Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base

Wen-tau Yih Ming-Wei Chang Xiaodong He Jianfeng Gao
Microsoft Research
Redmond, WA 98052, USA
{scottyih,minchang,xiaohe,jfgao}@microsoft.com

Abstract

We propose a novel semantic parsing framework for question answering using a knowledge base. We define a *query graph* that resembles subgraphs of the knowledge base and can be directly mapped to a logical form. Semantic parsing is reduced to query graph generation, formulated as a staged search problem. Unlike traditional approaches, our method leverages the knowledge base in an early stage to prune the search space and thus simplifies the semantic matching problem. By applying an advanced entity linking system and a deep convolutional neural network model that matches questions and predicate sequences, our system outperforms previous methods substantially, and achieves an F_1 measure of 52.5% on the WEBQUESTIONS dataset.

1 Introduction

Organizing the world's facts and storing them in a structured database, large-scale knowledge bases (KB) like DBPedia (Auer et al., 2007) and Freebase (Bollacker et al., 2008) have become important resources for supporting open-domain question answering (QA). Most state-of-the-art approaches to KB-QA are based on semantic parsing, where a question (utterance) is mapped to its formal meaning representation (e.g., logical form) and then translated to a KB query. The answers to the question can then be retrieved simply by executing the query. The semantic parse also provides a deeper understanding of the question, which can be used to justify the answer to users, as well as to provide easily interpretable information to developers for error analysis.

However, most traditional approaches for semantic parsing are largely decoupled from the knowledge base, and thus are faced with several challenges when adapted to applications like QA. For instance, a generic meaning representation may have the ontology matching problem when the logical form uses predicates that differ from those defined in the KB (Kwiatkowski et al., 2013). Even when the representation language is closely related to the knowledge base schema, finding the correct predicates from the large vocabulary in the KB to relations described in the utterance remains a difficult problem (Berant and Liang, 2014).

Inspired by (Yao and Van Durme, 2014; Bao et al., 2014), we propose a semantic parsing framework that leverages the knowledge base more tightly when forming the parse for an input question. We first define a *query graph* that can be straightforwardly mapped to a logical form in λ-calculus and is semantically closely related to λ-DCS (Liang, 2013). Semantic parsing is then reduced to query graph generation, formulated as a search problem with staged states and actions. Each state is a candidate parse in the query graph representation and each action defines a way to *grow* the graph. The representation power of the semantic parse is thus controlled by the set of legitimate actions applicable to each state. In particular, we stage the actions into three main steps: locating the *topic entity* in the question, finding the main *relationship* between the answer and the topic entity, and expanding the query graph with additional *constraints* that describe properties the answer needs to have, or relationships between the answer and other entities in the question.

One key advantage of this staged design is that through grounding partially the utterance to some entities and predicates in the KB, we make the search far more efficient by focusing on the promising areas in the space that most likely lead to the correct query graph, before the full parse is determined. For example, after linking "Fam-

ily Guy" in the question "Who first voiced Meg on Family Guy?" to `FamilyGuy` (the TV show) in the knowledge base, the procedure needs only to examine the predicates that can be applied to `FamilyGuy` instead of all the predicates in the KB. Resolving other entities also becomes easy, as given the context, it is clear that Meg refers to `MegGriffin` (the character in Family Guy). Our design divides this particular semantic parsing problem into several sub-problems, such as entity linking and relation matching. With this integrated framework, best solutions to each subproblem can be easily combined and help produce the correct semantic parse. For instance, an advanced entity linking system that we employ outputs candidate entities for each question with both high precision and recall. In addition, by leveraging a recently developed semantic matching framework based on convolutional networks, we present better relation matching models using continuous-space representations instead of pure lexical matching. Our semantic parsing approach improves the state-of-the-art result on the WEBQUESTIONS dataset (Berant et al., 2013) to 52.5% in F_1, a 7.2% absolute gain compared to the best existing method.

The rest of this paper is structured as follows. Sec. 2 introduces the basic notion of the graph knowledge base and the design of our query graph. Sec. 3 presents our search-based approach for generating the query graph. The experimental results are shown in Sec. 4, and the discussion of our approach and the comparisons to related work are in Sec. 5. Finally, Sec. 6 concludes the paper.

2 Background

In this work, we aim to learn a semantic parser that maps a natural language question to a logical form query q, which can be executed against a knowledge base $\mathcal{K}$ to retrieve the answers. Our approach takes a graphical view of both $\mathcal{K}$ and q, and reduces semantic parsing to mapping questions to *query graphs*. We describe the basic design below.

2.1 Knowledge base

The knowledge base $\mathcal{K}$ considered in this work is a collection of subject-predicate-object triples (e_1, p, e_2), where $e_1, e_2 \in \mathcal{E}$ are the entities (e.g., `FamilyGuy` or `MegGriffin`) and $p \in \mathcal{P}$ is a binary predicate like `character`. A knowledge base in this form is often called a *knowledge graph* because of its straightforward graphical representation – each entity is a node and two related entities are linked by a directed edge labeled by the predicate, from the subject to the object entity.

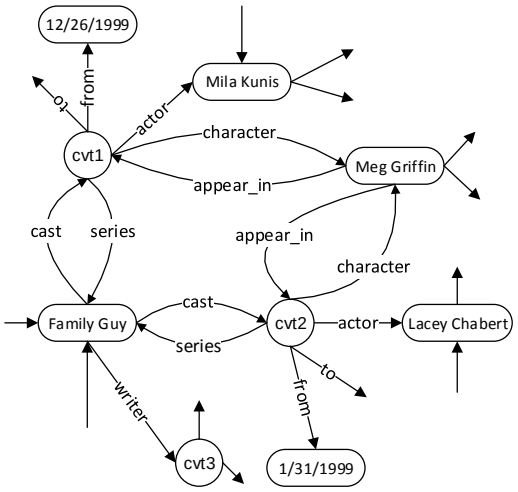


Figure 1: Freebase subgraph of Family Guy

To compare our approach to existing methods, we use Freebase, which is a large database with more than 46 million topics and 2.6 billion facts. In Freebase's design, there is a special entity category called *compound value type* (CVT), which is not a real-world entity, but is used to collect multiple fields of an event or a special relationship.

Fig. 1 shows a small subgraph of Freebase related to the TV show *Family Guy*. Nodes are the entities, including some dates and special CVT entities[1]. A directed edge describes the relation between two entities, labeled by the predicate.

2.2 Query graph

Given the knowledge graph, executing a logical-form query is equivalent to finding a subgraph that can be mapped to the query and then resolving the binding of the variables. To capture this intuition, we describe a restricted subset of λ-calculus in a graph representation as our *query graph*.

Our query graph consists of four types of nodes: *grounded entity* (rounded rectangle), *existential variable* (circle), *lambda variable* (shaded circle), *aggregation function* (diamond). Grounded entities are existing entities in the knowledge base $\mathcal{K}$. Existential variables and lambda variables are un-

[1] In the rest of the paper, we use the term *entity* for both real-world and CVT entities, as well as properties like date or height. The distinction is not essential to our approach.

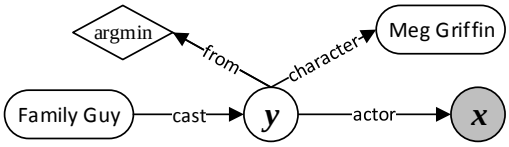


Figure 2: Query graph that represents the question "Who first voiced Meg on Family Guy?"

grounded entities. In particular, we would like to retrieve all the entities that can map to the lambda variables in the end as the answers. Aggregation function is designed to operate on a specific entity, which typically captures some numerical properties. Just like in the knowledge graph, related nodes in the query graph are connected by directed edges, labeled with predicates in $\mathcal{K}$.

To demonstrate this design, Fig. 2 shows one possible query graph for the question "Who first voiced Meg on Family Guy?" using Freebase. The two entities, `MegGriffin` and `FamilyGuy` are represented by two rounded rectangle nodes. The circle node y means that there should exist an entity describing some casting relations like the character, actor and the time she started the role[2]. The shaded circle node x is also called the answer node, and is used to map entities retrieved by the query. The diamond node $\arg\min$ constrains that the answer needs to be the earliest actor for this role. Equivalently, the logical form query in λ-calculus without the aggregation function is: $\lambda x.\exists y.\texttt{cast}(\texttt{FamilyGuy}, y) \wedge \texttt{actor}(y, x) \wedge \texttt{character}(y, \texttt{MegGriffin})$ Running this query graph against $\mathcal{K}$ as in Fig. 1 will match both `LaceyChabert` and `MilaKunis` before applying the aggregation function, but only `LaceyChabert` is the correct answer as she started this role earlier (by checking the `from` property of the grounded CVT node).

Our query graph design is inspired by (Reddy et al., 2014), but with some key differences. The nodes and edges in our query graph closely resemble the exact entities and predicates from the knowledge base. As a result, the graph can be straightforwardly translated to a logical form query that is directly executable. In contrast, the query graph in (Reddy et al., 2014) is mapped from the CCG parse of the question, and needs further transformations before mapping to subgraphs

[2] y should be grounded to a CVT entity in this case.

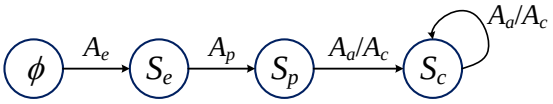


Figure 3: The legitimate actions to *grow* a query graph. See text for detail.

of the target knowledge base to retrieve answers.

Semantically, our query graph is more related to simple λ-DCS (Berant et al., 2013; Liang, 2013), which is a syntactic simplification of λ-calculus when applied to graph databases. A query graph can be viewed as the tree-like graph pattern of a logical form in λ-DCS. For instance, the path from the answer node to an entity node can be described using a series of *join* operations in λ-DCS. Different paths of the tree graph are combined via the *intersection* operators.

3 Staged Query Graph Generation

We focus on generating query graphs with the following properties. First, the tree graph consists of one entity node as the root, referred as the *topic entity*. Second, there exists only one lambda variable x as the answer node, with a directed path from the root to it, and has zero or more existential variables in-between. We call this path the *core inferential chain* of the graph, as it describes the main relationship between the answer and topic entity. Variables can only occur in this chain, and the chain only has variable nodes except the root. Finally, zero or more entity or aggregation nodes can be attached to each variable node, including the answer node. These branches are the additional *constraints* that the answers need to satisfy. For example, in Fig. 2, `FamilyGuy` is the root and $\texttt{FamilyGuy} \rightarrow y \rightarrow x$ is the core inferential chain. The branch $y \rightarrow \texttt{MegGriffin}$ specifies the character and $y \rightarrow \arg\min$ constrains that the answer needs to be the earliest actor for this role.

Given a question, we formalize the query graph generation process as a search problem, with *staged* states and actions. Let $\mathcal{S} = \bigcup\{\phi, \mathcal{S}_e, \mathcal{S}_p, \mathcal{S}_c\}$ be the set of states, where each state could be an empty graph (ϕ), a single-node graph with the topic entity ($\mathcal{S}_e$), a core inferential chain ($\mathcal{S}_p$), or a more complex query graph with additional constraints ($\mathcal{S}_c$). Let $\mathcal{A} = \bigcup\{\mathcal{A}_e, \mathcal{A}_p, \mathcal{A}_c, \mathcal{A}_a\}$ be the set of actions. An action grows a given graph by adding some edges

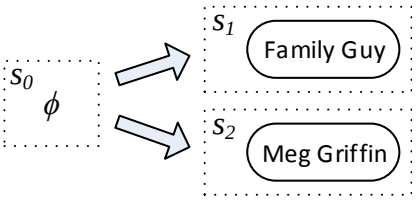


Figure 4: Two possible topic entity linking actions applied to an empty graph, for question "Who first voiced [Meg] on [Family Guy]?"

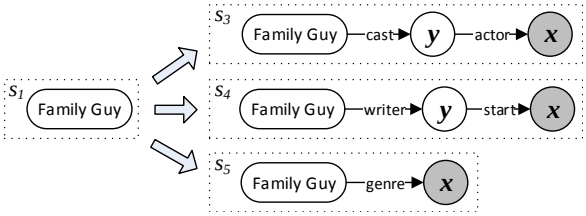


Figure 5: Candidate core inferential chains start from the entity `FamilyGuy`.

and nodes. In particular, $\mathcal{A}_e$ picks an entity node; $\mathcal{A}_p$ determines the core inferential chain; $\mathcal{A}_c$ and $\mathcal{A}_a$ add constraints and aggregation nodes, respectively. Given a state, the valid action set can be defined by the finite state diagram in Fig. 3. Notice that the order of possible actions is chosen for the convenience of implementation. In principle, we could choose a different order, such as matching the core inferential chain first and then resolving the topic entity linking. However, since we will consider multiple hypotheses during search, the order of the staged actions can simply be viewed as a different way to prune the search space or to bias the exploration order.

We define the reward function on the state space using a log-linear model. The reward basically estimates the likelihood that a query graph correctly parses the question. Search is done using the best-first strategy with a priority queue, which is formally defined in Appendix A. In the following subsections, we use a running example of finding the semantic parse of question q_{ex} = "Who first voiced Meg of Family Guy?" to describe the sequence of actions.

3.1 Linking Topic Entity

Starting from the initial state s_0, the valid actions are to create a single-node graph that corresponds to the topic entity found in the given question. For instance, possible topic entities in q_{ex} can either be `FamilyGuy` or `MegGriffin`, shown in Fig. 4.

We use an entity linking system that is designed for short and noisy text (Yang and Chang, 2015). For each entity e in the knowledge base, the system first prepares a surface-form lexicon that lists all possible ways that e can be mentioned in text. This lexicon is created using various data sources, such as names and aliases of the entities, the anchor text in Web documents and the Wikipedia redirect table. Given a question, it considers all the consecutive word sequences that have occurred in the lexicon as possible mentions, paired with their possible entities. Each pair is then scored by a statistical model based on its frequency counts in the surface-form lexicon. To tolerate potential mistakes of the entity linking system, as well as exploring more possible query graphs, up to 10 top-ranked entities are considered as the topic entity. The linking score will also be used as a feature for the reward function.

3.2 Identifying Core Inferential Chain

Given a state s that corresponds to a single-node graph with the topic entity e, valid actions to extend this graph is to identify the *core inferential chain*; namely, the relationship between the topic entity and the answer. For example, Fig. 5 shows three possible chains that expand the single-node graph in s_1. Because the topic entity e is given, we only need to explore legitimate predicate sequences that can start from e. Specifically, to restrict the search space, we explore all paths of length 2 when the middle existential variable can be grounded to a CVT node and paths of length 1 if not. We also consider longer predicate sequences if the combinations are observed in training data[3].

Analogous to the entity linking problem, where the goal is to find the mapping of mentions to entities in $\mathcal{K}$, identifying the core inferential chain is to map the natural utterance of the question to the correct predicate sequence. For question "*Who* first voiced Meg on [Family Guy]?" we need to measure the likelihood that each of the sequences in {`cast-actor`, `writer-start`, `genre`} correctly captures the relationship between *Family Guy* and *Who*. We reduce this problem to measuring semantic similarity using neural networks.

[3]Decomposing relations in the utterance can be done using decoding methods (e.g., (Bao et al., 2014)). However, similar to ontology mismatch, the relation in text may not have a corresponding single predicate, such as *grandparent* needs to be mapped to `parent-parent` in Freebase.

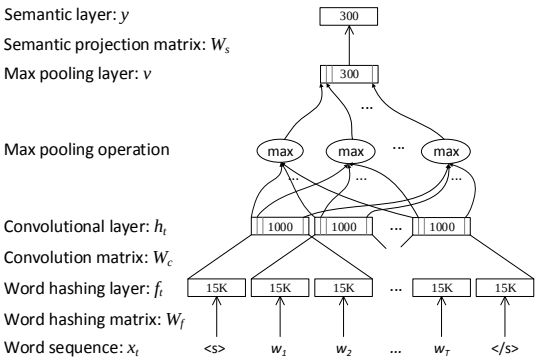


Figure 6: The architecture of the convolutional neural networks (CNN) used in this work. The CNN model maps a variable-length word sequence (e.g., a pattern or predicate sequence) to a low-dimensional vector in a latent semantic space. See text for the description of each layer.

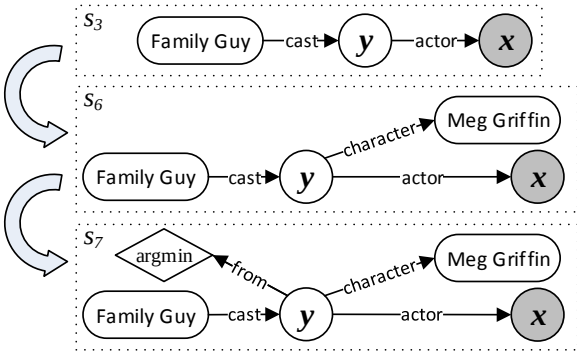


Figure 7: Extending an inferential chain with constraints and aggregation functions.

3.2.1 Deep Convolutional Neural Networks

To handle the huge variety of the semantically equivalent ways of stating the same question, as well as the mismatch of the natural language utterances and predicates in the knowledge base, we propose using Siamese neural networks (Bromley et al., 1993) for identifying the core inferential chain. For instance, one of our constructions maps the question to a *pattern* by replacing the entity mention with a generic symbol <e> and then compares it with a candidate chain, such as "who first voiced meg on <e>" vs. `cast-actor`. The model consists of two neural networks, one for the pattern and the other for the inferential chain. Both are mapped to k-dimensional vectors as the output of the networks. Their semantic similarity is then computed using some distance function, such as *cosine*. This continuous-space representation approach has been proposed recently for semantic parsing and question answering (Bordes et al., 2014a; Yih et al., 2014) and has shown better results compared to lexical matching approaches (e.g., word-alignment models). In this work, we adapt a convolutional neural network (CNN) framework (Shen et al., 2014b; Shen et al., 2014a; Gao et al., 2014) to this matching problem. The network architecture is illustrated in Fig. 6.

The CNN model first applies a word hashing technique (Huang et al., 2013) that breaks a word into a vector of letter-trigrams ($x_t \rightarrow f_t$ in Fig. 6). For example, the bag of letter-trigrams of the word "who" are #-w-h, w-h-o, h-o-# after adding the word boundary symbol #. Then, it uses a convolutional layer to project the letter-trigram vectors of words within a context window of 3 words to a local contextual feature vector ($f_t \rightarrow h_t$), followed by a max pooling layer that extracts the most salient local features to form a fixed-length global feature vector (v). The global feature vector is then fed to feed-forward neural network layers to output the final non-linear semantic features (y), as the vector representation of either the pattern or the inferential chain.

Training the model needs positive pairs, such as a pattern like "who first voiced meg on <e>" and an inferential chain like `cast-actor`. These pairs can be extracted from the full semantic parses when provided in the training data. If the correct semantic parses are latent and only the pairs of questions and answers are available, such as the case in the WEBQUESTIONS dataset, we can still hypothesize possible inferential chains by traversing the paths in the knowledge base that connect the topic entity and the answer. Sec. 4.1 will illustrate this data generation process in detail.

Our model has two advantages over the embedding approach (Bordes et al., 2014a). First, the word hashing layer helps control the dimensionality of the input space and can easily scale to large vocabulary. The letter-trigrams also capture some sub-word semantics (e.g., words with minor typos have almost identical letter-trigram vectors), which makes it especially suitable for questions from real-world users, such as those issued to a search engine. Second, it uses a deeper architecture with convolution and max-pooling layers, which has more representation power.

3.3 Augmenting Constraints & Aggregations

A graph with just the inferential chain forms the simplest legitimate query graph and can be executed against the knowledge base $\mathcal{K}$ to retrieve the answers; namely, all the entities that x can be grounded to. For instance, the graph in s_3 in Fig. 7 will retrieve all the actors who have been on `FamilyGuy`. Although this set of entities obviously contains the correct answer to the question (assuming the topic entity FamilyGuy is correct), it also includes incorrect entities that do not satisfy additional constraints implicitly or explicitly mentioned in the question.

To further restrict the set of answer entities, the graph with only the core inferential chain can be expanded by two types of actions: $\mathcal{A}_c$ and $\mathcal{A}_a$. A_c is the set of possible ways to attach an entity to a variable node, where the edge denotes one of the valid predicates that can link the variable to the entity. For instance, in Fig. 7, s_6 is created by attaching `MegGriffin` to y with the predicate `character`. This is equivalent to the last conjunctive term in the corresponding λ-expression: $\lambda x.\exists y.\texttt{cast}(\texttt{FamilyGuy}, y) \wedge \texttt{actor}(y, x) \wedge \texttt{character}(y, \texttt{MegGriffin})$. Sometimes, the constraints are described over the entire answer set through the aggregation function, such as the word "first" in our example question q_{ex}. This is handled similarly by actions $\mathcal{A}_a$, which attach an aggregation node on a variable node. For example, the $\arg\min$ node of s_7 in Fig. 7 chooses the grounding with the smallest `from` attribute of y.

The full possible constraint set can be derived by first issuing the core inferential chain as a query to the knowledge base to find the bindings of variables y's and x, and then enumerating all neighboring nodes of these entities. This, however, often results in an unnecessarily large constraint pool. In this work, we employ simple rules to retain only the nodes that have some possibility to be legitimate constraints. For instance, a constraint node can be an entity that also appears in the question (detected by our entity linking component), or an aggregation constraint can only be added if certain keywords like "first" or "latest" occur in the question. The complete set of these rules can be found in Appendix B.

3.4 Learning the reward function

Given a state s, the reward function $\gamma(s)$ basically judges whether the query graph represented by s is the correct semantic parse of the input question q. We use a log-linear model to learn the reward function. Below we describe the features and the learning process.

3.4.1 Features

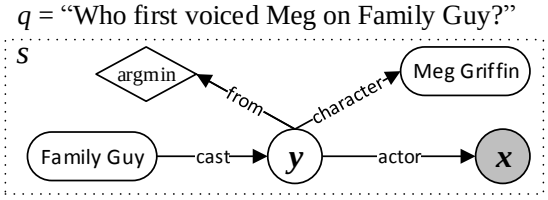
The features we designed essentially match specific portions of the graph to the question, and generally correspond to the staged actions described previously, including:

Topic Entity The score returned by the entity linking system is directly used as a feature.

Core Inferential Chain We use similarity scores of different CNN models described in Sec. 3.2.1 to measure the quality of the core inferential chain. **PatChain** compares the pattern (replacing the topic entity with an entity symbol) and the predicate sequence. **QuesEP** concatenates the canonical name of the topic entity and the predicate sequence, and compares it with the question. This feature conceptually tries to verify the entity linking suggestion. These two CNN models are learned using pairs of the question and the inferential chain of the parse in the training data. In addition to the *in-domain* similarity features, we also train a **ClueWeb** model using the Freebase annotation of ClueWeb corpora (Gabrilovich et al., 2013). For two entities in a sentence that can be linked by one or two predicates, we pair the sentences and predicates to form a parallel corpus to train the CNN model.

Constraints & Aggregations When a constraint node is present in the graph, we use some simple features to check whether there are words in the question that can be associated with the constraint entity or property. Examples of such features include whether a mention in the question can be linked to this entity, and the percentage of the words in the name of the constraint entity appear in the question. Similarly, we check the existence of some keywords in a pre-compiled list, such as "first", "current" or "latest" as features for aggregation nodes such as $\arg\min$. The complete list of these simple word matching features can also be found in Appendix B.

Overall The number of the answer entities retrieved when issuing the query to the knowledge base and the number of nodes in the query graph are both included as features.



(1) EntityLinkingScore(FamilyGuy, "Family Guy") = 0.9
(2) PatChain("who first voiced meg on <e>", cast-actor) = 0.7
(3) QuesEP(*q*, "family guy cast-actor") = 0.6
(4) ClueWeb("who first voiced meg on <e>", cast-actor) = 0.2
(5) ConstraintEntityWord("Meg Griffin", *q*) = 0.5
(6) ConstraintEntityInQ("Meg Griffin", *q*) = 1
(7) AggregationKeyword(argmin, *q*) = 1
(8) NumNodes(s) = 5
(9) NumAns(s) = 1

Figure 8: Active features of a query graph s. (1) is the entity linking score of the topic entity. (2)-(4) are different model scores of the core chain. (5) indicates 50% of the words in "Meg Griffin" appear in the question q. (6) is 1 when the mention "Meg" in q is correctly linked to `MegGriffin` by the entity linking component. (8) is the number of nodes in s. The knowledge base returns only 1 entity when issuing this query, so (9) is 1.

To illustrate our feature design, Fig. 8 presents the active features of an example query graph.

3.4.2 Learning

In principle, once the features are extracted, the model can be trained using any standard off-the-shelf learning algorithm. Instead of treating it as a binary classification problem, where only the correct query graphs are labeled as positive, we view it as a ranking problem. Suppose we have several candidate query graphs for each question[4]. Let g_a and g_b be the query graphs described in states s_a and s_b for the same question q, and the entity sets A_a and A_b be those retrieved by executing g_a and g_b, respectively. Suppose that A is the labeled answers to q. We first compute the precision, recall and F_1 score of A_a and A_b, compared with the gold answer set A. We then rank s_a and s_b by their F_1 scores[5]. The intuition behind is that even if a query is not completely correct, it is still preferred than some other totally incorrect queries. In this work, we use a one-layer neural network model based on lambda-rank (Burges, 2010) for training the ranker.

[4]We will discuss how to create these candidate query graphs from question/answer pairs in Sec. 4.1.

[5]We use F_1 partially because it is the evaluation metric used in the experiments.

4 Experiments

We first introduce the dataset and evaluation metric, followed by the main experimental results and some analysis.

4.1 Data & evaluation metric

We use the WEBQUESTIONS dataset (Berant et al., 2013), which consists of 5,810 question/answer pairs. These questions were collected using Google Suggest API and the answers were obtained from Freebase with the help of Amazon MTurk. The questions are split into training and testing sets, which contain 3,778 questions (65%) and 2,032 questions (35%), respectively. This dataset has several unique properties that make it appealing and was used in several recent papers on semantic parsing and question answering. For instance, although the questions are not directly sampled from search query logs, the selection process was still biased to commonly asked questions on a search engine. The distribution of this question set is thus closer to the "real" information need of search users than that of a small number of human editors. The system performance is basically measured by the ratio of questions that are answered correctly. Because there can be more than one answer to a question, *precision*, *recall* and F_1 are computed based on the system output for each individual question. The average F_1 score is reported as the main evaluation metric[6].

Because this dataset contains only question and answer pairs, we use essentially the same search procedure to *simulate* the semantic parses for training the CNN models and the overall reward function. Candidate topic entities are first generated using the same entity linking system for each question in the training data. Paths on the Freebase knowledge graph that connect a candidate entity to at least one answer entity are identified as the core inferential chains[7]. If an inferential-chain query returns more entities than the correct answers, we explore adding constraint and aggregation nodes, until the entities retrieved by the query graph are identical to the labeled answers, or the F_1 score cannot be increased further. Negative examples are sampled from of the incorrect candidate graphs generated during the search process.

[6]We used the *official* evaluation script from `http://www-nlp.stanford.edu/software/sempre/`.

[7]We restrict the path length to 2. In principle, parses of shorter chains can be used to train the initial reward function, for exploring longer paths using the same search procedure.

Method	Prec.	Rec.	F_1
(Berant et al., 2013)	48.0	41.3	35.7
(Bordes et al., 2014b)	-	-	29.7
(Yao and Van Durme, 2014)	-	-	33.0
(Berant and Liang, 2014)	40.5	46.6	39.9
(Bao et al., 2014)	-	-	37.5
(Bordes et al., 2014a)	-	-	39.2
(Yang et al., 2014)	-	-	41.3
(Wang et al., 2014)	-	-	45.3
Our approach – STAGG	52.8	60.7	**52.5**

Table 1: The results of our approach compared to existing work. The numbers of other systems are either from the original papers or derived from the evaluation script, when the output is available.

In the end, we produce 17,277 query graphs with none-zero F_1 scores from the training set questions and about 1.7M completely incorrect ones.

For training the CNN models to identify the core inferential chain (Sec. 3.2.1), we only use 4,058 chain-only query graphs that achieve $F_1 = 0.5$ to form the parallel question and predicate sequence pairs. The hyper-parameters in CNN, such as the learning rate and the numbers of hidden nodes at the convolutional and semantic layers were chosen via cross-validation. We reserved 684 pairs of patterns and inference-chains from the whole training examples as the held-out set, and the rest as the initial training set. The optimal hyper-parameters were determined by the performance of models trained on the initial training set when applied to the held-out data. We then fixed the hyper-parameters and retrained the CNN models using the whole training set. The performance of CNN is insensitive to the hyper-parameters as long as they are in a reasonable range (e.g., 1000 ± 200 nodes in the convolutional layer, 300 ± 100 nodes in the semantic layer, and learning rate $0.05 \sim 0.005$) and the training process often converges after ~ 800 epochs.

When training the reward function, we created up to 4,000 examples for each question that contain all the positive query graphs and randomly selected negative examples. The model is trained as a ranker, where example query graphs are ranked by their F_1 scores.

4.2 Results

Tab. 1 shows the results of our system, STAGG (Staged query graph generation), compared to existing work[8]. As can be seen from the table, our system outperforms the previous state-of-the-art method by a large margin – 7.2% absolute gain.

Method	#Entities	# Covered Ques.	# Labeled Ent.
Freebase API	19,485	3,734 (98.8%)	3,069 (81.2%)
Ours	9,147	3,770 (99.8%)	3,318 (87.8%)

Table 2: Statistics of entity linking results on training set questions. Both methods cover roughly the same number of questions, but Freebase API suggests twice the number of entities output by our entity linking system and covers fewer topic entities labeled in the original data.

Given the staged design of our approach, it is thus interesting to examine the contributions of each component. Because topic entity linking is the very first stage, the quality of the entities found in the questions, both in precision and recall, affects the final results significantly. To get some insight about how our topic entity linking component performs, we also experimented with applying Freebase Search API to suggest entities for possible mentions in a question. As can be observed in Tab. 2, to cover most of the training questions, we only need half of the number of suggestions when using our entity linking component, compared to Freebase API. Moreover, they also cover more entities that were selected as the topic entities in the original dataset. Starting from those 9,147 entities output by our component, answers of 3,453 questions (91.4%) can be found in their neighboring nodes. When replacing our entity linking component with the results from Freebase API, we also observed a significant performance degradation. The overall system performance drops from 52.5% to 48.4% in F_1 (Prec = 49.8%, Rec = 55.7%), which is 4.1 points lower.

Next we test the system performance when the query graph has just the core inferential chain. Tab. 3 summarizes the results. When only the PatChain CNN model is used, the performance is already very strong, outperforming all existing work. Adding the other CNN models boosts the performance further, reaching 51.8% and is only slightly lower than the full system performance. This may be due to two reasons. First, the questions from search engine users are often short and a large portion of them simply ask about properties of an entity. Examining the query graphs generated for training set questions, we found that 1,888

[8]We do not include results of (Reddy et al., 2014) because they used only a subset of 570 test questions, which are not directly comparable to results from other work. On these 570 questions, our system achieves 67.0% in F_1.

Method	Prec.	Rec.	F_1
PatChain	48.8	59.3	49.6
+QuesEP	50.7	60.6	50.9
+ClueWeb	51.3	62.6	51.8

Table 3: The system results when only the inferential-chain query graphs are generated. We started with the PatChain CNN model and then added QuesEP and ClueWeb sequentially. See Sec. 3.4 for the description of these models.

(50.0%) can be answered exactly (i.e., $F_1 = 1$) using a chain-only query graph. Second, even if the correct parse requires more constraints, the less constrained graph still gets a partial score, as its results cover the correct answers.

4.3 Error Analysis

Although our approach substantially outperforms existing methods, the room for improvement seems big. After all, the accuracy for the intended application, question answering, is still low and only slightly above 50%. We randomly sampled 100 questions that our system did not generate the completely correct query graphs, and categorized the errors. About one third of errors are in fact due to label issues and are not real mistakes. This includes label error (2%), incomplete labels (17%, e.g., only one song is labeled as the answer to "What songs did Bob Dylan write?") and acceptable answers (15%, e.g., "Time in China" vs. "UTC+8"). 8% of the errors are due to incorrect entity linking; however, sometimes the mention is inherently ambiguous (e.g., AFL in "Who founded the AFL?" could mean either "American Football League" or "American Federation of Labor"). 35% of the errors are because of the incorrect inferential chains; 23% are due to incorrect or missing constraints.

5 Related Work and Discussion

Several semantic parsing methods use a domain-independent meaning representation derived from the combinatory categorial grammar (CCG) parses (e.g., (Cai and Yates, 2013; Kwiatkowski et al., 2013; Reddy et al., 2014)). In contrast, our query graph design matches closely the graph knowledge base. Although not fully demonstrated in this paper, the query graph can in fact be fairly expressive. For instance, negations can be handled by adding tags to the constraint nodes indicating that certain conditions *cannot* be satisfied. Our graph generation method is inspired by (Yao and Van Durme, 2014; Bao et al., 2014). Unlike traditional semantic parsing approaches, it uses the knowledge base to help prune the search space when forming the parse. Similar ideas have also been explored in (Poon, 2013).

Empirically, our results suggest that it is crucial to identify the core inferential chain, which matches the relationship between the topic entity in the question and the answer. Our CNN models can be analogous to the embedding approaches (Bordes et al., 2014a; Yang et al., 2014), but are more sophisticated. By allowing parameter sharing among different question-pattern and KB predicate pairs, the matching score of a rare or even unseen pair in the training data can still be predicted precisely. This is due to the fact that the prediction is based on the shared model parameters (i.e., projection matrices) that are estimated using all training pairs.

6 Conclusion

In this paper, we present a semantic parsing framework for question answering using a knowledge base. We define a *query graph* as the meaning representation that can be directly mapped to a logical form. Semantic parsing is reduced to query graph generation, formulated as a staged search problem. With the help of an advanced entity linking system and a deep convolutional neural network model that matches questions and predicate sequences, our system outperforms previous methods substantially on the WEBQUESTIONS dataset.

In the future, we would like to extend our query graph to represent more complicated questions, and explore more features and models for matching constraints and aggregation functions. Applying other structured-output prediction methods to graph generation will also be investigated.

Acknowledgments

We thank the anonymous reviewers for their thoughtful comments, Ming Zhou, Nan Duan and Xuchen Yao for sharing their experience on the question answering problem studied in this work, and Chris Meek for his valuable suggestions. We are also grateful to Siva Reddy and Jonathan Berant for providing us additional data.

References

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer.

Junwei Bao, Nan Duan, Ming Zhou, and Tiejun Zhao. 2014. Knowledge-based question answering as machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 967–976, Baltimore, Maryland, June. Association for Computational Linguistics.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1415–1425, Baltimore, Maryland, June. Association for Computational Linguistics.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA, October. Association for Computational Linguistics.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, New York, NY, USA. ACM.

Antoine Bordes, Sumit Chopra, and Jason Weston. 2014a. Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 615–620, Doha, Qatar, October. Association for Computational Linguistics.

Antoine Bordes, Jason Weston, and Nicolas Usunier. 2014b. Open question answering with weakly supervised embedding models. In *Proceedings of ECML-PKDD*.

Jane Bromley, James W. Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a "Siamese" time delay neural network. *International Journal Pattern Recognition and Artificial Intelligence*, 7(4):669–688.

Christopher JC Burges. 2010. From RankNet to LambdaRank to LambdaMart: An overview. *Learning*, 11:23–581.

Qingqing Cai and Alexander Yates. 2013. Large-scale semantic parsing via schema matching and lexicon extension. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 423–433, Sofia, Bulgaria, August. Association for Computational Linguistics.

Evgeniy Gabrilovich, Michael Ringgaard, and Amarnag Subramanya. 2013. FACC1: Freebase annotation of ClueWeb corpora, version 1. Technical report, June.

Jianfeng Gao, Patrick Pantel, Michael Gamon, Xiaodong He, Li Deng, and Yelong Shen. 2014. Modeling interestingness with deep neural networks. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for Web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2333–2338. ACM.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1545–1556, Seattle, Washington, USA, October. Association for Computational Linguistics.

Percy Liang. 2013. Lambda dependency-based compositional semantics. Technical report, arXiv.

Hoifung Poon. 2013. Grounded unsupervised semantic parsing. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 933–943.

Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2:377–392.

Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Gregoire Mesnil. 2014a. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 101–110. ACM.

Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014b. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion*, pages 373–374.

Zhenghao Wang, Shengquan Yan, Huaming Wang, and Xuedong Huang. 2014. An overview of Microsoft Deep QA system on Stanford WebQuestions benchmark. Technical Report MSR-TR-2014-121, Microsoft, Sep.

Yi Yang and Ming-Wei Chang. 2015. S-MART: Novel tree-based structured learning algorithms applied to tweet entity linking. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Min-Chul Yang, Nan Duan, Ming Zhou, and Hae-Chang Rim. 2014. 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. Association for Computational Linguistics.

Xuchen Yao and Benjamin Van Durme. 2014. Information extraction over structured data: Question answering with Freebase. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 956–966, Baltimore, Maryland, June. Association for Computational Linguistics.

Wen-tau Yih, Xiaodong He, and Christopher Meek. 2014. Semantic parsing for single-relation question answering. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 643–648, Baltimore, Maryland, June. Association for Computational Linguistics.

Appendix A. Formal search algorithm

Given an input question, the query graph generation process can be formally defined as a search problem by a tuple $\langle \mathcal{S}, \mathcal{A}, \Pi, T, \gamma \rangle$. $\mathcal{S} = \bigcup \{\{\phi\}, \mathcal{S}_e, \mathcal{S}_p, \mathcal{S}_c\}$ is the set of states, where each state is an empty graph (ϕ), a single-node graph with the topic entity ($\mathcal{S}_e$), a core inferential chain ($\mathcal{S}_p$), or a more complex query graph with additional constraints ($\mathcal{S}_c$). $\mathcal{A} = \bigcup \{\mathcal{A}_e, \mathcal{A}_p, \mathcal{A}_c, \mathcal{A}_a\}$ is the set of actions. $\mathcal{A}_e$ picks an entity node; $\mathcal{A}_p$ determines the core inferential chain; $\mathcal{A}_c$ and $\mathcal{A}_a$ add constraints and aggregation nodes, respectively. Not all the actions are valid to a given state. We use $\Pi : \mathcal{S} \rightarrow 2^{\mathcal{A}}$ to denote the legitimate set of actions. Specifically, $\Pi(\phi) = \mathcal{A}_e$; $\Pi(s_e) = \mathcal{A}_p, \forall s_e \in \mathcal{S}_e$; $\Pi(s) = \mathcal{A}_c \cup \mathcal{A}_a, \forall s \in \mathcal{S}_p \cup \mathcal{S}_c$. $T(s, a) \rightarrow s'$ is the transition function that maps a state s and an action a to the next state s'. $\gamma : \mathcal{S} \rightarrow \mathbb{R}$ is the reward function that depends on the input question and is defined over the state space.

Staring from $s_0 = \phi$, valid actions are iteratively applied to the current best state in the priority queue. The procedure stops when there is no more state to examine. Algorithm 1 shows the pseudo code. The search procedure essentially keeps up to N candidate states in the priority queue ($N = 1000$ in this work), and explores as many legitimate states as possible to find the best query graph, according to the reward function γ. Obviously, whether the approach can work in practice depends on the quality of the reward function, as well as whether the search space can be effectively controlled.

Algorithm 1 Staged query graph generation

Require: Priority queue H with limited size N

$s_o \leftarrow \phi$; $r_o \leftarrow -\infty$
H.add(s_o,r_o)
while H is not empty **do**
  $s, r \leftarrow H$.pop()
  if $r > r_o$ **then**
    $s_o \leftarrow s$; $r_o \leftarrow r$;
  end if
  for all $a \in \Pi(s)$ **do**
    $s' \leftarrow T(s, a)$
    H.add($s', \gamma(s')$)
  end for
end while
return s_o

Appendix B. Implementation detail for constraints and aggregations

Based on the observation on the training set questions, we developed the following rules to determine whether we would consider adding a constraint or an aggregation node to a CVT node to extend a query graph.

- The constraint entity occurs in the question.
- The constraint predicate indicates the ending time of the event, but there is no value. (This indicates that it's a current event.)
- Some words of the constraint entity's name appear in the question.
- The predicate is `people.marriage.type_of_union`. (This indicates whether the relationship is domestic partnership, marriage or civil union.)
- The question contains "first" or "oldest" and the predicate is a "from" predicate (indicating the starting time of an event).
- The question contains "last", "latest" or "newest" and the predicate is a "to" predicate (indicating the ending time of an event).

For an answer node, we will only add a constraint node if the predicate is one of

- `people.person.gender`
- `common.topic.notable_types`

- `common.topic.notable_for`

Features of constraints and aggregations are indicator functions of whether some of the above rules are true, or simple statistics related to some rules. Again, these features were developed through the analysis on the training data. For a CVT node, the features are:

- Whether the constraint entity appears in the question.
- Whether it's a current event: the constraint predicate indicates the ending time of the event, but there is no value.
- Whether it's a current event and the question has one or more of the keywords "currently", "current", "now", "present" and "presently".
- The percentage of the words in the constraint entity that appear in the question.
- The type of the constraint predicate `people.marriage.type_of_union`.
- Whether the question contains "first" or "oldest" and the predicate is a "from" predicate, and the CVT node is the first when ordered by this "from" property.
- Whether the question contains "last", "latest" or "newest" and the predicate is a "to" predicate, and the CVT node is the last when ordered by this "to" property.

For the answer node, the features are:

- Gender consistency – male: whether the constraint predicate is `gender` and one of the male keywords {"dad", "father", "brother", "grandfather", "grandson", "son", "husband"} appears in the question.
- Gender consistency – female: whether the constraint predicate is `gender` and one of the female keywords {"mom", "mother", "sister", "grandmother", "granddaughter", "daughter", "wife"} appears in the question.
- The percentage of the words in the constraint entity that appear in the question when the constraint predicate is `notable_types` or `notable_for`.

Method	Prec.	Rec.	F_1
PatChain	46.0	54.5	45.6
+QuesEP	47.8	55.1	46.4
+ClueWeb	48.5	57.9	48.0

Table 4: The system results when only the inferential-chain query graphs are generated. The settings here are the same as those in Tab. 3, except that topic entities are identified using the Freebase API, instead of our entity linking component.

Appendix C. Freebase API for entity linking

We conducted a similar set of experiments by replacing our entity linking component with results from Freebase API, and summarized the results in Tab. 4. Comparing Tables 3 and 4, we can see that the negative impact of using an inferior entity linking component is consistent, leading to an approximately 4-point drop across different models for matching the core inferential chain.