipathi et al. 2021) adopts AMR to get shallow semantic parsing of questions. (3) (Liang et al. 2021) propose a BERT-based decoder to extract triple patterns. (4) STaG-QA (Ravishankar et al. 2022) separates the semantic parsing process from knowledge base interaction and achieves notable performance on multiple KBs.

QA results of baselines are from their original paper.

Metrics

We follow the metrics considered in baseline methods. For CWQ, we use Average F1 (Avg. F1) and Accuracy (Acc). For LC, we use Precision (P), Recall (R) and Macro F1 (F1).

Result and Analysis on QDTQA

Main Results The result of QDTQA on CWQ is summerized in Table 5. QDTQA significantly outperforms previous methods. To verify the effectiveness of question decomposition in QDTQA, we remove QDT from the input. Result shows that, without QDT, QDTQA drops 1.3% on average F1, which illustrates that our decomposition can improve the performance of downstream KBQA task. We also evaluate the performance of a model, replacing tree-based decomposition with sequence-based decomposition, with an average F1 decrease of 0.8%. It demonstrates that tree-based decomposition can provide richer information and is important for the representation of complex questions.

The Effectiveness of Question Decomposition We further analyze how QDT contributes to the result, from two critical factors of KBQA, i.e., structure and relation linking. We analyze all questions on the test set of CWQ. After using QDT, additional 7.31% questions derive correct query structures. We believe that QDT can provide a structural guidance to avoid invalid searching of noisy query structures.

Method	Avg. F1	Acc
(Qin et al. 2021)	0.462	-
(Huang, Kim, and Zou 2021)	0.682	-
T5-11B + Revise (Das et al. 2021)	0.582	0.556
CBR-KBQA (Das et al. 2021)	0.700	0.671
QDTQA	**0.728**	**0.679**
w/o QDT	0.715	0.666
w/o tree-based structure	0.720	0.670
w/ SplitQA	0.716	0.669
w/ DecompoRC	0.716	0.669
w/ HSP	0.717	0.669
w/ EDGQA	0.714	0.665

Table 5: QA performance of QDTQA on CWQ compared with baselines. We also report the performance with different decomposition methods.

Besides, relation linking can also benefit from question decomposition. Since a decomposition, beforehand, separates different semantic representations, which reduces the search space and avoids the distraction of other relations. In fact, 10.53% questions get better relation linking results after the incorporation of QDT.

The Impact of Different Decomposition Methods To evaluate the impact of different decomposition methods on QA system, we replace the decomposition result in QDTQA with four other decomposition methods. Results in Table 5 demonstrate that our decomposition consistently outperforms other methods. Combining the results of the previous experiments, it shows that our method is superior to others on both decomposition and question answering.

Error Analysis We randomly sample 100 questions that are incorrectly answered by QDTQA (F1 <1.0) and categorize them into three types.

Question Decomposition (5%). We manually verify whether a decomposition is acceptable under the same standard for annotation. For example, *"What movies have been written by* $[INQL]$ *authors of* $[DES]$ *monty python"* is a failed case.

Query Structure (44%). Even with a correct decomposition structure, it is still possible for QDTQA to generate an S-expression that does not match the golden one at the structure level. The reason lies on the fact that QDTQA utilizes QDT as an implicit restriction rather than forcing the system to strictly follow this structure.

Entity and Relation Linking (51%). Linking is still the main obstacle for KBQA. There are 26% and 25% errors caused by entity and relation linking, respectively.

Result on EDGQA + Clue-Decipher

To further verify that our decomposition method can enhance existing decomposition-based KBQA system, we conduct another experiment, incorporating our decomposition result on a state-of-the-art decomposition-based KBQA system, i.e., EDGQA (Hu et al. 2021). We adopt EDGQA as a basic system for two reasons. Firstly, it is the state-of-the-art decomposition-based system on a complex question answering dataset (LC). Secondly, it is the only open-source decomposition-based KBQA system in recent years.

Method	P	R	F1	ΔF1
NSQA	0.448	0.458	0.445	-
STaG-QA	**0.745***	0.548	0.536	-
(Liang et al. 2021)	0.511	0.593	0.549	-
EDGQA	0.505	0.560	0.531	0
w/ SplitQA	0.496	0.576	0.533	+0.002
w/ DecompRC	0.521	0.609	0.561	+0.030
w/ HSP	0.433	0.507	0.467	-0.064
w/ Clue-Decipher	0.548	**0.635**	**0.588**	**+0.056**

Table 6: QA performance of baseline methods and EDGQA + Clue-Decipher on LC. We also replace Clue-Decipher with other decomposition methods. * indicates that when calculating P, STaG-QA defines the empty answer to have P=1, different from others.

Table 6 compares EDGQA + Clue-Decipher with other methods on LC. Results show that our decomposition significantly improves EDGQA by 11% on F1 and sets a new state-of-the-art on LC, which demonstrates the utility of our decomposition method. We also report the performance of different decomposition methods on EDGQA in Table 6. The baseline methods are the same as discussed in decomposition evaluation. Clue-Decipher consistently outperforms other decomposition methods, which further verifies the superiority of our decomposition method.

Conclusion

In this paper, we focus on how to make question decomposition an effective way to answer complex questions over KBs. To summarize, we make the following contributions:

- We propose Question Decomposition Tree (QDT) to represent the decomposition structure of complex questions, along with a dataset QDTrees with 6,607 QDTs. We also present a linearized representation of QDT, making it easier to be generated in a neural way.
- We propose Clue-Decipher to generate QDTs that consistently surpass other methods from multiple aspects. It leverages the strong generation ability of NLG models and ensures that the original question remains unchanged.
- We design a QDT-based KBQA system called QDTQA which achieves state-of-the-art performance on CWQ. Besides, we demonstrate that our decomposition can further enhance an existing system on LC by 11%.

In the future, an interesting topic is to automatically generate decomposition annotations from existing pairs of questions and SPARQLs. Besides, QDT can be extended to model other linguistic phenomena, such as disjunction, which is worth exploring.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (NSFC) under Grant No. 62072224. We would like to thank Xixin Hu and Xuan Wu for their efforts in the early stages of this work. We also thank the anonymous reviewers for their constructive comments.

References

Auer, S.; Bizer, C.; Kobilarov, G.; Lehmann, J.; Cyganiak, R.; and Ives, Z. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*, 722–735. Springer.

Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; and Taylor, J. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the SIGMOD*, 1247–1250.

Chen, Y.; Li, H.; Hua, Y.; and Qi, G. 2020. Formal Query Building with Query Structure Prediction for Complex Question Answering over Knowledge Base. In Bessiere, C., ed., *Proceedings of the AAAI*, 3751–3758. ijcai.org.

Das, R.; Zaheer, M.; Thai, D.; Godbole, A.; Perez, E.; Lee, J.-Y.; Tan, L.; Polymenakos, L.; and McCallum, A. 2021. Case-based Reasoning for Natural Language Queries over Knowledge Bases. *EMNLP*, 9594–9611.

Gabrilovich, R. M., E.; and Subramanya, A. 2013. Facc1: Freebase annotation of clueweb corpora, version 1 (release date 2013-06-26, format version 1, correction level 0). http://lemurproject.org/clueweb09/FACC1/. Accessed: 2013-06-26.

Gu, Y.; Kase, S.; Vanni, M.; Sadler, B.; Liang, P.; Yan, X.; and Su, Y. 2021. Beyond IID: three levels of generalization for question answering on knowledge bases. In *Proceedings of the Web Conference*, 3477–3488.

Hu, X.; Shu, Y.; Huang, X.; and Qu, Y. 2021. EDG-Based Question Decomposition for Complex Question Answering over Knowledge Bases. In *Proceedings of the ISWC*, 128–145.

Huang, X.; Kim, J.-J.; and Zou, B. 2021. Unseen Entity Handling in Complex Question Answering over Knowledge Base via Language Generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, 547–557. Punta Cana, Dominican Republic: Association for Computational Linguistics.

Kapanipathi, P.; Abdelaziz, I.; Ravishankar, S.; Roukos, S.; Gray, A.; Astudillo, R.; Chang, M.; Cornelio, C.; Dana, S.; Fokoue, A.; Garg, D.; Gliozzo, A.; Gurajada, S.; Karanam, H.; Khan, N.; Khandelwal, D.; Lee, Y. S.; Li, Y.; Luus, F.; Makondo, N.; Mihindukulasooriya, N.; Naseem, T.; Neelam, S.; Popa, L.; Reddy, R.; Riegel, R.; Rossiello, G.; Sharma, U.; Bhargav, G. P.; and Yu, M. 2021. Leveraging Abstract Meaning Representation for Knowledge Base Question Answering. *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, (c): 3884–3894.

Lan, Y.; He, G.; Jiang, J.; Jiang, J.; Zhao, W. X.; and Wen, J. 2021. A Survey on Complex Knowledge Base Question Answering: Methods, Challenges and Solutions. In Zhou, Z., ed., *Proceedings of the IJCAI*, 4483–4491.

Lan, Y.; and Jiang, J. 2020. Query Graph Generation for Answering Multi-hop Complex Questions from Knowledge Bases. In *Proceedings of the ACL*, 969–974.

Li, B. Z.; Min, S.; Iyer, S.; Mehdad, Y.; and Yih, W.-t. 2020. Efficient One-Pass End-to-End Entity Linking for Questions. In *Proceedings of the EMNLP*, 6433–6441.

Liang, Z.; Peng, Z.; Yang, X.; Zhao, F.; Liu, Y.; and McGuinness, D. L. 2021. BERT-based Semantic Query Graph Extraction for Knowledge Graph Question Answering. In *Proceedings of the ISWC*.

Lin, C.-Y. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*, 74–81. Barcelona, Spain: Association for Computational Linguistics.

Min, S.; Zhong, V.; Zettlemoyer, L.; and Hajishirzi, H. 2019. Multi-hop reading comprehension through question decomposition and rescoring. In *Proceedings of the ACL*, 6097–6109.

Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W.-J. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 311–318. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Qin, K.; Li, C.; Pavlu, V.; and Aslam, J. A. 2021. Improving Query Graph Generation for Complex Question Answering over Knowledge Base. *EMNLP*, 4201–4207.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. T5: Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21: 1–67.

Ravishankar, S.; Thai, J.; Abdelaziz, I.; Mihidukulasooriya, N.; Naseem, T.; Kapanipathi, P.; Rossilleo, G.; and Fokoue, A. 2022. A Two-Stage Approach towards Generalization in Knowledge Base Question Answering. *AAAI*.

Sun, H.; Bedrax-Weiss, T.; and Cohen, W. 2019. PullNet: Open Domain Question Answering with Iterative Retrieval on Knowledge Bases and Text. In *Proceedings of the EMNLP-IJCNLP*, 2380–2390.

Talmor, A.; and Berant, J. 2018. The web as a knowledge-base for answering complex questions. In *Proceedings of the NAACL-HLT*, 641–651.

Trivedi, P.; Maheshwari, G.; Dubey, M.; and Lehmann, J. 2017. Lc-quad: A corpus for complex question answering over knowledge graphs. In *the Proceedings of ISWC*, 210–218.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; Davison, J.; Shleifer, S.; von Platen, P.; Ma, C.; Jernite, Y.; Plu, J.; Xu, C.; Le Scao, T.; Gugger, S.; Drame, M.; Lhoest, Q.; and Rush, A. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the EMNLP: System Demonstrations*, 38–45. Online: Association for Computational Linguistics.

Wolfson, T.; Geva, M.; Gupta, A.; Goldberg, Y.; Gardner, M.; Deutch, D.; and Berant, J. 2020. Break It Down: A Question Understanding Benchmark. *Trans. Assoc. Comput. Linguistics*, 8: 183–198.

Ye, X.; Yavuz, S.; Hashimoto, K.; Zhou, Y.; and Xiong, C. 2022. RNG-KBQA: Generation Augmented Iterative Ranking for Knowledge Base Question Answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 6032–6043. Dublin, Ireland: Association for Computational Linguistics.

Yih, W. T.; Chang, M. W.; He, X.; and Gao, J. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the ACL-IJCNLP*, 1321–1331. ISBN 9781941643723.

Zhang, H.; Cai, J.; Xu, J.; and Wang, J. 2019. Complex question decomposition for semantic parsing. In *Proceedings of the ACL*, 4477–4486.