A Transition-based Method for Complex Question Understanding

Yu Xia[1,*], **Wenbin Jiang**[2], **Yajuan Lyu**[2], **Sujian Li**[1,†]
[1]MOE Key Lab of Computational Linguistics, Peking University, Beijing, China
[2]Baidu Inc., Beijing, China
{yuxia, lisujian}@pku.edu.cn
{jiangwenbin, lvyajuan}@baidu.com

Abstract

Complex Question Understanding (CQU) parses complex questions to Question Decomposition Meaning Representation (QDMR) which is a sequence of atomic operators. Existing works are based on end-to-end neural models which do not explicitly model the intermediate states and lack interpretability for the parsing process. Besides, they predict QDMR in a mismatched granularity and do not model the step-wise information which is an essential characteristic of QDMR. To alleviate the issues, we treat QDMR as a computational graph and propose a transition-based method where a *decider* predicts a sequence of actions to build the graph node-by-node. In this way, the partial graph at each step enables better representation of the intermediate states and better interpretability. At each step, the *decider* encodes the intermediate state with specially designed encoders and predicts several candidates of the next action and its confidence. For inference, a *searcher* seeks the optimal graph based on the predictions of the *decider* to alleviate the error propagation. Experimental results demonstrate the parsing accuracy of our method against several strong baselines. Moreover, our method has transparent and human-readable intermediate results, showing improved interpretability.

1 Introduction

The task of complex question understanding (CQU) aims at converting complex questions which require multi-hop reasoning into consecutive trivial sub-questions. An example of CQU is shown in Figure 1. To answer the question "return me the author in the University of Michigan whose papers have more than 5000 total citations", CQU models decompose it into several trivial sub-questions (*e.g.* "return authors"), and the final answer is obtained by consecutively answering the sub-questions. To

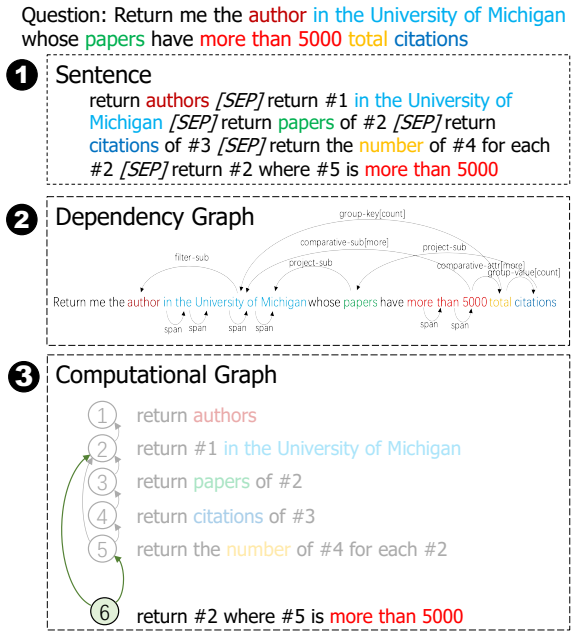

Figure 1: An example of CQU consists of a complex question and different modeling of QDMR.

capture the meaning of questions over unstructured sources such as text and images, Question Decomposition Meaning Representation (QDMR) (Wolfson et al., 2020) is proposed where questions are represented through a sequence of atomic executable operators, and the final answer can be obtained by answering the operator sequences in order. QDMR has been shown to improve the performance and interpretability for multi-hop question answering (Hasson and Berant, 2021; Subramanian et al., 2020; Talmor et al., 2021).

Existing works for CQU can be roughly divided into two categories: the seq2seq-based autoregressive parser (Wolfson et al., 2020) and the dependency-based non-autoregressive parser (Hasson and Berant, 2021). However, these approaches are based on end-to-end models which do not explicitly model the intermediate states and lack interpretability for the parsing process. Besides, they

[*] This work was done during internship at Baidu Inc.
[†] Corresponding author.

Proceedings of the 29th International Conference on Computational Linguistics, pages 4203–4211
October 12–17, 2022.

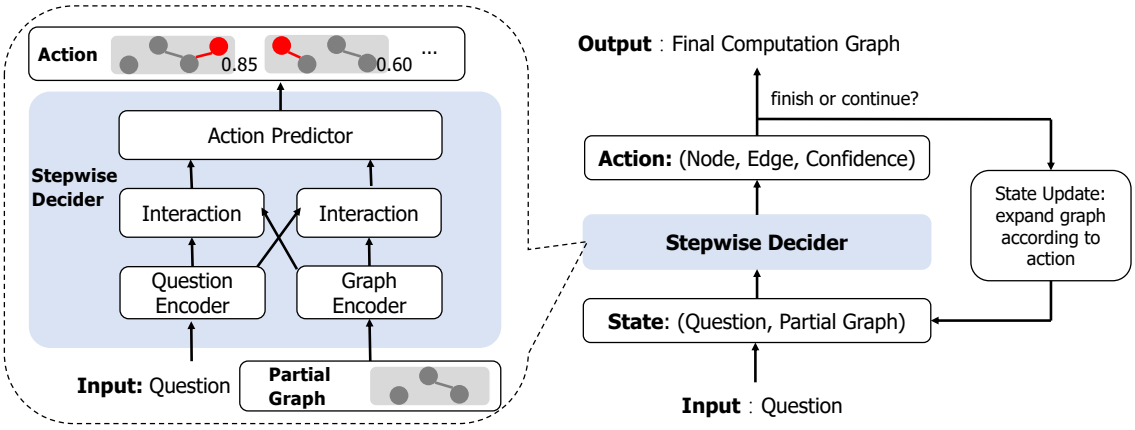

Figure 2: An overview of our transition-based framework.

predict QDMR in a mismatched granularity. The former category generates QDMR as a sentence (as shown in Figure 1 ❶) and adopts the seq2seq model to decode the QDMR token-by-token. This token-level modeling is sub-optimal since it ignores the inherent operator-level structure of QDMR and thus performs worse when the QDMR has longer operators and when the question is informative. The latter category maps QDMR to a dependency graph over the question tokens (as shown in Figure 1 ❷) and adopts a non-autoregressive parser to decode the entire dependency graph in a single step. It predicts all operators in the QDMR simultaneously and ignores the interaction between operators, trading off performance for computational efficiency. (Hasson and Berant, 2021) also tries combining the two approaches by exploiting the graph supervision to train the encoder in the seq2seq model. To sum up, these methods have drawbacks in modeling the intermediate states and the step-wise information which is a distinct characteristic of QDMR.

To alleviate the shortcomings of the above methods while preserving their advantages, we treat QDMR as a computational graph and propose a transition-based method where a *decider* predicts a sequence of actions to build the graph node-by-node. At each transition step, one new node and its referencing edges are decided given the question and the previously generated partial graph. In this way, the partial graph at each step enables better representation of the intermediate states and better interpretability. We illustrate the proposed method in Fig. 2. The generated graph starts from empty and expands incrementally in a node-by-node manner. At each step, given the question and the current state, the *step-wise decider* encodes them with specially designed encoders and predicts several candidates for the next action which includes a node, its connecting edges, and its confidence. After each step, the partial graph is either expanded according to the action or finalized as a QDMR. For inference, a *searcher* seeks the optimal graph based on the predictions of the *decider* to alleviate the error propagation.

To verify the effectiveness of our proposed method, abundant experiments are conducted on the BREAK dataset, which contains 83,978 examples from ten QA datasets across three modalities. Experimental results show that our method outperforms strong baselines and achieves the state-of-the-art on the BREAK dataset. We further analyze the interpretability of our method. Overall, our work makes the following major contributions:

1. To the best of our knowledge, we are the first to investigate the transition-based method for CQU by modeling the intermediate states which facilitate better encoding and better interpretability.

2. Experiments on BREAK demonstrate the parsing accuracy of our method against strong baselines. Moreover, further analysis and visualization verify the interpretability.

2 Method

2.1 Problem Definition

Given a question with n tokens, $q = (q_1, q_2, \ldots, q_n)$, the goal of CQU is to parse it to its QDMR. In this work, we treat QDMR as a computational graph $G = \langle V, E \rangle$ where V

and E are the node set and edge set. A node $v^i \in V$ is a sequence of $|v^i|$ tokens $v^i = v^i_1 ... v^i_{|v^i|}$, where token v^i_j is either a question token $\in \mathcal{V}_q$ (or some inflection of it), a word from a constant predefined lexicon $\in \mathcal{V}_{const}$, or a reference token $\in \mathcal{V}^i_{ref} = \{\#1, ..., \#(i-1)\}$ referring to a previous step. A directed edge $e^{ij} \in E$ pointing from v^i to v^j is the reference token $\#j$ in v^i.

2.2 Overview

To address the aforementioned challenges in this task, we propose a neural transition-based model for CQU which decides a new node along with its connections to existing nodes at each step to incrementally build a computational graph. The construction process is briefly illustrated in Fig. 2. At each step i, given the question q and the current state $s_i = \{G^i\}$ where $G^i = \langle V^i, E^i \rangle$ denotes the generated partial graph, the *stepwise decider* predicts several candidates of the next action $a^{i+1} = \langle v^{i+1}, \Delta E^i, p^{i+1} \rangle$, where v^{i+1} denotes the next node, ΔE^i denotes the edges start from v^{i+1} and p^{i+1} denotes the confidence of the action. Then, we expand the graph from G^i to $G^{i+1} = \langle V^i + v^{i+1}, E^i + \Delta E^i \rangle$ and update the current state according to the predicted action. We repeat the above iteration until the end action is predicted. In inference, we adopt a *searcher* to maintain and seek the optimal graphs based on the node confidence at each step.

2.3 State Representation

At each step i, given the question q and the current state $s_i = \{G^i\}$ where q denotes the question and $G^i = \langle V^i, E^i \rangle$ denotes the partial graph, we firstly use two encoders to obtain their representation respectively. Then, we feed them into a dual interaction layer to update them dynamically.

2.3.1 Question Encoder

We feed the input question $q = (q_1, q_2, \ldots, q_n)$ into the Transformer encoder of a pretrained seq2seq model (e.g. BART) to get the contextual representation matrix $H^q \in \mathbb{R}^{n \times d_h}$, where d_h is the dimension of the hidden states in BART and n is the length of question. In this way, question q can be represented as $H^q = \{\mathbf{h}^q_1, \mathbf{h}^q_2, \ldots, \mathbf{h}^q_n\}$, where $\mathbf{h}^q_i$ is the contextual representation of the i-th token of q. We call H^q static question representation to distinguish it with the dynamically updated question representation H^{q^i} introduced in section 2.3.3.

2.3.2 Graph Encoder

We treat the incrementally expanding graph as a sequence of actions in the chronological order of when they are added in. We adopt the order given in the dataset. We utilize the Transformer decoder to serve as the graph encoder . Concretely, we use the masked self-attention mechanism to ensure that the representation of the node and edges at step i takes all previous nodes and edges in G^{i-1} into consideration.

Formally, given the graph G^i, we get the input tokens of the linearized graph by seperating actions with special tokens to indicate their boundaries: $([\mathrm{A}_1], v^1_1, \ldots, v^1_{|v^1|}, [\mathrm{A}_2], \ldots, [\mathrm{A}_i], v^i_1, \ldots, v^i_{|v^i|})$. If v^i is the last action in the graph, we append a special token [END] to indicate the termination of the parsing process. We feed the input tokens into the graph encoder to get the contextual representation matrix $H^g \in \mathbb{R}^{p \times d_h}$, where p is the length of the input tokens. In this way, G^i can be represented as $H^{G^i} = \{\mathbf{h}^{G^i}_1, \mathbf{h}^{G^i}_2, \ldots, \mathbf{h}^{G^i}_p\}$, where $\mathbf{h}^{G^i}_i$ is the contextual representation of the i-th token of the input tokens. We repeat the above encoding every time a new token in the next action i.e. v^{i+1}_j is generated to integrate the partial action semantic.

2.3.3 Interaction Layer

We observe that different nodes tend to use different parts of the question, and that question tokens already present in the partial graph are less likely to be chosen in the later nodes. To model this observation, we apply the scaled dot-product attention proposed in (Vaswani et al., 2017) to dynamically update the question representation according to the generated partial graph.

$$\begin{aligned} \alpha^i_1 &= \mathrm{softmax}(\frac{W^K_1 H^q (W^Q_1 H^{G^i})^\mathsf{T}}{\sqrt{d_k}}), \\ H^{q^i} &= \alpha^i_1 W^V_1 H^q \end{aligned} \tag{1}$$

where $\{W^Q_1, W^K_1\} \in \mathbb{R}^{d_h \times d_h}$ denote learnable matrices that transform the graph and question representation into the query and key subspace respectively. The attention weights over all question tokens $\alpha^i \in \mathbb{R}^{|q|}$ softly indicate whether a token is already present in the partial graph. $W^V_1 \in \mathbb{R}^{d_h \times d_h}$ denotes the learnable matrix that projects the question representation into the value subspace, and the projected value vectors are averaged according to α^i to get the updated question tokens representation H^i. Similarly, to make the representation of

G^i attends to all question tokens in q, we apply the source-attention mechanism which takes the output of the question encoder as the key:

$$\alpha_2^i = \text{softmax}(\frac{W_2^K H^{G^i}(W_2^Q H^q)^\mathsf{T}}{\sqrt{d_k}}), \quad (2)$$
$$\hat{H}^{G^i} = \alpha_2^i W_2^V H^{G^i}$$

Finally, we apply MaxPooling over all question tokens representation and node tokens representation to obtain the final representation for action prediction:

$$h_{\text{final}} = \text{MaxPooling}(H^{q^i}, \hat{H}^{G^i}) \quad (3)$$

where the MaxPooling is performed on the first dimension and $h_{\text{final}} \in \mathbb{R}^{d_h}$.

2.4 Action Prediction

At each step i, given the representation of the current state h_{final}, we use the action predictor to predict several candidates of the next action $a^{i+1} = \langle v^{i+1}, \Delta E^i, p^{i+1} \rangle$, where v^{i+1} denotes the next node, ΔE^i denotes the edges start from v^{i+1} and p^{i+1} denotes the confidence of the action.

2.4.1 Node Prediction

Note that as mentioned in 2.1 , a node v^{i+1} is a sequence of $|v^{i+1}|$ tokens $v^{i+1} = (v_1^{i+1}, \ldots, v_{|v^{i+1}|}^{i+1})$, where token v_j^{i+1} is either a question token $\in \mathcal{V}_q$ (or some inflection of it), a word from a constant predefined lexicon $\in \mathcal{V}_{const}$, or a reference token $\in \mathcal{V}_{ref}^{i+1} = \{\#1, \ldots, \#i\}$ referring to a previous step. Among them, the reference token $\#j$ also belongs to the edge tokens e^{ij}. Therefore, we decompose the prediction of the reference tokens into two stages. In the node prediction stage, we predict #R indicating a placeholder for the edge token. Then, in the edge prediction stage, we predict the exact number of the reference token to replace R and get $\#j$. The probability over vocabulary $\mathcal{V} = \mathcal{V}_q \bigcup \mathcal{V}_{const} \bigcup \{\#\text{R}\}$ can be obtained by:

$$P(v_z^{i+1}|v_{<z}^{i+1}, q, G^i, \theta) = \text{softmax}(W^P h_{\text{final}} + b^P) \quad (4)$$

where $W^P \in \mathbb{R}^{d_k \times |\mathcal{V}|}, b^P \in \mathbb{R}^{|\mathcal{V}|}$ are learnable parameters that transform the final representation into the probability over $\mathcal{V}$. The model is parameterized by θ. $v_{<z}^{i+1}$ denotes the partial action $\{v_0^{i+1}, \ldots, v_{j-1}^{i+1}\}$.

2.4.2 Edge Prediction

Instead of treating the reference tokens as static tokens in the vocabulary as the previous works, which shares the embeddings among different examples and thus ignores their semantics. We obtain their representation dynamically according to the constructed partial graph. Specifically, we average the representation of $v_1^i \ldots v_{|v^i|}^i$ to obtain the representation of $\#i$ which is denoted as $h_{\#i}$. Then, we adopt a bilinear function to compute the similarity between each reference representation $h_{\#i}$ and the final representation h_{final}. The probability over $\mathcal{V}_{ref}^{i+1}$ can be obtained by:

$$P(v_z^{i+1}|v_{<z}^{i+1}, q, G^i, \theta) = \text{softmax}(E^{\text{ref}} W^{\text{ref}} h_{\text{final}})$$
$$j = \text{argmax}_{j \in \mathcal{V}_{ref}^{i+1}} P(j|v_{<z}^{i+1}, q, G^i, \theta) \quad (5)$$

where $E^{\text{ref}} \in \mathbb{R}^{i \times d_h}$ denotes the embedding matrix of the reference tokens. $W^{\text{ref}} \in \mathbb{R}^{d_h \times d_h}$ is a learnable matrix. We use the predicted reference number j to replace R for v_z^{i+1} if it is a #R.

2.4.3 Action Confidence

We apply beam search with beam size K_1 sampling on $P(v_z^{i+1}|v_{<z}^{i+1}, q, G^i, \theta)$ to get top K_1 candidate actions. The confidence of each candidate action a^{i+1} is defined as the probability of the predicted sequence i.e. the product of the probabilities of the predicted tokens $v_1^{i+1} \ldots v_{|v^{i+1}|}^{i+1}$:

$$P(a^{i+1}|q, G^i, \theta) = \prod_{z=1}^{|v^{i+1}|} P(v_z^{i+1}|v_{<z}^{i+1}, q, G^i, \theta) \quad (6)$$

2.5 Training

We train our transition-based model with the standard maximum likelihood estimate using teacher forcing. In other words, we maximize the sum of the stepwise action confidence. The loss *w.r.t* an example is defined as follows:

$$P(G|q, \theta) = \prod_{i=1}^{|G|} P(a^{i+1}|q, G^i, \theta)$$
$$\mathcal{L}(G|q, \theta) = -\frac{1}{m} \sum_{i=1}^{m} logP(a^{i+1}|q, G^i, \theta) \quad (7)$$

where $G = (a^1, \ldots, a^m) = ([\text{A}_1], v_1^1, \ldots, v_{|v^1|}^1, [\text{A}_2], \ldots, [\text{A}_m], \ldots, v_{|v^m|}^m, [\text{END}])$ and m denotes the number of the action in G.

2.6 Inference

The whole inference procedure is shown in Algorithm 1. At line $1 \sim 3$, we first obtain the question representation and initialize the output with an empty graph. In the loop from line 5, we predict an action at each transition step until the [END] is generated. At each transition step, we first initialize the predicted action v_0^{i+1} with $[A_i]$. Then, in the loop from line 9, we generate a token at each step until the $[\mathrm{A}_{i+1}]$ or [END] indicating the termination of an action is predicted. At line $10 \sim 15$, we obtain the partial graph representation and the final representation. At line $18 \sim 21$, we first predict v_z^{i+1} by sampling on Eq. 4. Then, we decide the exact number by Eq. 5 if v_z^{i+1} is a #R token. At line 24, we obtain topK_2 a^{i+1} by sampling on Eq. 7. Finally, we add the action predictions to G^i and get the new graph G^{i+1}.

Note that in the above inference procedure, we adopt a *searcher* which seeks for high-probability output graphs to relieve the error propagation and guide the direction of the graph expansion. Inspired by the traditional beam search which decodes a single token at each search step, we replace the original generation probability with the action confidence defined in Eq. 6. At step i, we maintain K_2 candidate actions sampling on the probability $P(a^{i+1}|q, G^i, \theta)$ where K_2 denotes the beam size.

Algorithm 1 Inference procedure of our framework

Require: a question with n tokens $q = (q_1, q_2, ..., q_n)$.
Ensure: topK_2 computational graphs.

1: $H^q \leftarrow$Question-Encoder(q);
2: $G^0 \leftarrow (V^0, E^0), V^0 \leftarrow \emptyset, E^0 \leftarrow \emptyset$;
3: $i \leftarrow 1$;
4: // Generating topK_2 Action Sequence
5: **while** the last action a^{i+1} is not [END] **do**
6: $\quad v_0^{i+1} \leftarrow [A_i]$;
7: $\quad z \leftarrow 1$;
8: $\quad$ // Generating topK_1 Action Candidates
9: $\quad$ **while** the last token v_{z-1}^{i+1} is not in $\{[\mathrm{A}_{i+1}], [\mathrm{END}]\}$ **do**
10: $\quad\quad G^i \leftarrow G^i + \{v^{i+1}\}$;
11: $\quad\quad$ // Partial Graph Encoding
12: $\quad\quad H^{G^i} \leftarrow$Graph-Encoder$(G^i, q)$;
13: $\quad\quad$ // Interaction
14: $\quad\quad H^{q^i} \leftarrow$Attention$(H^q, H^{G^i})H^q$;
15: $\quad\quad \hat{H}^{G^i} \leftarrow$Attention$(H^{G^i}, H^q)H^{Gi}$;
16: $\quad\quad h_{\text{final}} \leftarrow \mathrm{MaxPooling}(H^{q^i}, \hat{H}^{G^i})$;
17: $\quad\quad$ // Node & Edge Prediction
18: $\quad\quad$ get topK_1 v_z^{i+1} by a search step on Eq. 4;
19: $\quad\quad$ **if** v_z^{i+1}=#R **then**
20: $\quad\quad\quad$ get the number of #R by Eq. 5.
21: $\quad\quad$ **end if**
22: $\quad\quad z \leftarrow z + 1$;
23: $\quad$ **end while**
24: $\quad$ get topK_2 a^{i+1} by a search step on Eq. 7;
25: $\quad G^{i+1} \leftarrow G^i + \{a^{i+1}\}$;
26: $\quad i \leftarrow i + 1$;
27: **end while**

3 Experiment

3.1 Experimental Setup

Datasets and Metrics Our evaluation is conducted on the dataset BREAK. The question-QDMR pairs are crowd-sourced based on questions sampled from ten widely-used QA datasets. It consists of 83,978 examples including 60,150 examples with QDMR and 23,828 examples with high-level QDMR. We do not include examples with high-level QDMR for the sake of a fair comparison with the previous work.The QA datasets included in BREAK and the statistics are listed in Table 1. The train/dev/test sets are partitioned following the original datasets. The distribution over QDMR sequence length is listed in Table 4. Note that the gold answers of the test set are not publicly available, so we report performance on the development set.

Metrics Following the previous work (Hasson and Berant, 2021), we use Normalized Exact Match (NormEM) and Logical Form Exact Match (LF-EM) as the evaluation metrics. **Normalized Exact Match (NormEM)**: The predicted and gold QDMRs are first normalized by a rule-based procedure, and then exact string match is computed between the two normalized QDMRs. The value for each sample is either 0 or 1. **Logical Form Exact Match (LF-EM)**: The predicted and gold QDMRs are first converted to the logical form by a rule-based procedure, and then exact string match is computed between the two logical form QDMRs. The value for each sample is either 0 or 1.

Implementation Details We follow the previous work (Hasson and Berant, 2021) for implementation. We use BART-base (Lewis et al., 2019) as our backbone. The models are implemented in Pytorch (Paszke et al., 2019) and are trained on a single

	BREAK									
	ACADEMIC	ATIS	CLEVR	COMQA	GEO	CWQ	DROP	NLVR2	SPIDER	Total
Train	195	4042	9453	3546	547	1985	7683	9915	6955	44321
Dev	-	457	2215	988	50	475	1268	1805	502	7760
Test	-	407	2267	986	280	528	1279	1797	525	8069

Table 1: The question distribution of each QD dataset in BREAK.

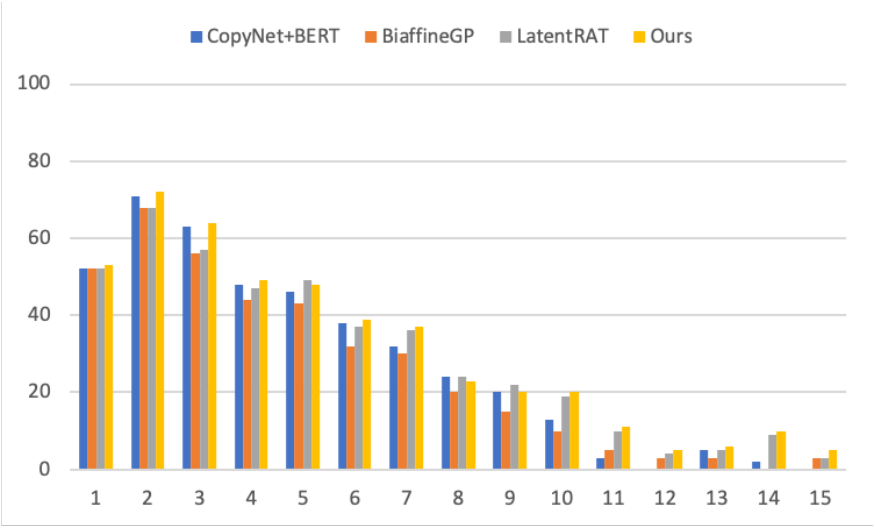

Figure 3: The average LF-EM of our method and baselines for different lengths of QDMR on the dev set.

Model	NormEM	LF-EM
CopyNet+BERT	37.3	47.4
BiaffineGP	-	45.3
Latent-RAT	35.6	46.9
BART	38.1	47.7
Ours w/o Beam Search	38.9	48.1
Ours w/ Beam Search	**41.2**	**49.7**

Table 2: NormEM and LF-EM on the dev set.

GeForce RTX 3090 GPU. We set the batch size as 32, and the max training epoch number as 20 with early stopping (patience=5). We utilize Adam optimizer (Kingma and Ba, 2014) with a dynamic learning rate according to the slanted triangular schema. The beam size of both inner and outer beam search is set to 5.

Baselines We compare our framework with various previous works in terms of parsing accuracy and interpretability.

CopyNet+BERT is a seq2seq model consisting of a BERT encoder and an LSTM decoder with a copy mechanism. **BiaffineGP** is based on the biaffine dependency parser of (Dozat and Manning, 2018) except that it predicts a DAG and not a tree. Besides, it applies an Integer Linear Programming layer on top of it to eliminate constraint violations in the output graph. **Latent-RAT** is based on RAT transformer layers (Shaw et al., 2018; Wang et al., 2019) to predict the graph structure using the encoder and predict the QDMR sequence using the decoder. **BART** is based on the pretrained seq2seq model BART.

3.2 Results

3.2.1 Main Results

Table 2 shows the overall performance of our method and all the baselines on the development set of BREAK. Our method achieves the best results among the recently available methods. Specifically, our method without beam search achieves comparable performance to BART with advantages of 0.8 NormEM and 0.4 LF-EM and outperforms CopyNet+BERT by 1.6 NormEM and 0.7 LF-EM. We attribute the performance gain to the better modeling of the question and the current state. Enhanced by beam search, our method exceeds BART by 3.1 NormEM and 2.0 LF-EM which demonstrates the potential of increasing the search space.

Question	Wrong Prediction	Ours

How many was the difference between Sobieski's force and the Turks and Tatars?	Latent-RAT:
	1.select(sub=Sobieski)
	2.project(projection=force of #REF;sub=#1)
	3.select(sub=Turks)
	4.select(sub=the Tatars)
	5.arithmetic[difference](left=#2;right=#3)
	6.arithmetic[difference](left=#4;right=#5)
		1.select(sub=Sobieski)
		2.project(projection=force of #REF;sub=#1)
		3.project(projection=size of #REF; sub=#2)
		4.select(sub=the Turks and Tatars)
		5.project(projection=the force of #REF; sub=#4)
		6.project(projection=size of #REF;sub=#5)
		5.arithmetic[difference](left=#6; right=#3)

How many year after Knopf was founded was it officially incorporated?	BiaffineGP:
	1.project(projection=Knopf was founded years; sub=#1)
	2.select(sub=was it officially incorporated)
	3.project(projection=years;sub=#2);
	4.arithmetic[difference](left=#3;right=#1)
		1.select(sub=Knopf was founded)
		2.select(sub=Knopf was officially incorporated)
		3.project(projection=year of #REF; sub=#1)
		4.project(projection=year of #REF; sub=#2)
		5.arithmetic[difference](left=#4; right=#3)

Table 3: Two cases from the dev set. The outputs are converted to Logical Form for comparison with BiaffineGP. One can refer to (Hasson and Berant, 2021) for the conversion details.

Length	1-2	3-4	5-6	7-8	9+
Percentage(%)	10.7	44.9	27.0	10.1	7.4

Table 4: The distribution over the length of QDMR actions.

	Token	Action
Train	11.59	4.75
Dev	11.35	4.90

Table 5: The average length of token sequences and action sequences in QDMR.

3.2.2 Length Analysis

In order to explore how much does our transition-based framework contribute to examples with longer steps, we plot and compare the average LF-EM of different methods for each possible number of steps in QDMR. From Figure 3 we can see that, as the number of steps increases, our method exceeds the baselines greater. It verifies that our method handles complex decompositions better.

Table 5 shows the average length of token sequences used in seq2seq models and of action sequences used in our method. As shown in the table, the action sequence is much shorter than the token sequence i.e. reduced from 11 to 5 in length. In other words, representing QDMR as an action sequence has the advantage of more compact encoding which makes the modeling of long-distance dependency easier. Therefore, it is more appropriate for QDMR generation.

3.2.3 Interpretability Analysis

In order to verify the interpretability of our method, we print the beam search process together with the log probabilities of different action sequences. From Figure 4 we can see that, the sequence with the highest log probability (-2.44) matches the gold decomposition. We note that although the last three sequences do not match with the gold decomposition, they are logically equivalent to the provided gold one which can also get the correct answer to the question.

3.2.4 Case Study

We show two examples from the development set to illustrate the effectiveness of our model by comparing the results of different models in Table 3. The first example shows that beam search helps for searching the optimal graph. Latent-RAT fails to predict the correct structure and starts to deviate from step 3. In contrast, our method seeks the optimal graph in a larger search space and predicts the correct structure. The second example shows that our modeling of the question and the current state helps the model decide a step more accurately. In the example, BiaffineGP predicts the whole graph in an end-to-end manner. Our method predicts more accurately with the help of the better representation.

4 Related Work

Complex Question Understanding Complex question understanding is proposed by (Wolfson et al., 2020) as a standalone language understanding task. They introduce a formalism named QDMR to represent the meaning of questions that relies on question decomposition and is agnostic to

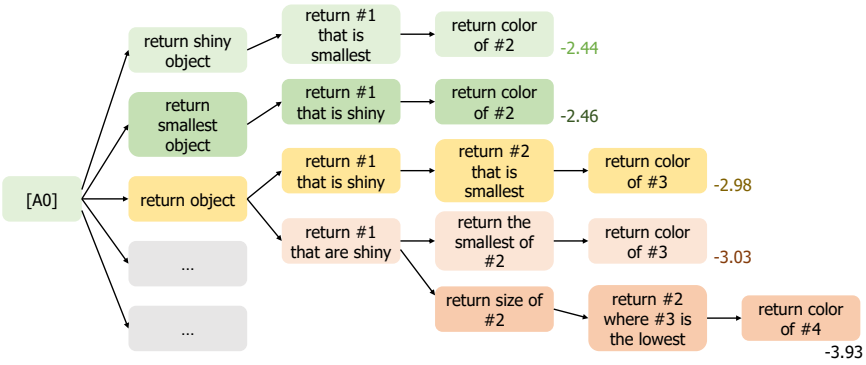

Figure 4: The beam search process of our method corresponds to the question "What color is the smallest, shiny object?". For simplicity, we do not draw the unselected nodes. The log probability of the final graph is listed at the end of the sequence.

the information modality. Existing approaches can be divided into two categories: the seq2seq-based method and the dependency-based method. The seq2seq-based method (Hasson and Berant, 2021) treats QDMR as a sentence and adopts a seq2seq model to decode the textual QDMR one token at each step. This token-wise modeling seems sub-optimal for generating QDMR which ignores its inherent structure. The dependency-based method (Hasson and Berant, 2021) maps QDMR to a dependency graph over question tokens and adopts a non-autoregressive graph parser to predict the entire graph in a single step. It predicts all steps of QDMR simultaneously which does not model the interaction between different predictions, trading off performance for efficiency. There is also work combining the two categories by exploiting the graph supervision to train a seq2seq model. It adds an auxiliary loss term where the graph is decoded from the encoder representations. However, the above methods do not explicitly model the step-wise information which is an essential and distinct characteristic of QDMR. Before CQU was proposed, some work has explored decomposing questions to facilitate answering complex questions that require discrete reasoning (Talmor and Berant, 2018). IBM Watson (Ferrucci et al., 2010) decomposes questions into sub-questions in multiple ways or not at all. DECOMPRC (Min et al., 2019) recasts sub-question generation as a span prediction problem which requires only 400 decomposition examples to train a competitive model. (Iyyer et al., 2017; Talmor and Berant, 2018) have also decomposed questions to create a sequential question answering task. Despite the initial success, their decomposition methods remain preliminary and they conduct experiments on a much more limited set of questions than in BREAK.

Semantic Parsing Semantic parsing is a larger area of work that aims at parsing natural language utterances into logical forms(Zelle and Mooney, 1996; Zettlemoyer and Collins, 2012; Liang et al., 2013). They are usually executed over structured knowledge bases such as relational databases(Yu et al., 2018) and graph KBs(Yih et al., 2016). Our work is inspired by the idea of transition-based systems from semantic parsing(Chen et al., 2018). CQU differs from semantic parsing in that it produces meaning representation expressed in natural language which is easy to annotate at scale and can be potentially converted to other meaning representations based on the task at hand. Besides, CQU focuses on representing the semantics of complex questions which is important for QA systems and for probing models for reasoning.

5 Conclusion

In this paper, to model the intermediate states and the step/operator-wise semantic, we view QDMR as a computational graph and propose a transition-based method where a *decider* wrapped with a *searcher* incrementally constructs the graph. Experimental results show that our framework outperforms the state-of-the-art CQU model by 3.1 NormEM and 2.0 LF-EM. Further visualization also demonstrates the interpretability of our method by giving transparent and human-readable intermediate results.

Acknowledgements

This work is partially supported by Na- tional Key Research and Development Project (2019YFB1704002) and National Natural Science Foundation of China (61876009). This work is also supported by Baidu-PKU Joint Project.

References

Bo Chen, Le Sun, and Xianpei Han. 2018. Sequence-to-action: End-to-end semantic graph generation for semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 766–777, Melbourne, Australia. Association for Computational Linguistics.

Timothy Dozat and Christopher D Manning. 2018. Simpler but more accurate semantic dependency parsing. *arXiv preprint arXiv:1807.01396*.

David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. 2010. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79.

Matan Hasson and Jonathan Berant. 2021. Question decomposition with dependency graphs. *arXiv preprint arXiv:2104.08647*.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1821–1831.

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

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Percy Liang, Michael I Jordan, and Dan Klein. 2013. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446.

Sewon Min, Victor Zhong, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2019. Multi-hop reading comprehension through question decomposition and rescoring. *arXiv preprint arXiv:1906.02916*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*.

Sanjay Subramanian, Ben Bogin, Nitish Gupta, Tomer Wolfson, Sameer Singh, Jonathan Berant, and Matt Gardner. 2020. Obtaining faithful interpretations from compositional neural networks. *arXiv preprint arXiv:2005.00724*.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. *arXiv preprint arXiv:1803.06643*.

Alon Talmor, Ori Yoran, Amnon Catav, Dan Lahav, Yizhong Wang, Akari Asai, Gabriel Ilharco, Hannaneh Hajishirzi, and Jonathan Berant. 2021. Multimodalqa: Complex question answering over text, tables and images. *arXiv preprint arXiv:2104.06039*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*.

Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. 2020. Break it down: A question understanding benchmark. *Transactions of the Association for Computational Linguistics*, 8:183–198.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 201–206.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

Luke S Zettlemoyer and Michael Collins. 2012. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420*.