Learning to Answer Complex Questions over Knowledge Bases with Query Composition

Nikita Bhutani*
Xinyi Zheng*
nbhutani@umich.edu
zxycarol@umich.edu
University of Michigan, Ann Arbor

H V Jagadish
University of Michigan, Ann Arbor
jag@umich.edu

ABSTRACT

Recent years have seen a surge of knowledge-based question answering (KB-QA) systems which provide crisp answers to user-issued questions by translating them to precise structured queries over a knowledge base (KB). A major challenge in KB-QA is bridging the gap between natural language expressions and the complex schema of the KB. As a result, existing methods focus on simple questions answerable with one main relation path in the KB and struggle with complex questions that require joining multiple relations. We propose a KB-QA system, TextRay, which answers complex questions using a novel *decompose-execute-join* approach. It constructs complex query patterns using a set of simple queries. It uses a semantic matching model which is able to learn simple queries using implicit supervision from question-answer pairs, thus eliminating the need for complex query patterns. Our proposed system significantly outperforms existing KB-QA systems on complex questions while achieving comparable results on simple questions.

CCS CONCEPTS

• **Information systems** → **Question answering**; *Query representation.*

KEYWORDS

question answering, complex questions, neural networks

ACM Reference Format:
Nikita Bhutani, Xinyi Zheng, and H V Jagadish. 2019. Learning to Answer Complex Questions over Knowledge Bases with Query Composition. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357384.3358033

1 INTRODUCTION

Question answering over knowledge bases (KB-QA) is a topic of tremendous practical value and research interest. KBs such as Freebase [6] and YAGO [12] have become pivotal in providing precise answers to user's questions asked in natural language. An important direction in KB-QA is based on semantic parsing, where a question is mapped to a structured query over the KB. Consider the simple question 1 in Fig. 1 and its corresponding query,

select ?x **where** {
Martin_Luther_King person.education ?c .
?c education_institution ?x .
?x is_a College . }

which is constructed by mapping expressions in the question to query components, namely a topic entity (Martin_Luther_King), a main relation path (person.education-education_institution) and any constraints (answer type college). The answers are retrieved by executing the query over the KB.

A key challenge in KB-QA is learning how to bridge the gap between natural language expressions in the questions and the complex schema of the KB using only question-answer pairs. As a result, KB-QA systems have focused on simple questions which can be answered by querying a single relation (or path) in the KB. Little effort has been made to support compositional questions where queries involve joining multiple relations. We call such questions *complex questions* throughout this paper. Consider example question 3 in Fig. 1. The corresponding query has multiple query components: multiple topic entities (Brazil and Portuguese) and more than one main relation (import_from-import_by and spoken_in). Generating

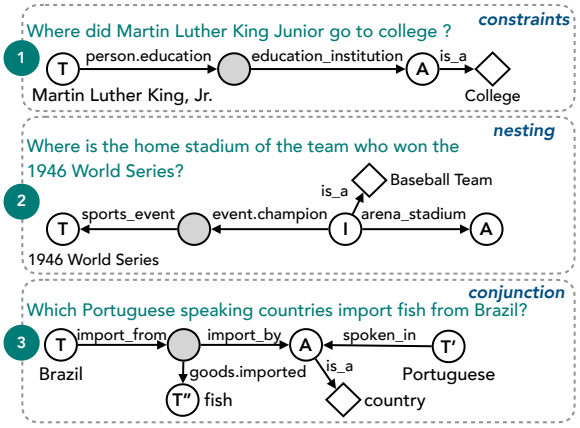

Figure 1: Example queries with constrained main relation (1) and multiple main relations (2 and 3). A main relation connects a topic T to an intermediate answer I or query answer A. The relation could be a n-ary relation, indicated in grey.

*Both authors contributed equally to this research.

CIKM '19, November 3–7, 2019, Beijing, China

ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3358033

such complex queries is much harder due to the structural complexity of the query patterns and the many expressions in the questions mapping to query components.

Traditionally, KB-QA systems handled compositionality by using query templates [1, 8, 25, 35]. While templates can encode complex query patterns, they inevitably have limited coverage. Modern KB-QA systems [2, 4, 13, 30] offer better coverage by modeling semantic parsing as a query generation task, where the goal is to construct a query pattern for a question using likely candidates for its query components. Candidates for query patterns are collected using bottom-up parsing or staged generation methods. The query candidates are ranked based on their semantic similarity to the question, and the best candidate is executed over the KB. Although more robust, these KB-QA systems face two major challenges: a) searching for good candidate query patterns, and b) learning the semantic matching function.

When a query becomes complex, the number of candidates grows exponentially with the number of query components and joins. Consider the following example where the topic 1946_World_Series is 3 hops from the answer. Given the topic entity, collecting and scoring all 3-hop paths to find the most likely path is too expensive.

Example 1. Where is the home stadium of the team who won the 1946 World Series?
Partial Queries:
q_1: ?a event.champion ?c . ?c sports_event 1946_World_Series .
q_2: ?b arena_stadium ?x .
Join: q_1 join$_{?a=?b}$ q_2
Execute: ans = Busch Stadium

We hypothesize that complex query patterns can be constructed by joining a set of simple queries. Simple queries have fewer query components and thus, fewer candidates. In the example above, q_1 and q_2 have fewer components than the original query. Scoring the candidates for a simple query and executing the best query (q_1) can yield intermediate answers (Cardinals for ?a), which restrict the search space for subsequent simple queries (?b in q_2 binds to answers of q_1). The process of decomposing a complex query into simple queries makes the search for complex patterns tractable.

A natural question then is how to decompose the query generation process and learn the semantic matching function for simple queries. A simple approach is to annotate the linguistic parse tree of a question with query components and learn to map the tree elements to a query pattern [16, 34]. The mapping can be learned from example questions annotated with complex query patterns. Obtaining complex query patterns, however, can be cumbersome and error-prone. Recent works [4, 20] have shown that the mapping can instead be learned using distant supervision from question-answer pairs. While this works for simple questions, it is challenging for complex questions where only answers (e.g., Busch Stadium) to complex questions are available and not the simple queries (e.g., St. Louis Cardinals for q_1) that constitute the complex query. We propose that by restricting each simple query to a single relation path and by leveraging some prior domain knowledge, a semantic parser can be trained to answer simple queries using only implicit supervision for the simple queries.

We have constructed a KB-QA system, TextRay that learns to answer complex questions using only question-answer pairs. It adopts a *decompose-execute-join* approach, where it constructs a complex query by joining a sequence of simple queries. Each simple query focuses on one relation path, which ensures the search for its candidates is efficient and implicit supervision signals are reliable for learning the semantic parser. For training the semantic parser, it estimates the quality of a simple query candidate indirectly based on the answers retrieved by its future complete query candidates. It further incorporates simple, light-weight domain knowledge to improve the quality of the weak, implicit supervision signals. To summarize, this paper makes the following contributions:

- We present a new KB-QA system, TextRay, that automatically translates a given complex, compositional question to the matching query over a knowledge base (Section 3) .
- We propose a novel *decompose-execute-join* approach to construct a complex query pattern from partial queries. This enables efficient search for good candidates for a complex query (Section 4).
- We present a neural-network based semantic matching model that learns to score candidates for partial queries using implicit supervision from question-answer pairs. It uses an effective scoring strategy for candidates that combines the quality of full-query derivations of a candidate with domain knowledge (Section 5).
- We provide an extensive evaluation of our system on multiple QA datasets, where it significantly outperforms previous approaches on complex questions (Section 6).

We introduce key concepts in Section 2. We outline problems studied in this paper and our approach in Section 3 before providing details in Section 4-5. We present experiments in Section 6 and related work in Section 7.

2 BACKGROUND

Our goal is to design a KB-QA system that can map a complex question Q in natural language to a matching query G, which can be executed against a knowledge base $\mathcal{K}$ to retrieve answers to Q.
Knowledge Base. A knowledge base $\mathcal{K}$ is a collection of triples of the form of (s, r, o), where s, r and o denote subject, relation and object respectively. A triple can also be interpreted as a directed edge from s to o labeled with relation r, and $\mathcal{K}$ as a directed graph. Higher-order relations are expressed using special mediator nodes.
Complex Question. A complex question Q in natural language corresponds to a query G over the $\mathcal{K}$ such that G involves joining multiple main relations in the $\mathcal{K}$. G has a single query focus ?x that corresponds to the answer to Q. G can be interpreted as a sequence of simple partial queries $G = (G_1, G_2, ..., G_o)$ connected via different join conditions. Each partial query corresponds to a main relation in G. Partial queries may share the query focus ?x (e.g., example 3 in Fig. 1) or a variable (e.g., example 2 in Fig. 1).
Computation Plan. A computation plan C is a tree that decides how a complex query G is constructed and executed from the partial queries. It uses two main functions: *simQA* and *join*. *simQA* denotes search and execution of likely partial query candidates. *join* denotes the join condition for two partial queries. Fig. 2 shows computation plans for running examples 2 and 3.

3 SOLUTION OVERVIEW

Fig. 3 shows the design of a system that answers complex questions with simple queries. There are several key steps in the process.

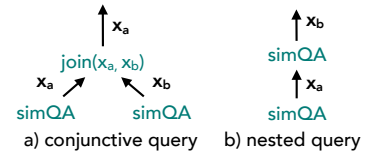

Figure 2: Example computation plan indicating how to construct the complex query given the partial queries

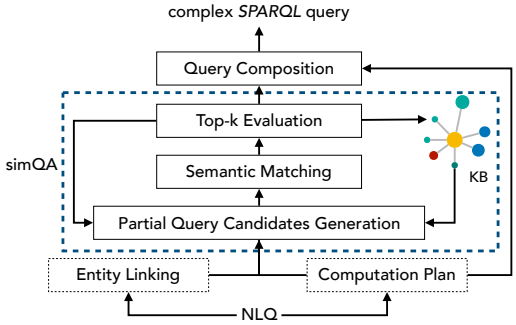

Figure 3: System Architecture

Given a question Q, the system generates a computation plan that describes how its matching query G can be broken down into partial queries G_i. Next, based on the computation plan it finds candidates for the partial queries $\{G_i^{(k)}\}_{k=1}^{L}$. For instance, candidates for G_1 and G_2 can be collected simultaneously for a conjunctive question with a computation plan a) in Fig. 2. On the other hand, the search for G_2 can benefit from the answers of G_1 for the computation plan b). Given the candidates for partial queries, it computes their semantic similarity $S_{sem}(Q, G_i^{(k)})$ to the question. The best M candidates for each partial query are executed, their answers help find the candidates for the subsequent query and so on. In this manner, multiple full-query derivations are generated from simple query candidates. The derivation with the highest overall score is finally executed to find answers to the complex question.

The system needs a model for computing the semantic similarity of the partial query candidates. The model can be learned offline using a set of question-answer pairs. Learning to guess a good candidate from a set of candidates requires a set of positive and negative examples of (question-partial query) pairs. The system generates these examples based on implicit supervision, i.e., whether any of full-query derivations of a partial query can generate the labeled answers for the question. Since such implicit supervision can be susceptible to spurious queries, it incorporates simple, light-weight domain knowledge as priors in scoring the candidates.

Embodying these ideas, we design a system TextRay that adopts a *decompose-execute-join* approach to answer complex questions. Our query composition approach and our semantic matching model trained with weak, implicit supervision are the major contributions of this paper. Formally, these tasks can be defined as follows:

Query Composition. Given a complex question Q and its computation plan C, find a sequence of partial queries $(G_1, G_2, ..., G_o)$ and construct the complex query pattern G such that executing G over $\mathcal{K}$ provides answers to Q. To find a correct partial query G_i, we have to collect candidates $\{G_i^{(k)}\}_{k=1}^{L}$, score them and reserve the best candidates for finding G_{i+1}. Given K-best full-query derivations $G^{(k)} = (G_1^{(k)}, G_2^{(k)}, ..., G_o^{(k)})$, we have to find the derivation G^* that best captures the meaning of Q. (Section 4)

Semantic Matching. Given a question Q and a partial query candidate $G_i^{(k)}$, a semantic matching model provides a semantic similarity score $S_{sem}(Q, G_i^{(k)})$. We have to learn the model offline using distant supervision from a collection of question answer-set pairs $\{Q^i, A^i\}_{i=1}^{N}$, where Q_i is a complex question and A_i is the set of entities from $\mathcal{K}$ that are answers to the complex question. (Section 5)

4 QUERY COMPOSITION

We assume that a complex question can be answered using a sequence of partial queries, each targeting one main relation in the knowledge base $\mathcal{K}$. This offers several benefits. First, it ensures query composition is tractable. Consider a computation plan that describes how a complex query pattern can be constructed by joining one component (a node or a relation edge) at a time. Obtaining a plan of this granularity will be tedious and error-prone. In contrast, it is easy to obtain a plan that only describes how partial queries must be joined. Also, independently matching each component will lose useful information for collectively resolving components in the partial queries. Lastly, executing more specific partial query patterns will be more efficient than query patterns for individual components. For instance, a query pattern q: Portuguese spoken_in ?o will find fewer matches to join with than q: Portuguese ?r ?o .

Identifying entities. In order to find candidates, we begin by identifying entities from $\mathcal{K}$ that are mentioned in the question. In our example question 3 from Fig. 1, mention *'Portuguese'* refers to the language Portuguese or dialect Brazilian_Portuguese in $\mathcal{K}$, and *'Brazil'* refers to the South American country in $\mathcal{K}$. We use an entity linking system [3, 28] that returns a set of possibly overlapping pairs $E = \{(mention, entity)\}$ with attached confidence scores. We consider 10 best pairs based on their confidence scores.

Constructing Computation Plan. Even though a computation plan can include aggregations and value comparisons, we assume it includes two operations, *simQA* and *join* for simplicity. The *simQA* operator denotes search for a partial query, and *join* describes the join condition of two partial queries. Post-order traversal of the plan yields a sequence in which the partial queries are executed: $z = z_1, z_2, \ldots, z_n$, where $z_i \in \{simQA, join\}$. Note that we do not chunk the original question into sub-questions but simply encode the number of partial queries required to construct the query G. We obtain the computation plan using augmented pointer networks [24] trained to predict its likelihood as $P(C|Q) = \prod_{i=1}^{j} P(z_i|Q, z_{1:i-1})$. In practice, however, the number of main relations is small (2-3). A computation plan can also be approximated using simple syntactic cues such as the number of verb phrases in the question. The dependencies can be estimated from the type of clause connectors in the question i.e., coordinating (example 1 in Fig. 2) vs subordinating clause (example 2 in Fig. 2).

4.1 Partial Query Candidate Generation

We next have to construct the complex query pattern using the sequence described in the computation plan. We generate candidates for a partial query by a staged generation method, measure their

semantic similarities to the question and find the best candidate (*simQA* operation). Compared to previous methods [2, 30], we tailor the staged generation strategy to handle compositionality. The candidate generation process is described by a set of states S and actions A. We introduce a new state S_t and action A_t (Fig. 4).

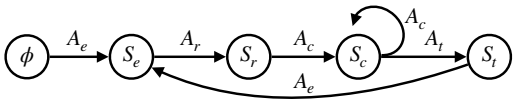

Figure 4: Action space showing how to construct a query.

States $S = \{\phi, S_e, S_r, S_c, S_t\}$ indicate an empty query (ϕ), single entity (S_e), a main relation path (S_r), a constraint (S_c) in a partial query candidate. Action $A = \{A_e, A_r, A_c, A_t\}$ grow a candidate by adding one query component at a time. As shown in Fig. 5, action A_e finds candidates for the seed entity, A_r adds main relation paths to the entity candidates and action A_c adds any constraints. Action A_t denotes termination of the partial query G_i and the transition to a state S_t. In the state S_t, the candidate generation refers to the computation plan and determines how to find candidates for subsequent partial query G_{i+1}. If the next operation is *simQA*, G_{i+1} depends on the answers of G_i. Candidates for G_i, therefore, must be scored and the best candidate be executed. The answers of G_i would become the seed for collecting candidates for G_{i+1}. If the next operation is *join*, candidate generation for G_{i+1} can resume independently (see Fig. 6).

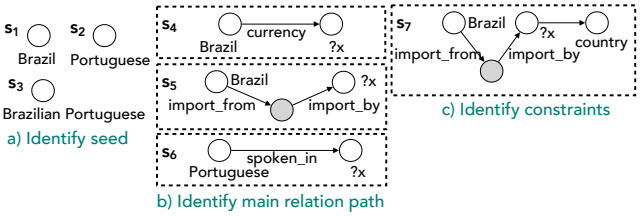

Figure 5: Staged candidate generation for a partial query.

Identify seed entity (A_e). An entity is a seed for a candidate. It could be an entity from $E = \{(mention, entity)\}$ pairs identified by the entity linking system or an answer of a previous partial query. For instance, Brazil, Portuguese or Brazilian Portuguese are seed entities. We only allow entities that are not consumed by a previous partial query candidate. For instance, Brazil cannot be the seed for the subsequent partial query in Fig. 6.

Identify main relation path (A_r). We next find different relation paths in $\mathcal{K}$ that connect the seed entities to answers using a relation edge or a mediated 2-hop relation path. One end of a relation path is a seed entity, and the other end is a variable denoting the answer to the partial query.

Identify constraints (A_c). We try to attach any constraint entities to the main relation and variables. We consider a set of constraints $E_c = \bigcup\{E, E_t, E_o\}$, where E is the set entity links, E_t are entities connected to the answer via specific relations[1], and E_o are named entities of type *date*, *time*. We ignore entities in E_t that do not appear in the question. We next collect 1-hop paths connecting constraint entities to the query. For instance, COUNTRY is a type constraint on the answer in Fig. 5. We consider all subsets of the constraints to accommodate multiple constraints in the query.

Terminate partial query (A_t). After collecting candidates of a partial query G_i, the search refers to the computation plan to determine the dependencies for the subsequent partial query G_{i+1}. When the next operation is a join operation, it indicates G_{i+1} does not share any entities from the question. In that case, the search does not have to wait to generate candidates for G_{i+1}. A non-overlapping entity in E becomes the seed for G_{i+1} (as shown in Fig. 6). When the next operation in the computation plan is *simQA*, it indicates the G_{i+1} relies on the answers to G_i i.e., the seed entities for G_{i+1} are the answer entities of G_i. In such a case, we score the partial query candidates $G_i^{(k)}$ based on their semantic similarity to question Q, and find the K-best query candidates. We translate these candidates to SPARQL queries and execute them against $\mathcal{K}$.

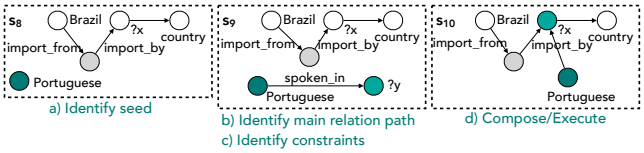

Figure 6: Staged generation for a subsequent partial query.

4.2 Query Composition and Execution

We score candidates of each partial query based on their semantic similarity to the question and execute the best candidates. To maintain tractability and efficiency, we adopt a beam search and only execute k best candidates at each level. The beam search returns a set of k best derivations, where a derivation $G^{(k)} = (G_1^{(k)}, G_2^{(k)}, ..., G_o^{(k)})$ is a sequence of partial queries.

Note that we only compute semantic similarity of partial queries and not derivations. We still need to judge which derivation among the k best derivations is the correct query G^* for the input question. One approach is to simply aggregate the semantic similarities of the partial queries. While such a strategy might work for aggressively pruning the large search space of derivations, we need a more reliable scoring function. We train a log-linear model using a set of (question-answer) pairs for scoring derivations based on the scores of the partial queries and other features. Some features we use are confidence scores of seed entities in the queries, number of constraint entities, number of variables, number of partial queries, number of relation edges, and number of answer entities. Once the features are extracted, the model next has to be trained for ranking and scoring the derivations. Due to the lack of correct queries, one effective strategy to learn the model is to rely on the F_1 score of the predicted answer set of a query to determine its correctness. Given the F_1 scores of the queries, we use a pointwise ranking method to determine the best derivation. We translate the best derivation to a SPARQL query, execute the query and return the set of answers entities as answers to the question.

5 SEMANTIC MATCHING MODEL

5.1 Model Architecture

We use a neural network based model for semantic matching. The architecture of the proposed model is shown in Fig. 7. It encodes

[1] common.topic.notable_types, common.topic.notable_for

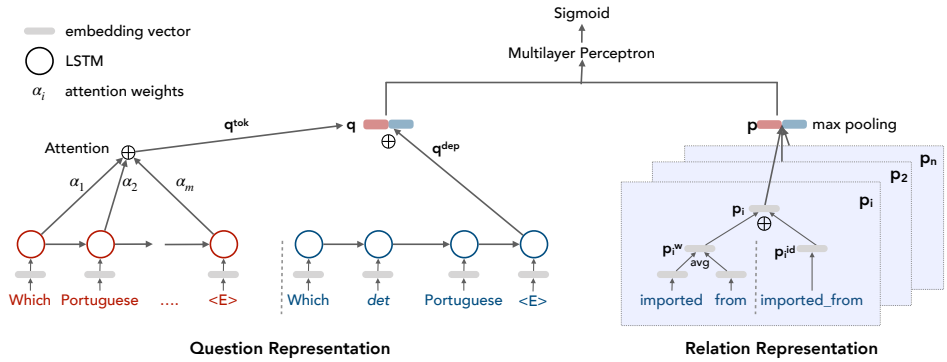

Figure 7: Semantic Matching Model

both the question and query as semantic vectors to find their semantic similarity. We use two Long Short-Term Memory (LSTM) networks and an attention mechanism to learn the latent representation of the question Q. We only consider the main relation path and constraint relations to encode the partial query G_i. This is because the entities in the query are already grounded by the entity linking system, and we only need to infer different relations in the query. We feed the latent vector q for the question and p for the partial query as input to a multi-layer perceptron (MLP). The MLP outputs a scalar that is used as the semantic matching score $S_{sem}(Q, G_i)$. We first describe how we encode a question and a partial query. We then describe in detail other elements of the architecture before discussing the learning objective of the model.

Encoding Question. We encode the question using its word sequence and dependency structure. Since complex questions tend to be longer, they have more long-range dependencies. Encoding the dependency structure thus becomes crucial. Specifically, let $\langle w_1, w_2, \ldots, w_n \rangle$ be the words in Q. We replace the words corresponding to the seed entities with a dummy token w_E, and those corresponding to constraint entities with a dummy token w_C. We then use a word embedding matrix E_w to map the questions words to embedding vectors $\langle q_1^w, q_2^w, \ldots, q_n^w \rangle$. We take the last hidden state of an LSTM network as the latent vector representation q^w. Similarly, we encode the dependencies by considering the dependency tree as a sequence of question words and their dependency labels. We use the same embedding matrix E_w to map the dependency sequence to embedding vectors and use the last hidden state of another LSTM network to obtain latent vector q^{dep}.

Encoding main relation path and constraint relations. We encode the partial query using the various relations in the query. For each relation P_i in the partial query, we consider the sequence of both relation ids and relation names [18]. For example, the id sequence of the main relation in Fig. 7 is {import_by}, while the word sequence is {'import', 'by'}. We convert the word sequence to a sequence of embedding vectors and represent P_i as the average embedding p^w of the embedding vectors. At the id level, we translate the relation directly using another embedding matrix E_p of relations. The semantic vector of P_i then is $p = [p^w; p^{id}]$. We apply max pooling over the hidden vectors of the relations and obtain a compositional semantic representation of the partial query.

Attention mechanism. Our goal is to estimate the semantic similarity of a question vector q and a partial query vector p. Complex questions, however, are long and have expressions for matching multiple partial queries in the KB. Irrelevant information in the question can thus distract the matching. We use an attention mechanism to improve the quality of question representation. The idea behind attention is to learn to emphasize parts of the question that are informative to the given attention vector. Following [11, 19], we use the partial query vector p as the attention vector to highlight relevant parts of the question word vector q^w. Specifically, given all hidden vectors h_t at time step $t \in \{1, 2, \ldots, n\}$, the context vector c is the weighted sum of all the hidden states:

$$c = \sum_{t=1}^{n} \alpha_t h_t$$

where α_t is an attention weight. The weights are computed as:

$$\alpha = softmax(W tanh(W_q q^w + W_p p))$$

where W, W_p, W_q are learnable network parameters. The attention weights can be interpreted as the extent to which the model focuses on a specific word in the question given a partial query.

Objective function. The context vector c, question dependency vector q^{dep} and query vector p are concatenated and fed to a multi-layer perceptron (MLP), a feed-forward neural network with two hidden layers and a scalar output neuron indicating the semantic simlarity score $S_{sem}(q, G_i)$. Our semantic matching model adopts a cross entropy loss function during the training:

$$loss = y log(S_{sem}(q, G_i)) + (1 - y) log(1 - S_{sem}(q, G_i))$$

where $y \in \{0, 1\}$ is a label indicating whether G_i is correct or not. Clearly, the model relies on positive and negative samples of (question, partial query) pairs for training. Constructing such training data is cumbersome and expensive. We next describe how training data can be generated from question-answer pairs.

5.2 Implicit Supervision

If fully-annotated queries were available, training the semantic matching model to reward good partial queries will be straightforward. But obtaining such data is costly and time-consuming. Obtaining questions annotated with the ground answers, however, is easier. In such an implicit supervision setting, the quality of a partial query candidate has to be indirectly measured by computing the F_1 score of the derived answers compared to the labeled answers.

Estimating Candidate Quality. The implicit supervision increases the difficulty of learning a reliable semantic matching model. As the computation plan becomes deeper, the supervision signals become weaker. For instance, it is difficult to estimate the quality of a seed entity solely from the labeled answers to the question. Matching one relation at a time keeps the computation plan simple and provides an opportunity to reliably estimate the quality of partial query candidates despite the implicit and delayed signals.

Our main challenge is that no ground-truth is available for the partial query candidates. The supervision signals are delayed i.e., the quality of a partial query candidate $G_i^{(k)}$ can be estimated only after a full query derivation is constructed and executed. Enumerating and executing all possible full query derivations becomes expensive as the query becomes complex. Intuitively, we can leverage the computation plan to prune the search space of full query derivations, and estimate the quality of a partial query based on the quality of the derivations. Formally, we model quality of $G_i^{(k)}$ as:

$$V(G_i^{(k)}) = \max_{i \le t \le n-1} R(D_{t+1}^{(k)})$$

where D_t is the derivation at level t in the computation plan, n is the number of partial queries and R is the F_1 score of the answers of the derivation. Equivalently, the score of a partial query candidate is the F_1 score of its best future derivations.

Example 2. Consider two candidate partial queries:

$G_1^{(1)}$ = ?a event.champion ?c . ?c sports_event 1946_World_Series .

$G_1^{(2)}$ = 1946_World_Series sports_event.umpire ?a .

Now consider the derivations starting from the candidates, their retrieved answers and the ground answer 'Busch Stadium':

$D_2^{(1)}$ = ?a event.champion ?c . ?c sports_event 1946_World_Series . ?a arena_stadium ?x .

$A(D_2^{(1)})$: Busch Stadium

$D_2^{(2)}$ = 1946_World_Series sports_event.umpire ?a . ?a place_of_birth ?x .

$A(D_2^{(2)})$: Texas, Missouri, Illinois, New Jersey

The F_1 scores and values then are:

$R(D_2^{(1)}) = 1.0, V(G_1^{(1)}) = 1.0, V(G_2^{(1)}) = 1.0$

$R(D_2^{(1)}) = 0.0, V(G_1^{(2)}) = 0.0, V(G_2^{(2)}) = 0.0$

We use the scores $V(G_i^{(k)})$ to approximate the label $y^{(k)} \in \{0, 1\}$ for the candidate. Intuitively, candidates with scores greater than a threshold can be considered as a positive example for the question, and negative otherwise. However, using a fixed threshold might lead to many false negatives for questions where no candidate has a score greater than the threshold. We, therefore, rescale the scores of the candidates by the maximum score of any candidate before estimating the labels for the candidates.

We propose a strategy to estimate the scores efficiently. The key idea is to leverage the computation plan to prune the space of full query derivations. If the computation plan indicates a join between two partial queries, we greedily execute the partial query candidates $G_i^{(k)}$ and $G_{i+1}^{(l)}$ and derive the reward for the derivation $D_{i+1}^{(k)}$ and $D_{i+1}^{(l)}$ from the intersection of the answers of the partial query candidates. If the computation plan indicates a dependency between the results of partial queries, instead of enumerating all possible relation paths for the subsequent query, we only focus on paths leading to the ground answer entity. If no such path exists, it indicates that all derivations of $G_i^{(k)}$ are negative examples. If such a path exists, then we only collect subsequent partial query candidates that use the relation path. This ensures that no true positive examples are missed out.

Addressing Spurious Matches. Implicit supervision strategy described above is susceptible to spurious partial queries which happen to find correct answers but do not capture the semantic meaning of the question. Identifying such spurious partial candidates as positive examples can greatly affect the training data quality and the performance of the semantic matching model. We propose to alleviate this problem by incorporating some domain knowledge as priors for scoring. By qualitatively examining the positive examples, we found that correct candidates tend to use the same words as the natural language question. For instance, a question "*Which governmental jurisdiction held the LIX Legislature of the Mexican congress in 2011 ?*" has a partial query with main relation governmental_jurisdiction.governing_officials sharing words governmental and jurisdiction. Thus, surface-form similarity of the main relation and question can be used to promote true positive and false negative examples. We define $lexical\_score(Q, G_i^{(k)})$ that computes the ratio of number of words in the main relation of $G_i^{(k)}$ that are mentioned in the question Q.

We also use a small hand-crafted lexicon that contains lexical pairs (w_G, w_Q) of words from the relation (w_G) and keywords from the question (w_Q) based on their co-occurrence frequency. Intuitively, mentions of certain words in the question should boost the scores of relations that use words that co-occur with the question words. We define $co\_occur\_score(Q, G_i^{(k)})$ as the fraction of relation words that hit the lexicon. Using these priors the quality of a partial query candidate $G_i^{(k)}$ is given by: $V(G_i^{(k)}) + \gamma lexical\_score(Q, G_i^{(k)}) + \delta co\_occur\_score(Q, G_i^{(k)})$, where γ and δ are hyper-parameters denoting the strength of the priors. We found $\gamma = 0.75$ and $\delta = 0.75$ reduced the number of false positives in the training data.

6 EXPERIMENTS

We present extensive experiments which demonstrate our approach to construct a complex query from partial queries is superior to traditional approaches that map a question directly to a query.

6.1 Datasets and Baseline Systems

We use two KB-QA benchmark datasets.

- **ComplexWebQuestions (CompQWeb)**[24]: This is a recent benchmark designed for highly compositional questions. The dataset contains 34,689 examples, each containing a complex question, answers and a SPARQL query against Freebase. We use the SPARQL queries only for qualitative analysis. The dataset is divided into 27,734 train, 3,480 dev and 3,475 test cases.
- **WebQuestionSP (WebQSP)**[31]: This is an enhanced version of the de facto benchmark dataset WebQuestions (WebQ) [4]. It consists of 4,737 questions and their answers, split into 3,098 train and 1,639 test cases. Additionally, this dataset provides correct SPARQL query for each question. Unlike the CompQWeb dataset, most of the questions in WebQSP are simple. Only 4%

questions in the test set have multiple entity constraints [32]. We evaluate on this dataset to demonstrate our proposed approach is effective across questions of varying complexity. Note that we chose WebQSP over WebQ because the SPARQL queries provide ground truth for qualitative analysis.

Knowledge Bases. We use the entire Freebase[2] dump on 2015-08-02 as the background knowledge source for comparison to baseline systems. We host it with Virtuoso engine [3]. It contains about 46 million entities and 2.9 billion facts. Our approach, however, can be easily adapted to any other structured knowledge base.

Evaluation Metric. Following prior work [4, 18, 30], we use averaged F_1 score of the predicted answers to measure the effectiveness of a KB-QA system. This metric is effective when queries have multiple correct answers. We also compute precision@1 as the fraction of questions that were answered with the exact gold answer [3].

Baseline systems. We compare against two KB-QA systems.

- **CompQA**[18]: It generates full query candidates for complex questions and uses a neural network model for semantic matching. They handle complex query structures by encoding the constraints explicitly in their semantic matching model. We used the code provided by the authors for experiments.
- **Parasempre**[4]: It parses questions to logical forms that are executed against Freebase. It generates query candidates by recursively generating logical forms. We used the publicly-available implementation of their system and their pre-trained model.

Many other KB-QA systems [1, 2, 8, 33] powered by Freebase are designed to handle simple questions with a few additional constraints. We directly report numbers from their papers. The only systems that can handle highly compositional questions [23, 24] do not use Freebase as their knowledge source.

6.2 Experimental Setup

Entity Linking. We used the entity links released by [28] for the WebQSP dataset. Since CompQWeb is a new dataset with no entity links, we ran AQQU [3] to annotate questions with entity links.

Pre-trained embeddings. We initialize the embeddings for word-level representations for questions and relations with Glove vectors of dimension 300 [22]. There were 7,371 (906) words in the questions vocabulary, 2,853 (1,982) words in the relations vocabulary (excluding the stop words) and 6,904 (3,818) unique relations in the ComplexWeb (WebQSP) dataset. We randomly initialize the embeddings for relation ids. The embeddings are not fixed during training and are learned along with other parameters.

Training Semantic Matching Model[4]**.** We used NVIDIA GeForce GTX 1080 Ti GPU for training the semantic matching model. To encode the question word sequence and question dependency structure, we use two bi-directional LSTMs. We set the size of the LSTM hidden layer to 300. For the multi-layer perceptron, we use 1024 as the size of hidden layer and *sigmoid* as the activation function for hidden layer. Based on our training data generation for (question, partial query) pairs, we could find 104k (5,026) positive examples and 2.5M (370k) negative examples for CompQWeb (WebQSP) dataset when using a threshold of 0.5 for the partial query candidate quality score ($V(G^{(k)})$). We upsample the positive examples such that there are 2 negative examples for each positive example. We found similar performances for training data with 4:1 to 1:1 negative examples to positive examples ratio. We chose 2:1 balance ratio in our experiments. We trained the model with a batch size of 64 and used *Adam*[15] optimizer for adaptive learning rate optimization.

Hyperparameter tuning. We created different configurations with learning rate in {0.0005, 0.001, 0.002}, learning rate decay in {0.1, 0.5}, attention dimension size in {100, 300, 500} and number of training epochs in [5, 10]. We used the dev splits for tuning using different configurations. We used learning rate 0.0005, learning rate decay 0.5 and attention dimension size 500.

6.3 Results and Discussion

Effectiveness on Complex Questions. As reported in Table 1, our proposed system TextRay achieves the best performance (with a large margin of 26.82% absolute gain in F_1) among these methods, confirming the potential and effectiveness of TextRay. In contrast to CompQA, we do not assume queries have a fixed number (one) of main relations. By leveraging the computation plan, TextRay can generate candidates for complex queries which can contain any number of main relations. Our decompose-execute-join approach helps find candidates for partial queries that can potentially construct better complex query candidates.

We also achieve significantly higher F_1 than Parasempre that generates query candidates by recursively generating logical forms. By relying on a mapping of natural language expression to relations and a small set of composition rules, it can generate highly compositional logical forms. However, the logical forms use relations from the large vocabulary in the KB instead of the neighborhood of the seed [29]. Also, Parasempre relies heavily on keeping a large beam of good derivations that lead to the correct answer. This becomes the bottleneck when understanding complex questions. In contrast, our staged generation design ensures that only areas of the KB that could potentially lead to a successful query are explored. SplitQA [24] and MHQA [23] can handle complex questions, but rely on noisy textual data sources. Evidently, we could answer complex questions more reliably and effectively from structured knowledge sources which naturally support compositionality.

Method	Average F_1	Precision@1
TextRay	**33.87**	**40.83**
CompQA [18]	4.83	4.83
Parasempre [4]	7.05	12.37
SplitQA (web) [24]	-	27.50
MHQA (web) [23]	-	30.10

Table 1: Average F_1 scores and Precision@1 on CompQWeb.

Effectiveness on Simple Questions. We provide a complimentary evaluation on simple questions to demonstrate TextRay can adapt to different complexities of questions. For these experiments, we assume the computation plan simply has a single *simQA* operation i.e., the question has one main relation. We found TextRay achieved comparable if not higher F_1 scores on the WebQSP dataset as other KB-QA systems that adopt a candidate enumeration-scoring strategy (see Table 2). STAGG [30], a popular KB-QA system

[2] http://commondatastorage.googleapis.com/freebase-public/rdf/freebase-rdf-2015-08-02-00-00.gz

[3] https://virtuoso.openlinksw.com

[4] https://github.com/umich-dbgroup/TextRay-Release

Method	Average F_1	Precision@1
TextRay	60.3	**72.16**
CompQA [18]	59.5	61.64
Parasempre [4]	46.9	51.5
STAGG [30]	**66.8**	67.3
MulCQA [2]	52.4*	-
AQQU [3]	49.4*	-

Table 2: Average F_1 scores and Precision@1 on WebQSP dataset. * are results on the WebQ dataset.

Method	CompQWeb F_1^**	WebQSP F_1^*
TextRay	50.83	84.57
CompQA [18]	31.30	75.03
Parasempre [4]	19.15	60.95

Table 3: Upper bound F_1 scores for candidate generation.

	CompQWeb		WebQSP	
	%	Avg. *best* F_1	%	Avg. *best* F_1
Top-1	48.7	33.87	66.5	60.31
Top-2	55.6	36.05	79.4	69.73
Top-5	65.6	39.97	89.1	76.5
Top-10	71.5	42.42	93.7	79.9

Table 4: Percentage of questions with the highest F_1 score in the top-k candidate derivations, and the average best F_1.

Setup	CompQWeb F_1	WebQSP F_1
TextRay (Full System)	**33.87**	**60.31**
No constraints	28.16	58.39
No attention	29.92	58.31
No priors	31.28	59.43

Table 5: Component-wise ablation results (Average F_1).

uses a similar approach but improves the results using feature engineering and by augmenting entity linking with external knowledge.

Our proposed system can be integrated with better entity linking and more sophisticated model architectures [33] for semantic matching. The semantic matching model, like in TextRay, however, should be able to accommodate for long, complex question sequences (e.g., via attention). Lastly, we found TextRay adapts well to varying complexities of questions compared to other systems tailored to a specific class of questions [2].

Effectiveness of Candidate Generation. It is possible to pretrain a system's semantic matching model on different datasets, which could yield different overall F_1 scores when evaluating a target dataset. Thus, we measure the upper bound F_1^* scores of the candidate queries each system generates. Comparing these upper bounds provides insights into the expressivity of their query composition process, agnostic to the quality of the semantic matching model. We found TextRay could consistently find better candidate queries than other systems, as reflected in the best F_1^* scores of their candidate queries (see Table 3).

Top-k results. Table 4 shows the top-k results on the two datasets. For a large majority of the questions, the query with the highest F_1 score was among the top-10 predictions. Despite using an end-to-end semantic matching model, TextRay generates SPARQL queries that can be executed over the KB. It can be treated as a natural language end-point to the KB. The top-k queries can be provided as alternative interpretations to the user, who can then select a query to execute. This process would be less tedious and error-prone for a user than writing a complex SPARQL query.

6.4 Ablation Study

We also investigate the contributions of various components in complex question answering. Table 5 summarizes these results.

Encoding Constraints. Complex questions tend to have additional constraints on the main path (such as ordinal, answer type, time, etc). Ignoring these constraints can hurt the F_1 score of a query. In the CompQWeb dataset, we observed that 35% of the queries had at least one constraint. Most of the queries (85%) in WebQSP were simple with no constraints. To demonstrate the effectiveness of including additional constraints in query candidates and in semantic matching, we construct simple baseline: candidate generation does not add additional constraints on the main path sequence, and semantic matching model does not encode additional constraints. As can be seen, overall F_1 is higher when constraints are included. The performance gains were smaller on the WebQSP dataset that comprises mostly of simple questions.

Using Attention Mechanism. Attention mechanism helps alleviate the problem of multiple expressions in the question matching multiple relations in the KB. Including attention helps the model focus on parts of the question that are relevant to the main relation. We found the F_1 score dropped by 3.95 when attention was disabled for the CompQWeb dataset. This problem is less severe in simple questions, as reflected in marginal gains in F_1 scores on the WebQSP dataset when attention is included.

Incorporating Domain Knowledge. We observed that the improvement due to prior domain knowledge is 2.6% on CompQWeb. While this may seem marginal, we observe that the improvements were significant over top-k ($k > 1$) results. We found the average best F_1 was 29.47 for top-10 results when no prior knowledge was incorporated in the training and ranking phase. The average best F_1 was 40.06 when prior knowledge was considered.

Ranking. As shown in Table 4, the upper bound for average F_1 score for top-10 query candidates is much higher than the F_1 score for the best query predicted by the model. This indicates that a good re-ranking method can improve the overall performance of the system on the KB-QA task. Our ranking model uses features including topic entity linking scores, the query structure and the semantic similarity score for partial queries. We conducted experiments with two different pointwise-ranking methods: linear regression and logistic regression. We compare with a simple baseline where score of a complex query derivation is simply the sum of scores of the partial queries. For training the regression based model, we used F_1

Variant	CompQWeb F_1	WebQSP F_1
Linear Regression	33.87	60.31
Logistic Regression	28.75	52.53
Sum of partial scores	29.05	59.23

Table 6: Average F_1 for different ranking variants.

scores as labels. For training the classifier, we used a threshold to label an example as positive or negative. We found linear regression outperformed other ranking methods because its learning objective is to best predict the F_1 scores of the queries (see Table 6).

6.5 Qualitative Analysis

Implicit Supervision. We evaluate the quality of our training data for complex questions (CompQWeb) obtained with implicit supervision for learning the semantic matching function. We leverage the ground-truth SPARQL queries of the questions. For each question, we approximate the true labels for the partial query candidates by comparing them with the ground-truth SPARQL query. We then compare with the labels derived by our implicit supervision strategy. We found on an average, there were 4 partial query candidates that matched the ground-truth. This provides an upper bound for the number of positive examples per question. Using our approach for generating training data, we found on average 3.06 partial queries were correctly labeled as positive (true positives) and 103.08 were correctly labeled as negative (true negatives). The average number of false positives (1.72) and false negatives (0.78) were small. We find this quality is sufficient to train a semantic matching model.

Query Analysis. We compared the complex queries generated by TextRay for complex questions (CompQWeb) with the ground truth queries from the dataset to investigate if our semantic matching model generated any spurious queries that had high F_1 scores but did not capture the meaning of the input question. Specifically, we looked at the questions for which the predicted answers had F_1 scores greater than 0. Since exactly matching the string representations of the predicted and ground-truth SPARQL query is too aggressive, we compared the query components of the queries. In particular, we collect entities, relations, filter clauses (FILTER) and ordering constraints (ORDER BY) from the queries for comparison.

Given the set of query components of predicted query and ground query of a question, we compute precision, recall and F_1 scores for the query components. We found the precision was 87.02%, recall was 69.37% and F_1 was 77.19%. This reflects that the queries that achieved high F_1 scores, were indeed precise and not spurious. We examine the questions where the predicted query had no overlap with ground query as potential candidates for spurious queries. There were very few (28) such questions in the test set. Many of them used relations that were close in meaning (e.g., educational_institution.mascot vs. sports_team.team_mascot).

6.6 Error Analysis

We analyze randomly sampled 50 queries which had F_1 scores less than 0.1. We broadly classified the errors. Table 7 shows some of the failed questions. About 36% of the errors were made because incorrect entities were identified by the entity linking system. 88% of these were made when finding the first partial query. Since the complex queries are constructed using staged actions (entity identification, main relation identification etc.), errors made early in the process propagate and affect the future partial query candidates.

Not surprisingly, a large fraction of the errors (45%) was made because a wrong or ambiguous relation was scored high by the semantic matching model. It indicates that relation matching is the most crucial component of a KB-QA system. Interestingly, in 50% of the cases with relation matching errors, TextRay found a relation that had the same domain as the correct path (e.g., education.education.student vs. people.person.education – education.education.institution. Such relation paths are hard to distinguish with a semantic matching model. Also, often noisy signals (such as in example 3 in Table 7) made it difficult to correctly predict multiple main relations in the question. Lastly, we found that 19% of the errors were made when constraints or value nodes were missed.

Reason	Example
entity linking	What city was the pro athlete who began his career in 2002 born ? *The Pro Comic Book*
pred ambiguity	Which character did Armie Hammer play in the movie that included Robin Dowell ? *portrayed_in_films* vs. *film.film.starring*
wrong pred	In which movies does Tupac act in, that was edited by Malcolm Campbell ? *film.actor.film* vs. *film.editor.film*
no constraint	Which country with a population of 10,005,000 speaks Portuguese ?

Table 7: Example failed queries from CompQWeb

6.7 Future Work

There are two interesting directions to extend our work. First, while resolving multiple query components simultaneously is beneficial, the inference could be improved if the question representation reflected all prior inferences. Second, more operations such as value comparisons and aggregations could be included in the computation plan and incorporated in the semantic matching model.

7 RELATED WORK

There is a growing body of research on answering natural language questions over the knowledge bases. These can broadly be classified into three categories: retrieval-based methods, template-based methods and semantic parsing-based methods. The retrieval-based methods rely on information extraction techniques to answer natural language questions. Typically, a set of candidate answers from the KB are retrieved using relation extraction [10, 29] or distributed representations [7, 27], based on which the final answer is identified. However, these methods cannot handle highly compositional questions that require identifying multiple entities and relations.

Template-based methods [1, 8, 16, 25] map the input question into a structured query using templates. The answers are retrieved by executing the structured query over the knowledge base. Traditionally, templates were manually defined to handle compositional questions [25, 35], and suffered from limited coverage. Recent works [1] build the templates automatically from QA pairs. These answer complex questions by decomposing it into a series of binary factoid questions that can be answered using templates learned from the QA pairs. Our work is inspired by these ideas, but only uses answers to the questions for learning the semantic matching model that matches one main relation at a time.

Semantic parsing is another popular approach to answer natural questions over the knowledge bases. Conventional approaches [4, 5] relied on domain-independent meaning representations to generate

candidate logical forms that are then transformed to structured queries over the KB. Even though the meaning representations can be complex, these methods struggle to map them to correct relations in the KB for composing complex queries. More recently, with the advancements of neural network models, many encode-compare approaches [18, 30, 33] can answer questions by learning and comparing continuous representations of question and query candidates. These methods require QA pairs as training data. Obtaining this data for each subquestion of a complex question is expensive. [14, 24] have explored decomposing complex intents into multiple related simpler questions. But training these semantic parsers also requires QA pairs for each subquestion as training data.

We formulate the complex query construction as a search problem where questions with fully annotated queries are difficult to obtain. Thus, our work is broadly related to structured output prediction [21] and path finding [9, 26] methods that learn to navigate the search space from QA pairs. Compositionality is a bottleneck in these approaches [17] where the search space for finding correct paths is large and the expected reward signals sparse. They work well on simple questions where the search space is small and the search quality can be reliably approximated. We extend these ideas of learning from implicit supervision and integrate it with partial query evaluation and prior estimation to effectively prune the search space and preserve the supervision signals.

8 CONCLUSION

We have presented TextRay, a new KB-QA system that answers complex questions over a knowledge base by constructing complex queries from simpler partial queries. It integrates a novel query candidate generation strategy and a semantic matching model learned from implicit supervision to find and join partial queries efficiently. Our system outperforms previous state-of-the-art systems on highly compositional questions by a large margin of 26.82% on F_1.

ACKNOWLEDGMENTS

We would like to thank Kangqi Luo for helping us set up the baseline system. This work was supported by the UM Office of Research.

REFERENCES

[1] Abdalghani Abujabal, Mohamed Yahya, Mirek Riedewald, and Gerhard Weikum. 2017. Automated template generation for question answering over knowledge graphs. In *Proc. WWW '17*. International World Wide Web Conferences Steering Committee, 1191–1200.
[2] Junwei Bao, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. 2016. Constraint-based question answering with knowledge graph. In *Proc. COLING '16*. 2503–2514.
[3] Hannah Bast and Elmar Haussmann. 2015. More accurate question answering on freebase. In *Proc. CIKM '15*. ACM, 1431–1440.
[4] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proc. EMNLP '13*. 1533–1544.
[5] Jonathan Berant and Percy Liang. 2014. Semantic Parsing via Paraphrasing. In *Proc. ACL' 14*. ACL, 1415–1425.
[6] Kurt D. Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proc. SIGMOD '08*, Jason Tsong-Li Wang (Ed.). ACM, 1247–1250.
[7] Antoine Bordes, Jason Weston, and Nicolas Usunier. 2014. Open Question Answering with Weakly Supervised Embedding Models. In *Proc. ECML '14*, Toon Calders, Floriana Esposito, Eyke Hüllermeier, and Rosa Meo (Eds.), Vol. 8724. Springer, 165–180.
[8] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yangqiu Song, Seung-won Hwang, and Wei Wang. 2017. KBQA: learning question answering over QA corpora and knowledge bases. *Proc. VLDB '17* 10, 5 (2017), 565–576.
[9] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. 2017. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. *arXiv preprint arXiv:1711.05851* (2017).
[10] Yansong Feng, Songfang Huang, Dongyan Zhao, et al. 2016. Hybrid question answering over knowledge base and free text. In *Proc. COLING '16*. 2397–2407.
[11] Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching Machines to Read and Comprehend. In *NIPS*.
[12] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artif. Intell.* 194 (2013), 28–61.
[13] Sen Hu, Lei Zou, Jeffrey Xu Yu, Haixun Wang, and Dongyan Zhao. 2018. Answering Natural Language Questions by Subgraph Matching over Knowledge Graphs. *IEEE Trans. Knowl. Data Eng.* 30, 5 (2018), 824–837.
[14] Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based Neural Structured Learning for Sequential Question Answering. In *Proc. ACL '17*. 1821–1831.
[15] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.
[16] Fei Li and HV Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proc. VLDB '14* 8, 1 (2014), 73–84.
[17] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. 2016. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. *arXiv preprint arXiv:1611.00020* (2016).
[18] Kangqi Luo, Fengli Lin, Xusheng Luo, and Kenny Q. Zhu. 2018. Knowledge Base Question Answering via Encoding of Complex Query Graphs. In *Proc. EMNLP '18*. 2185–2194.
[19] Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Proc. EMNLP '15*. 1412–1421.
[20] Haoruo Peng, Ming-Wei Chang, and Wen-tau Yih. 2017. Maximum Margin Reward Networks for Learning from Explicit and Implicit Supervision. In *Proc. EMNLP '17*. 2368–2378.
[21] Haoruo Peng, Ming-Wei Chang, and Wen-tau Yih. 2017. Maximum Margin Reward Networks for Learning from Explicit and Implicit Supervision. In *Proc. EMNLP '17*. 2368–2378.
[22] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proc. EMNLP '14*. 1532–1543.
[23] Linfeng Song, Zhiguo Wang, Mo Yu, Yue Zhang, Radu Florian, and Daniel Gildea. 2018. Exploring graph-structured passage representation for multi-hop reading comprehension with graph neural networks. *arXiv preprint arXiv:1809.02040* (2018).
[24] Alon Talmor and Jonathan Berant. 2018. The Web as a Knowledge-Base for Answering Complex Questions. In *Proc. NAACL '18*. 641–651.
[25] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. 2012. Template-based question answering over RDF data. In *Proc. WWW '12*. ACM, 639–648.
[26] Wenhan Xiong, Thien Hoang, and William Yang Wang. 2017. DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning. In *Proc. EMNLP '17*. 564–573.
[27] Kun Xu, Siva Reddy, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2016. Question Answering on Freebase via Relation Extraction and Textual Evidence. In *Proc. ACL '16*.
[28] Yi Yang and Ming-Wei Chang. 2015. S-MART: Novel Tree-based Structured Learning Algorithms Applied to Tweet Entity Linking. In *Proc. ACL'15*. 504–513.
[29] Xuchen Yao and Benjamin Van Durme. 2014. Information Extraction over Structured Data: Question Answering with Freebase.. In *Proc. ACL '14*. ACL, 956–966.
[30] Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base. In *Proc. ACL '15*. 1321–1331.
[31] Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Proc. ACL '16*, Vol. 2. 201–206.
[32] Pengcheng Yin, Nan Duan, Ben Kao, Junwei Bao, and Ming Zhou. 2015. Answering questions with complex semantic constraints on open knowledge bases. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 1301–1310.
[33] Mo Yu, Wenpeng Yin, Kazi Saidul Hasan, Cicero dos Santos, Bing Xiang, and Bowen Zhou. 2017. Improved neural relation detection for knowledge base question answering. *arXiv preprint arXiv:1704.06194* (2017).
[34] Weiguo Zheng, Jeffrey Xu Yu, Lei Zou, and Hong Cheng. 2018. Question answering over knowledge graphs: question understanding via template decomposition. *Proc. VLDB '18* 11, 11 (2018), 1373–1386.
[35] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. 2014. Natural language question answering over RDF: a graph data driven approach. In *Proc. SIGMOD '14*. ACM, 313–324.