Automated Template Generation for Question Answering over Knowledge Graphs

Abdalghani Abujabal
Max Planck Institute for Informatics
Saarland Informatics Campus, Germany
abujabal@mpi-inf.mpg.de

Mohamed Yahya
Max Planck Institute for Informatics
Saarland Informatics Campus, Germany
myahya@mpi-inf.mpg.de

Mirek Riedewald
Northeastern University, USA
mirek@ccs.neu.edu

Gerhard Weikum
Max Planck Institute for Informatics
Saarland Informatics Campus, Germany
weikum@mpi-inf.mpg.de

ABSTRACT

Templates are an important asset for question answering over knowledge graphs, simplifying the semantic parsing of input utterances and generating structured queries for interpretable answers. State-of-the-art methods rely on hand-crafted templates with limited coverage. This paper presents QUINT, a system that automatically learns utterance-query templates solely from user questions paired with their answers. Additionally, QUINT is able to harness language compositionality for answering complex questions without having any templates for the entire question. Experiments with different benchmarks demonstrate the high quality of QUINT.

Keywords

Question Answering; Semantic Parsing; Knowledge Graphs

1. INTRODUCTION

Motivation. Templates play an important role in question answering (QA) over knowledge graphs (KGs), where user utterances are translated to structured queries via semantic parsing [4, 35, 41]. Utterance templates are usually paired with query templates, guiding the mapping of utterance constituents onto query components.

Table 1 shows two utterance-query templates used by Fader et al. [13]. Each template i) specifies how to chunk an utterance into phrases, ii) guides how these phrases map to KG primitives by specifying their semantic roles as predicates or entities, and iii) aligns syntactic structure in the utterance to the semantic predicate-argument structure of the query.

A benefit of templates is that the mappings to the KG are traceable and can be leveraged to generate explanations for the user to understand why she receives specific answers. For example, when the utterance *"Who invented the Internet?"* returns the answer Al Gore (known for funding the Internet), an explanation might tell us that it was answered from the first template of Table 1, and that the KG predicate used was `knownFor`, originating from *'invented'* in the utterance through the $pred_1$ alignment.

State of the Art and its Limitations. Many prior approaches to QA over KGs rely on hand-crafted templates or rules, which inevitably results in limited coverage. Some methods use utterance-query templates with alignments as in Table 1 [13, 23, 28, 35, 41, 48]. Bast and Haussmann [3] rely on three manually constructed query templates without any utterance templates. Instead, they exhaustively instantiate each query template from the utterance. Yih et al. [47] define a sequence of stages for generating queries, each relying on a set of manually defined rules for how query conditions are added. Berant et al. [4] rely on a manually specified grammar for semantic parsing and mapping text spans onto KG primitives.

Our method supports complex questions that require multiple KG predicates to obtain an answer. For example, the question *"Which were the alma maters of the PR managers of Hillary Clinton?"* is answered by a chain of KG predicates. Our system, called QUINT, is able to exploit natural language compositionality and templates learned from simple questions to answer complex questions without any templates that capture the complete question.

Approach and Contribution. Our approach is to automatically learn utterance-query templates with alignments between the constituents of the question utterance and the KG query (Section 3). These templates are learned by distant supervision, solely from questions paired with their answer sets as training data [19, 20]. Prior work used the same input for training QA systems over KGs, but relying on hand-crafted templates [3, 47] or hand-crafted rules in various formalisms such as DCS [4] to generate the structure of KG queries. In contrast, our automatically learned utterance templates are based on standard dependency parse trees, thus benefiting from methodological progress and tools on mainstream parsing [11]. Dependency parse representations also enable us to cope with compositional utterances. On the KG query side, templates are expressed in a SPARQL-style triple-pattern language. Our templates include automatic support for semantic answer typing, an impor-

Table 1: Curated templates by Fader et al. [13]. Shared $pred$ and ent annotations indicate an alignment between a phrase in the utterance template and a KG semantic item in the corresponding utterance template.

#	Utterance Template	Query Template
1	Who VP_{pred_1} NP_{ent_1}	(?x, $pred_1$, ent_1)
2	What NP_{pred_1} is NP_{ent_1}	(ent_1, $pred_1$, ?x)

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052583

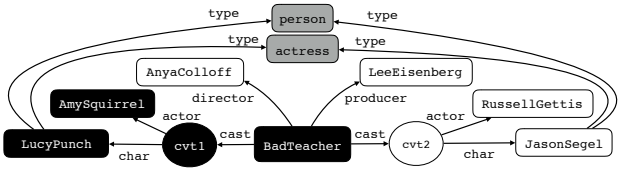


Figure 1: Example KG fragment.

tant aspect where prior work relied on manually specified typing patterns [3, 4]. The alignments between utterance and query are computed by integer linear programming.

Template learning is an offline step. Online (Section 4), when the user interacts with the system, QUINT performs light-weight template matching. QUINT, uses basic templates to answer structurally complex compositional questions without observing such questions during training. QUINT accomplishes this by i) automatically decomposing the question into its constituent clauses, ii) computing answers for each constituent using our templates, and iii) combining these answers to obtain the final answer.

Our salient contributions are: i) automatically learning role-aligned question-query templates, ii) handling compositional complex questions, and iii) experiments with the WebQuestions and Free917 benchmarks and with complex questions.

2. SYSTEM OVERVIEW

2.1 Knowledge Graphs

Figure 1 shows a KG fragment. Nodes are entities $e \in E$ (e.g., `LucyPunch`), types $c \in C$, also known as classes (e.g., `actress`) or literals $t \in T$ (e.g., dates). Nodes are connected by edges labeled with predicates $p \in P$ (e.g., `cast`). Two nodes connected by an edge form a fact (e.g., `LucyPunch type actress`). A KG is a set of facts, which also form a graph (hence the name). Additionally, artificial Compound Value Type (CVT) nodes are used to represent n-ary relations by connecting the different entities and literals participating in such a relation. For example, `LucyPunch`'s role in `BadTeacher` as `AmySquirrel`, in Figure 1, requires a CVT node `cvt1`. We deem a pair of predicates pointing to/from a CVT node to be a predicate as well (e.g., `cast.char`). Any $s \in S = E \cup C \cup P$ is called a *semantic item*.

To query a KG we use a graph pattern-matching language based on SPARQL. A triple pattern is a fact with one or more components replaced by variables (e.g., `?x type actress`). A query q is a set of triple patterns. Figure 4 shows a graphical representation of an example query. The variable `?x` is designated as a *projection variable*. An answer a to a query q over a KG is an entity such that there is a mapping of the query variables to the semantic items of the KG where the projection variable maps to a and the application of the mapping to the query results in a subgraph of the KG.

2.2 Lexicons

To connect utterance vocabulary to semantic items in the KG, we require a lexicon $\mathcal{L}$. Our lexicon consists of a predicate lexicon $\mathcal{L}_P$ and a type lexicon $\mathcal{L}_C$. Hakimov et al. [15] addressed the lexical gap between the vocabulary used by users and the KG semantic items and showed that when high quality lexicons are used the performance of QA systems substantially improves. We construct $\mathcal{L}_P$ and $\mathcal{L}_C$ using distant supervision [24], similar to the QA work of Bast and Haussmann [3], Berant at al. [4], and Yih et al. [47]. We utilize ClueWeb09-FACC1, a corpus of 500M Web pages annotated with Freebase entities.

To create $\mathcal{L}_P$, we run the following extraction pattern over the corpus: *"e_1 r e_2"*, where e_1 and e_2 are entities and r is a phrase. For example, *"[[Albert Einstein | `AlbertEinstein`]] was born in [[Ulm | `Ulm`]] ..."* Following the distant supervision assumption, if $(e_1\ p\ e_2)$ is a triple in the KG, then r expresses p and we add $r \mapsto p$ to $\mathcal{L}_P$. For the KG triple (`AlbertEinstein birthPlace Ulm`) we add *"was born in"* $\mapsto$ `birthPlace` to $\mathcal{L}_P$. Of course, this assumption will not always hold, so we assign the mapping a weight w proportional to how many times it was observed in ClueWeb09-FACC1. In Section 3.3, we present how we extract $\mathcal{L}_{P_{train}} \subseteq \mathcal{L}_P$ at training time, which we give higher precedence over the full $\mathcal{L}_P$ at testing time.

For $\mathcal{L}_C$, we run Hearst patterns [16] over the annotated corpus, where one argument is an entity and the other is a noun phrase. For example, for *"e and other np"*, we add to $\mathcal{L}_C$ the entry $np \mapsto c$ for each c such that $(e$ `type` $c) \in KG$. For example, given the sentence *"[[Albert Einstein | `AlbertEinstein`]] and other scientists ..."* we add, for each type c to which the entity `AlbertEinstein` belongs, the entry *"scientist"* $\mapsto c$ to our $\mathcal{L}_C$. Again, entries are assigned a weight w proportional to their frequencies in ClueWeb09-FACC1. Table 2 shows entries from our $\mathcal{L}_P$ and $\mathcal{L}_C$.

For entities, we use an off-the-shelf named entity recognition and disambiguation (NER/NED) system. We use the S-MART [44] system which is designed for short informal texts such as tweets. It was used in the QA pipeline of Yih et al. [47]. At testing time, we found that we get the best results when we use our templates for named entity recognition and then take the n best entity candidates from S-MART for the recognized mentions and let our query ranking scheme do the final disambiguation (see Section 5).

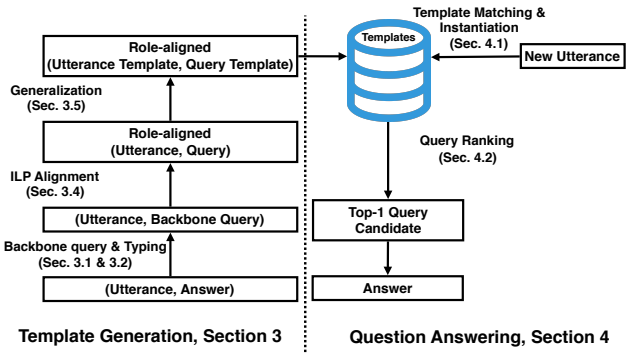


Figure 2: An outline of how QUINT generates role-aligned templates at training time and how it uses them for question answering.

2.3 Templates

Figure 2 outlines our system for learning and subsequently using templates to answer natural language questions over knowledge graphs. Templates are generated at training time (Figure 2, left), and are used at testing (answering) time (Figure 2, right) for answering questions. The input to the training stage is pairs of question *utterance* u and its *answer set* A_u from the KG. An example of a training utterance is $u =$*"Which actress played character Amy Squirrel on Bad Teacher?"* [sic], which is paired with the answer set $A_u = \{$`LucyPunch`$\}$. We use the letter u to refer to both the utterance and its dependency parse tree interchangeably. A *dependency parse tree* [18] of an utterance is a directed rooted tree whose nodes correspond to utterance tokens and edges represent grammatical relations between the nodes. Our utterance templates

are based on the dependency parse of utterances. Our motivation is that a dependency parse (1) can capture long range dependencies between the tokens of an utterance which helps when answering compositional questions (Section 4.3) and (2) gives great flexibility allowing QUINT to skip irrelevant tokens when instantiating query templates (Section 4.1). Figure 3 shows the dependency parse of the above utterance.

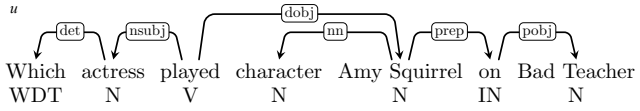


Figure 3: The dependency parse tree of our example utterance (top) together with the answer set (bottom).

QUINT starts by heuristically constructing a KG query q that captures u. It constructs a backbone query $\hat{q}$ for q by finding for each $a \in A_u$ the smallest subgraph connecting the KG entities detected in u and a (Section 3.1). For our example in Figure 3, this is the subgraph connecting the black nodes in Figure 1. QUINT forms $\hat{q}$ from this subgraph by replacing the nodes corresponding to a or a CVT with variables (Figure 4). To account for types, it extends $\hat{q}$ by adding the type constraints connected to a to the corresponding variable (Section 3.2) – the gray nodes in Figure 1. Figure 5 shows the resulting $\hat{q}$.

Next, QUINT aligns $\hat{q}$ with u, which gives us $q \subseteq \hat{q}$ that best captures u, a chunking of u, and an alignment between the constituents of u and q. The alignment will be carried over to the templates created from u and q. We formulate the alignment problem as a constrained optimization and find the best alignment m using integer linear programming (ILP) (Section 3.3). Figure 7 shows u, the resulting q, and alignment m indicated with shared ent, $pred$, and $type$ annotations between nodes in u and q, which also specify the semantic role for this alignment.

Next, QUINT performs generalization to generate a template from a concrete pair of aligned utterance dependency parse tree and query graph (Section 3.4). It removes the concrete text in the nodes participating in m and similarly for the semantic items in q, keeping the annotations ent, $pred$, and $type$, thereby turning these nodes into placeholders (Figure 8). The result is template $t = (u_t, q_t, m_t)$ composed of an utterance template u_t, a query template q_t, and an alignment m_t between the two.

When the user issues a new question, QUINT matches its dependency parse tree against the template library created during training (Section 4.1). In Figure 9 the bold edges and nodes in u' are those matched by u_t (Figure 8). For each match, the corresponding query template q_t is instantiated using the alignment m_t and the lexicon. Figure 9 also shows a query resulting from such an instantiation. Finally, QUINT ranks these candidate queries using a learning-to-rank approach (Section 4.2). The answers of the top-ranked query are returned to the user. By showing which template was used and how it was instantiated, QUINT can explain answers to the user.

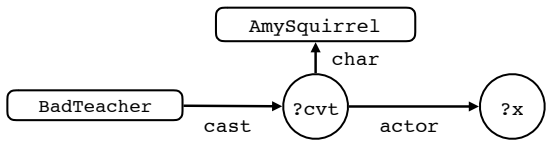


Figure 4: Backbone query $\hat{q}$.

3. TEMPLATE GENERATION

We now present the details of the offline template generation stage. The input to this stage is pairs of utterance u and its answer set A_u as in Figure 3.

3.1 Backbone Query Construction

To generate the backbone query $\hat{q}$, QUINT starts by annotating a training utterance u with the named entities it contains and disambiguating these to Freebase entities using an off-the-shelf named entity recognition and disambiguation (NER/NED) system [44]. For our example, the resulting annotated question is:

"Which actress played character [[*Amy Squirrel* | `AmySquirrel`]] *on* [[*Bad Teacher* | `BadTeacher`]]*?"*

Next, for each answer $a \in A_u$, QUINT finds the smallest connected subgraph of the KG that contains the above entities found in the question as well as a. For our example and the KG of Figure 1, this is the subgraph with black nodes. To this end, starting with an entity found in the question, QUINT explores all paths of length two when the middle node is a CVT node and paths of length one otherwise, to restrict the search space, similarly to [47]. If the middle node is a CVT node and the question contains multiple entities, QUINT explores paths of length one connecting the CVT node with these entities. We assume that this subgraph captures the meaning of the question and connects it to one of its answers a. There may be multiple such graphs. QUINT then transforms this subgraph into a query by replacing a with the variable `?x` and any CVT nodes with distinct variables (e.g., `?cvt1`). The above procedure is performed for each $a \in A_u$ for a given u, resulting in multiple queries. We keep the query with the highest F1 with respect to A_u. Figure 4 shows $\hat{q}$ for our example.

3.2 Capturing Answer Types

Capturing the answer types given in the question is important for precision. In the question *"Which **actor** died in New York?"*, $\hat{q}$ generated thus far would be `{?x deathPlace NewYork}`, which does not capture the fact that the question is only interested in actors, hurting precision. Walker et al. [39] showed that identifying the expected answer type of an utterance boosts the performance of QA systems. This conclusion is proven in our experiments as well (Section 5). Earlier QA systems either use manual rules to find phrases in the question that evoke one of a number of predefined set of possible types in the query [3, 4], or neglect type constraints altogether [47]. QUINT automatically creates templates that capture which phrases in the question evoke types in the query, and uses the full Freebase type system as potential mapping targets.

Starting with $\hat{q}$ generated thus far (Figure 4), QUINT connects to the answer variable node in $\hat{q}$ one *type constraint* for each $c \in C$ such that the variable originates from the answer entity $a \in A_u$ and $(a$ `type` $c) \in KG$. In the KG of Figure 1, `LucyPunch` $\in A_u$ has the types `person` and `actress`, the resulting $\hat{q}$ is shown in Figure 5. $\hat{q}$ now contains more type constraints than actually given in the question. In the next section we show how the correct one is determined as part of the alignment step.

3.3 Utterance-Query Alignment

With $(u, \hat{q})$ pairs at hand, QUINT proceeds to aligning the constituents of the two. The alignment i) gives us a chunking of u into phrases that map to semantic items in $\hat{q}$, ii) removes spurious type constraints from $\hat{q}$, resulting in $q \subseteq \hat{q}$, and iii) gives us an alignment m between the constituents of u and q.

Alignment is driven by our lexicons $\mathcal{L}_P$ and $\mathcal{L}_C$ (Section 2.2), but faces inherent ambiguity, either from truly ambiguous phrases

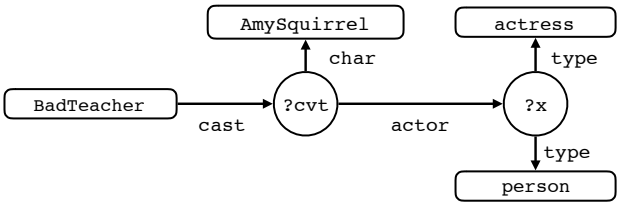


Figure 5: Backbone query $\hat{q}$ with types.

or from inevitable noise in the automatically constructed lexicons. We model the resolution of this ambiguity as a constrained optimization and use an ILP to address it. We start by building a bipartite graph with Ph, the set of all phrases from u, on one side and $S_{\hat{q}}$, the set of semantic items in $\hat{q}$, on the other as shown in Figure 6. $Ph = \{ph_1, ph_2, ...\}$ is generated by taking all subsequences of tokens in u. We add an edge between each $ph_i \in Ph$ and $s_j \in S_{\hat{q}}$ where $(ph_i \mapsto s_j) \in \mathcal{L}_P \cup \mathcal{L}_C$ with a weight w_{ij} from the lexicon. Table 2 shows a fragment of our lexicons.

Additionally, we add an edge connecting each entity in $\hat{q}$ to the phrase that evokes it in u. Entities are added to prevent their surface forms in the question from mapping to a class or a predicate. We use an off-the-shelf NER/NED system to identify entities in u (Section 3.1). To resolve the ambiguity in the mapping of types and predicates and obtain the intended alignment, the ILP decides which subset of the edges we need to keep.

We extend our notation before presenting our ILP. For semantic item s_j, E_j, C_j and P_j are 0/1 constants indicating whether s_j is an entity, type, or predicate, respectively. X_{ij} is a 0/1 decision variable whose value is determined by the solution of the ILP. The edge connecting ph_i to s_j in the bipartite graph is retained iff $X_{ij} = 1$. Given a set of types connected to a variable v from which we want to pick at most one, this set of types is $S(v) = \{c_1, c_2, ...\}$ ({`actress`,`person`} in our example) and the set of phrases that can map to types in $S(v)$ is $Ph(v)$.

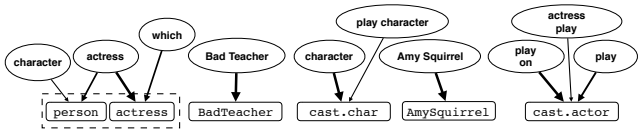


Figure 6: Bipartite graph provided to the ILP, with phrases at the top and semantic items at the bottom. An edge corresponds to a mapping from a phrase to a semantic item, and its thickness corresponds to the mapping weight. Dashed box represents $S($`?x`$)$ for `?x` in Figure 5.

The objective of the ILP is to maximize the total weight of the mapped phrases: $\sum_{i,j} w_{ij} X_{ij}$, where w_{ij} comes from the lexicon. The constraints make sure that the resulting alignment is a meaningful one:

1. Each semantic item is obtained from at most one phrase: $\forall s_j \in S_{\hat{q}} : \sum_{i,j} X_{ij} \leq 1$. In Figure 6, this means that `cast.actor` comes either from *'play on'* or *'play'*.

2. A token contributing to an entity phrase cannot contribute to any other phrase: $\forall ph_i \in Ph, ph_{i'} \in Ph(t), t \in ph_i, s_j, s_{j'} \in S_{\hat{q}} : X_{ij} + E_j \leq \sum_{i',j'} X_{i'j'} + 1$. In Figure 6, this means that the tokens of *'Amy Squirrel'* cannot be part of a mention of a semantic item other than `AmySquirrel` (if there was such a candidate).

Table 2: Fragment of our lexicons: $\mathcal{L}_P$ and $\mathcal{L}_C$.

Ph.	**S. Item**	w	**Ph.**	**S. Item**	w
"play on"	`cast.actor`	0.5	"role"	`cast.char`	0.6
"play"	`cast.actor`	0.3	"actress"	`actress`	0.7
"character"	`cast.char`	0.6	"actress"	`person`	0.3

3. For each variable v, at most one phrase in $Ph(v)$ can map to at most one type in $S(v)$: $\forall v, ph_i \in Ph(v), c_j \in S(v) : \sum_{i,j} X_{ij} \leq 1$. In Figure 6, this means that either `person` or `actress` will be chosen, and only one of the phrases mapping to the chosen type among these.

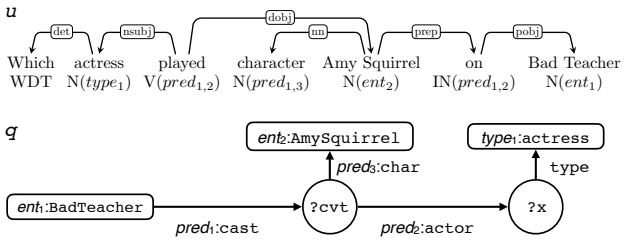


Figure 7: Aligned utterance query pair (u, q, m). m is indicated by shared ent, $pred$, and $type$ annotations (e.g., *"played on"* is aligned with `cast.actor`).

We solve the ILP using Gurobi to obtain the intended alignment m, indicated by shared ent, $type$, and $pred$ semantic annotations between u and $\hat{q}$. Additionally, by discarding all type constraints not connected to a phrase in m (`?x type person` in our example), we obtain q from $\hat{q}$. Figure 7 shows the utterance from our running example aligned with q (contrast with $\hat{q}$ in Figure 5).

An important by-product of alignment at training time is a lexicon $\mathcal{L}_{P_{train}} \subseteq \mathcal{L}_P$, composed of the phrase-predicate alignments observed during training. This lexicon is much cleaner than the noisier $\mathcal{L}_P$. We will use both at testing time, giving precedence to mappings obtained from $\mathcal{L}_{P_{train}}$ when they exist.

3.4 Generalization to Templates

Next, QUINT constructs templates from aligned utterance-query pairs (u, q, m) obtained from the alignment process above. On the utterance side, QUINT takes the utterance u represented using its dependency parse tree and restricts it to the smallest connected subgraph that contains the tokens of all phrases participating in m. In Figure 7 this results in removing the node corresponding to *'which'*. To create a template from this subgraph, we turn the nodes participating in m into placeholders by removing their text and keeping the POS tags and semantic alignment annotations (ent, $type$, $pred$). We use universal POS tags [25] for stronger generalization power. We replace compound nouns with a noun token that can be used to match compound nouns at testing time to ensure generalization. At testing time, our templates allow for robust chunking of an incoming question into phrases corresponding to entities (i.e. as named entity recognizers), predicates (i.e. as relation extractors) and types (i.e. as noun phrase chunkers). For NER, we show that using our templates at testing time gives superior results when compared to using an off-the-shelf NER system.

On the query side, we take the query and remove the concrete labels of edges (predicates) and nodes (entities and types) participating in m, keeping the semantic alignment annotations. We use

the number of utterance-query pairs which generate a template as a signal in query ranking (Section 4).

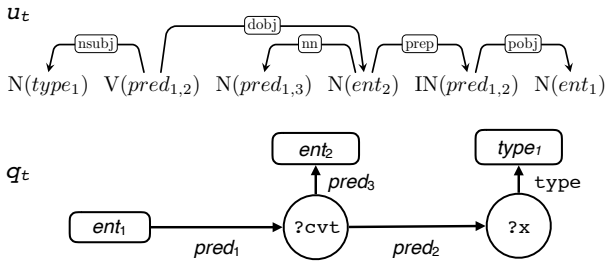


Figure 8: A template $t = (u_t, q_t, m_t)$. m_t is indicated by shared ent, $pred$, and $type$ annotations. Figure 7 shows the concrete utterance-query pair used to generate this template.

4. QUESTION ANSWERING WITH TEMPLATES

4.1 Candidate Query Generation

At answering time, when a user poses a new utterance u', QUINT matches it against all templates in our repository. u' matches a template (u_t, q_t, m_t) if a subgraph of its dependency parse tree is isomorphic to u_t considering their edge labels and the POS tags in their nodes. Figure 9 shows an example of u' whose dependency parse matches the utterance template u_t in Figure 8. Bold edges and nodes are the ones participating in the isomorphism with u_t. Note how using a dependency parse to represent an utterance allows us to ignore the token *'popular'* in u', which does not carry any semantics that can be captured by our specific KG. If the question asked, say, for an *'American'* actress instead, then another template learned at training time would allow us to instantiate a query that captures this important constraint. Our ranking scheme described below decides which is the best match.

For each matching utterance template (usually several), QUINT instantiates the corresponding query template q_t based on the alignment m_t and the lexicon $\mathcal{L}$. Lexicon lookups are guided by the semantic alignment annotations in the query template, which restrict specific parts of the query to types, entities, or predicates. For predicates, QUINT first performs a lookup in the cleaner $\mathcal{L}_{P_{train}}$ created at training time (Section 3.3). If no results are returned, QUINT queries the full predicate lexicon $\mathcal{L}_P$ (Section 2.2). Figure 9 bottom shows a query q' obtained from instantiating the template of Figure 8 based on u' shown at the top of the same figure.

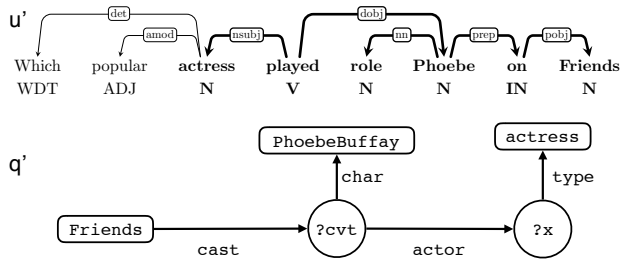


Figure 9: Template instantiation (using t in Figure 8).

Table 3: Query candidate ranking features.

Category	#	Description
Alignment	1,2	Average & sum of lexicon weights for predicates
	3,4	Average & sum of lexicon weights for entities
	5	# of utterance tokens literally matching their predicate
	6	# of entity mentions matching their canonical name in the KG
	7	Indicator: question n-gram $\wedge$ predicate p, $n = 1, 2$ and 3
Semantic	8,9	Average & sum of entity popularity scores
	10	# of predicates in the query
	11	# of entities in the query
Template	12	Indicator: t captures a type constraint
	13	# of training utterances that generate t
	14	t coverage: % of tokens in utterance matched by u_t
	15	Indicator: utterance node with $type$ annotation $\wedge$ semantic answer type (if t is typed)
Answer	16	Indicator: answer set size (=1, =2, =3, $\in$ [4-10], $>$ 10)

4.2 Query Ranking

Query generation yields multiple candidate queries for an utterance, either due to multiple matching templates as discussed above or due to ambiguity of phrases in the lexicon. We adopt a learning-to-rank approach, analogously to Bast and Haussmann [3], to rank the obtained queries, and return the highest ranking query as the one intended by the question. We use a random forest classifier to learn a preference function between a pair of queries [9]. Table 3 lists the features used for the feature vector $\phi(u', t, q')$ associated with the instantiation of template t from utterance u' resulting in query q', which we discuss below. For instantiations of a pair of templates t_1 and t_2 matching utterance u' and resulting in the queries q_1 and q_2, respectively, the feature vector used is a result of concatenating $\phi(u', t_1, q_1')$, $\phi(u', t_2, q_2')$, and $\phi(u', t_1, q_1') - \phi(u', t_2, q_2')$.

We compute four types of features as shown in Table 3. *Alignment* features measure the association between utterance tokens and KG semantic items. *Semantic* features consider the query exclusively. *Template* features measure the appropriateness of a template t for a given an utterance. The *answer* feature template indicates which of the predefined ranges the query answer set size belongs to. Answer size is a good cue as empty answer sets or very large ones are indicators of incorrect queries that are over or under-constrained. Features 7, 15, and 16 define feature templates that are instantiated based on the training data. For example, instantiations of feature 7 are pairs of utterance n-gram and query predicate. Indicator features take boolean values. For instance, from the utterance and the query in Figure 9 we generate the feature "*actress play* $\wedge$ *cast.actor*" with value 1.

4.3 Answering Complex Questions

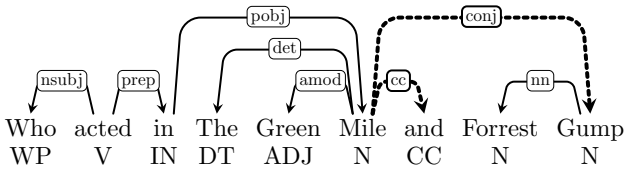
The class of complex questions we target are those composed of multiple clauses, each centered around a relationship, which collectively describe a single "variable" (with possibly multiple bindings in the KG). For example, the question *"actors starring in The Departed who were born in Cambridge, MA"* is composed of two clauses: *"actors starring in The Departed"* (e.g., `MattDamon`, `MarkWahlberg`, `JackNicholson`) and the relative clause *"actors who were born in Cambridge, MA"* (e.g., `MattDamon`, `UmaThurman`, `MarkWahlberg`). `MattDamon` and `MarkWahlberg` are answers to the complete question.

Handling complex questions is a natural extension of the procedure in Section 4.1. We do this in three steps: i) automated depen-

dency parse rewriting when necessary, ii) sub-question answering, and iii) answer *stitching*.

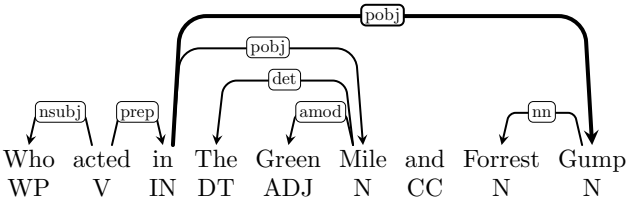
Automated dependency parse rewriting. The need for rewriting arises when we have complex questions that we are unable to fully capture with our template repository. An example is *"Who acted in The Green Mile and Forrest Gump?"*. For this question, our templates trained on the simpler WebQuestions dataset, would be unable to capture the second constraint, expressed through a coordinating conjunction (using *'and'*). The problem can be seen in Figure 10(a), where there is no direct connection between *'Forst Gump'* and the relation phrase *'acted in'*, which our templates would expect given the data used to learn them. To overcome this, we perform simple automated rewriting to get two separate interrogative propositions following Del Corro and Gemulla [11] for relative clauses and coordinating conjunctions. In our concrete example, this is done by connecting *'Gump'*, the target of the *conj* edge, to *'in'*, the head of *'Mile'* which is the source of the *conj* edge. We give the new edge the label *pobj*, the same as the one connecting the head of *conj* to its parent. The *conj* and *cc* edges are subsequently removed. The resulting dependency graph and sub-questions captured by our templates are shown in Figure 10(b).

More generally, a rewriting is triggered if i) we detect a coordinating conjunction or relative clause [11] and ii) matches against our template repository result in less sub-questions than expected (e.g., a single coordinating conjunction or relative clause should result in two matched sub-questions). While the question above requires rewriting, the question *"What film directed by Steven Spielberg did Oprah Winfrey act in?"*, where both relations are spelled out, requires no rewriting by QUINT.



"Who acted in The Green Mile and Forrest Gump"

(a)



"Who acted in The Green Mile"
"Who acted in Forrest Gump"

(b)

Figure 10: (a) shows the original dependency parse of the question before rewriting (b) shows the dependency parse after rewriting i.e., dropping dashed edges in (a) and adding a new edge (bold) connecting *"Gump"* to *"in"*.

Sub-question answering. We match our templates against the complete automatically rewritten (if needed) dependency parse . Each match corresponds to a sub-question that can be answered independently. Here, we follow the procedure described in Sections 4.1 and 4.2 for each question independently, and keep the ranked list of queries returned for each subquestion.

Stitching. For each sub-question detected and mapped to a list of queries, we assign a query in its ranked list of queries (Section 4.2) a score of $\frac{1}{r}$, where r is the rank of the query within the list. The idea here is to assign a higher numerical score to a higher ranked query. We return the answers resulting from the combination of queries, one for each sub-question, such that the intersection of their answer sets is non-empty and the sum of their scores is highest.

5. EXPERIMENTS

We evaluate QUINT on the WebQuestions and Free917 datasets, as well as a newly introduced set of complex questions. These experiments serve two goals: (1) to show that automatically learned templates can deliver high-quality results, and (2) that our templates provide a natural path toward handling complex questions.

5.1 Setup

Benchmarks. We compare QUINT to previous work using the following benchmarks over Freebase:

- **WebQuestions** [4], which consists of 3778 training questions and 2032 test questions, each paired with its answer set, collected using the Google Suggest API and crowdsourcing.
- **Free917** [10], which consists of 917 questions manually annotated with their Freebase query; 641 training and 276 test questions. For Free917, we made use of a manually crafted entity lexicon provided by the designers of the benchmark for entity linking. All other systems in Table 4 used this lexicon. We used the standard train/test splits for WebQuestions and Free917.
- **ComplexQuestions.** This benchmark is composed of 150 test questions that exhibit compositionality through multiple clauses, e.g., *"What river flows through India as well as China?"* Crucially, ComplexQuestions contains *no* training questions: its purpose is to demonstrate that our template-based approach, while trained only on the simpler single-clause WebQuestions, can handle more complex questions. We constructed this benchmark using the crawl of WikiAnswers (`http://wiki.answers.com`), a large, community-authored corpus of natural language questions, collected by Fader et al. [13]. We asked a human annotator to collect questions with multiple clauses and also to provide the gold standard answer set for them from Freebase.

Evaluation metrics. We use the evaluation metric adopted by each of the benchmarks. For WebQuestions this is the average F1 score across all test questions. We also adopt F1 for evaluating systems on ComplexQuestions. For Free917, the metric is accuracy, defined as the fraction of questions answered perfectly i.e., with the exact gold standard answer set.

Template Generation. We analyze our templates generated at training time. For WebQuestions, QUINT generates $\hat{q}$ queries (Sections 3.1, 3.2) for 3555 of the 3778 training questions. For the others, either no entity candidates were identified by the NER/NED system or no subgraphs connecting the identified entities were found. The ILP successfully aligned (Section 3.3) 3003 of the 3555 u–$\hat{q}$ pairs, resulting in 3003 (u, q, m) triples. The others could not be aligned due to lexicon misses. These produced 1296 distinct (u_t, q_t, m_t) templates. We use these templates to answer test questions in WebQuestions and ComplexQuestions later on.

For Free917, QUINT generates $\hat{q}$ queries for 602 of the 641 training questions. For the others, no subgraphs connecting the identified entities were found. The ILP successfully aligned 571 of the 602 u–$\hat{q}$ pairs, resulting in 571 (u, q, m) triples. These produced 284 distinct (u_t, q_t, m_t) templates.

Method	WebQuestions Average $F1$	Free917 Accuracy
Cai and Yates [10] (2013)	-	59.0
Berant et al. [4] (2013)	35.7	62.0
Kwiatkowski et al. [19] (2013)	-	68.0
Yao and Van Durme [46] (2014)	33.0	-
Berant and Liang [5] (2014)	39.9	68.6
Bao et al. [2](2014)	37.5	-
Bordes et al. [8] (2014)	39.2	-
Yao [45] (2015)	44.3	-
Dong et al. [12] (2015)	40.8	-
Bast and Haussmann [3] (2015)	49.4	76.4
Berant and Liang [21] (2015)	49.7	-
Yih et al. [47] (2015)	**52.5**	-
Reddy et al. [28] (2016)	50.3	78.0
This Work		
QUINT-untyped	50.8	**78.6**
QUINT	51.0	72.8

Table 4: Results on the WebQuestions and Free917 test sets.

5.2 Results on WebQuestions and Free917

Question Answering Performance. Table 4 shows the results on the test sets for WebQuestions and Free917 with additional entries for earlier work on these benchmarks. QUINT uses the templates generated by our full system described in Sections 3 and 4. QUINT-untyped uses templates where we skip the step described in Section 3.2 that allows the capturing of answer types.

On WebQuestions, QUINT outperforms existing approaches, while performing slightly below the system of Yih et al. [47]. However, the difference in $F1$ scores between our results and that of Yih et al. is not statistically significant using a two-sided paired t-test at 95% confidence level. Recent work has looked at a different setup combining KGs with additional textual resources for answering questions. For example, Kun Xu et al. [40] and Savenkov et al. [29] use Wikipedia and Web search results combined with community question answering data, respectively. These systems achieve slightly higher F1 scores than our system with these resources (53.3 and 52.2, respectively), but the performance drops without these (47.1 and 49.4, respectively). QUINT uses entity-annotated text corpora for creating lexicons offline, but following the original benchmark setup, does not invoke any other resources at answering time.

On Free917, QUINT-untyped outperforms all other methods and obtains the best result to date. Interestingly, QUINT-untyped outperforms QUINT on this benchmark. We comment on this below.

Templates as Named Entity Recognizers. In the above experiment, we used our templates for named entity recognition, and relied on the S-MART NER/NED system for generating the top entity candidates for each recognized entity mention, with the final entity resolution being done by our ranking stage. On average, S-MART generated 2 entity candidates per mention. We tried fully relying on S-MART, by adopting its NER annotations as well. In this case, we restricted matches to those templates compatible with S-MART's annotations (i.e., their ent annotations agree with the entity annotations produced by S-MART). In this case, our F1 score drops to 49.0 as the number of questions with no matching templates increases. This shows the value of our templates as entity recognizers in questions.

Effect of Typing. In our experiments, typing had a positive impact on the results for WebQuestions, and a negative one on Free917. In both cases, this shows the importance of typing. For Free917, a large portion of the test questions ask for literals such as dates or monetary values. However, our type system does not capture these in a fine grained manner, limiting our ability to distinguish between, say monetary values and sizes. This is something to explore in future work. For WebQuestions, our type system comprehensively covers those types used by the entities, so we see a positive impact, which is in line with previous work [39].

Table 5 shows answers produced by QUINT for four sample questions. In the first and second questions, QUINT (typed) was able to restrict the answer set to only college/high school respectively where the QUINT-untyped failed. In the third question, the gold answer does not conform with answer type constraint in the question (*"city"*). Therefore, QUINT (typed) produced only the city as final answer, however, QUINT-untyped produced the correct answer. For the fourth question, typing the answer entity does not add value since both the answer type (`HumanLanguage`) as well as the type of the object of the KG predicate (`languageSpoken`) agree. The type signature for this KG predicate is `Country` and `HumanLanguage` for the subject and the object, respectively.

5.3 Results on ComplexQuestions

Results on ComplexQuestions reveal the full potential of QUINT. For prior works to fully answer ComplexQuestions they would need to have their manually created templates manually extended: a cumbersome task that quickly blows up. In contrast, QUINT trains on the structurally simple training subset of WebQuestions, which does not capture the full complexity of ComplexQuestions. It exploits language compositionality to automatically decompose complex question utterances to their constituent clauses and form simpler "sub-questions" which it answers individually before combining their answers to answer the complete question (Section 4.3).

Table 7 shows the results on ComplexQuestions achieved by QUINT and the system of Bast and Haussmann [3], the best publicly available system on WebQuestions (Table 4). It is important to keep in mind that the system of Bast and Haussmann is not designed to handle ComplexQuestions. To guarantee a fair comparison, we ran two variants of this system; Bast and Haussmann-basic is the officially published system and Bast and Haussmann++ where we i) manually decompose each complex question into its constituent sub-questions, ii) answer each sub-question using Bast and Haussmann-basic, and iii) run our stitching mechanism on the answer sets of sub-questions to answer the complete question.

QUINT achieves an F1 of 49.2, outperforming the other two systems. The difference between QUINT and Bast and Haussmann++ comes from the difference between the two systems already observed in Table 4. In both cases, this shows the effectiveness of our procedure for handling complex question detailed in Section 4.3. When looking at the results of Bast and Haussmann-basic, we see that this system is capturing one sub-question at most, to the detriment of its precision. For the first sample complex question shown in Table 6, Bast and Haussmann-basic captured only one sub-question (movie starred Woody Harrelson) leading to low precision, while QUINT correctly answered it. For the second question, capturing one sub-question is sufficient. QUINT failed on the third question since it could not stitch the answer sets for the sub-questions. It was able to generate the correct query for the first sub-question (*"everton"*) but not for the second (*"leeds"*). Bast and Haussmann-basic could not rank the correct query for the first sub-question (the only one it could capture) on top.

Question	Gold	QUINT (typed)	QUINT-untyped
"what college did john stockton go to?"	Gonzaga University	Gonzaga University	Gonzaga Prep. School, Gonzaga University
"where did aaron rodgers go to high school?"	Pleasant Valley High School	Pleasant Valley High School	Butte College, UC Berkeley, Pleasant Valley High School
"what city is acadia university in?"	Canada, Nova Scotia, Wolfville	Wolfville	Canada, Nova Scotia, Wolfville
"what language does cuba speak?"	Spanish Language	Spanish Language	Spanish Language

Table 5: Anecdotal results from WebQuestions for both variants of QUINT: typed and untyped.

Question	Gold	QUINT	Bast and Haussmann-basic
"which movie starred woody harrelson and wesley snipes?"	White Men Can't Jump, Wildcats, Money Train	White Men Can't Jump, Wildcats, Money Train	White Men Can't Jump, Wildcats, Rampart, Cheers, Money Train, ...
"what movie included characters named louis tully and dr peter venkman?"	Ghostbusters	Ghostbusters	Ghostbusters
"which players have played for everton and leeds?"	Ross Barkley	-	Goodison Park

Table 6: Sample ComplexQuestions for both QUINT and Bast and Haussmann-basic.

5.4 Discussion of Limitations

A detailed analysis of the results on the WebQuestions test set reveals that of the 2032 test questions, 3 could not be matched to any of our templates. Another 33 test questions were matched to a template, but no query candidates were generated. The first cause of this issue is the incompleteness of our predicate lexicon, resulting in empty lookups. This suggests that we can benefit from extending our lexicon construction scheme. The second cause is incorrect dependency parse trees and POS tag annotations. For example, for *"what influenced william shakespeare to start writing?"*, the verb *'influenced'* was tagged as a noun, and therefore the question was mapped to templates which could not generate any suitable query. Methodological progress on mainstream parsing and POS tagging would positively affect our system.

For 260 test questions, some query candidates were generated, but none of them returned any correct answers. We identified mistakes made by the NER/NED system, missing entries in our lexicon $\mathcal{L}$ as well as wrong gold standard as the causes. This meant that QUINT could not generate the right query as input to the ranking stage to begin with. Another important cause for this issue is the lack of any appropriate templates for some questions. For example, for the utterance *"what countries are located near egypt?"*, none of the utterance templates that match this question are paired with the appropriate query template. In this case, we need a query template that connects Egypt to adjoining countries through a CVT node.

This leaves 1736 test questions for which QUINT generates at least one candidate query that returns at least one correct answer. It is also interesting to establish an upper bound on our performance with the above issues. For this, we created an oracle ranker that returns the generated query with the highest F1 for each question. The result was an F1 of 70.0, which means that further improvement to our ranking scheme is possible.

On Free917, a detailed analysis reveals that of the 276 test questions, 3 could not be matched to any of our templates. Another 31 test questions were matched to a template, but no query candidates were generated. For 13 questions, some query candidates were generated, but none of them returned any correct answer. This leaves 229 test questions for which QUINT generates at least one candidate query that returns at least one correct answer.

On ComplexQuestions, we could answer 81 of 150. The remaining 69 questions were not answered because either one of the sub-questions did not have its correct query in the top-ranked ones, or our templates failed to capture some sub-questions.

Method	Average $F1$
Bast and Haussmann-basic	27.8
Bast and Haussmann++	46.7
QUINT	49.2

Table 7: Results on ComplexQuestions.

6. RELATED WORK

With traditional Web search over textual corpora reaching maturity, there has been a shift towards semantic search focusing on entities, and more recently on relationships. This shift was brought on by an explosion of structured [6] and semi-structured data [14].

Entity search over KGs has gained wide attention [1, 7, 17, 32, 33]. These are keyword queries asking for entities (or other resources), and have been shown to be very common [26].

More recent efforts have focused on natural language questions as an interface for knowledge graphs. Questions express complex relation-centric information needs more naturally, allowing for better KG utilization. They are also more natural when dealing with new modalities such as voice interaction (e.g., Cortana, Google Home). Multiple benchmarks have been introduced for this problem [4, 10, 34, 37]. These differ in the underlying KGs and supporting resources, size, and question phenomena they evoke, resulting in various solutions from those heavily relying on machine learning to more hybrid approaches using a combination of rules and hand-crafted scoring schemes. We presented experimental results for QUINT on the benchmarks introduced by Berant et al. [4] and Cai and Yates [10]. We could not conduct experiments on benchmarks such as QALD [37] and BioASQ [34] due to the small size of their training sets, and because they emphasize aspects not addressed by QUINT (e.g., aggregation, ordering).

Templates play an important role in QA over KGs. Unger at al. [35, 36], Yahya et al. [41, 42], and Zou et al. [51] present approaches that use manually defined templates to handle complex questions with compositionality. These systems use regularities in how syntactic patterns map to semantic ones to create their templates. The drawback of these approaches is the limited coverage of templates, making them brittle when it comes to unconventional question formulations. By automating template generation, we can learn new templates dynamically.

Fader et al. [13] use a small number of handcrafted templates for QA over KGs, focusing on simple queries with a single triple

pattern, as illustrated in Table 1. In contrast, we use dependency parsing on the utterance side, allowing for better generalization. On the query side, we are not limited to single triple patterns.

Zheng et al. [50] tackle the problem of utterance-query template construction in a setting different than ours with a query workload and a question repository as input. The task is to pair questions with workload queries that best capture them. Each pair is subsequently generalized to a template. In contrast, our approach does not rely on the availability of observed SPARQL queries.

Berant et al. [4] use a set of rules for composing logical forms in the DCS semantic representation constructed from all pairs of non-overlapping text spans. Bast and Haussmann [3] use three manually defined query templates, with no corresponding utterance templates. These are exhaustively instantiated based on the utterance. This is similar to our query generation, but is performed at answering time. Yih et al. [47] use a staged approach to map utterances to queries. Each stage uses a set of rules for adding conditions to a query at answering time. Other works [8, 12, 43] rely on embedding both questions and KG entities, paths, and subgraphs in a shared space.

Yao and Van Durme [46] address QA over KGs as an information extraction problem, reminiscent of traditional QA over text corpora. This approach makes heavy use of manually defined templates for extracting cues about the answer entity and the relevant KG predicates.

A popular formalism for mapping utterances into logical forms is CCGs [31, 49]. Kwiatkowski et al. [19] use a probabilistic CCG to build a linguistically motivated logical form. The subsequent translation to a KG-specific semantic representation is performed by a trained ontology matching model. Cai and Yates [10] combine learning a CCG grammar from question-query pairs with an approach for lexicon extension. We work with dependency parsing to capture question syntax. This is a pragmatic choice to benefit from the methodological progress and tools available [11]. Reddy et al. [28] propose an approach to map a dependency parse to a logical form in two steps. First, using a repository of manual rules, a dependency parse is transformed into a logical form whose symbols are words and edge labels. This is transformed into a semantic representation following Reddy et al. [27] using graph matching.

Work has also been done to combine KGs with supporting textual data for QA. The role of text here is either to support ranking of answer candidates [29, 40], or as a source of answers [13, 38]. In our work we use a text corpus linked to our KG for lexicon construction offline. However, we stick to answering exclusively over the KG at answering time.

Finally, going beyond a single KG, people have looked at QA over interlinked KGs. PowerAqua [22] answers questions in this setting, focusing on how to combine answers obtained from different sources. Shekarpour et al. [30] describes a two-stage system for answering questions and keyword queries over linked data, composed of two stages: question segmentation and segment disambiguation, followed by query generation.

7. CONCLUSION

We presented a method for automatically generating templates that map a question to a triple pattern query over a KG. Our system achieves performance close to the best state-of-the-art system on the WebQuestions benchmark and achieves the best result to date on the Free917 bechmark while requiring less human supervision. Additionally, our approach can be used to answer compositional questions despite never having seen such questions at training time.

8. REFERENCES

[1] K. Balog, E. Meij, and M. de Rijke. Entity search: Building bridges between two worlds. In *ISS Workshop*, 2010.

[2] J. Bao, N. Duan, M. Zhou, and T. Zhao. Knowledge-based question answering as machine translation. In *ACL*, 2014.

[3] H. Bast and E. Haussmann. More accurate question answering on freebase. In *CIKM*, 2015.

[4] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, 2013.

[5] J. Berant and P. Liang. Semantic parsing via paraphrasing. In *ACL*, 2014.

[6] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *INT J SEMANT WEB INF*, 5(3), 2009.

[7] R. Blanco, P. Mika, and S. Vigna. Effective and efficient entity search in RDF data. In *ISWC*, 2011.

[8] A. Bordes, S. Chopra, and J. Weston. Question answering with subgraph embeddings. In *EMNLP*, 2014.

[9] L. Breiman. Random forests. *Machine Learning*, 2001.

[10] Q. Cai and A. Yates. Large-scale semantic parsing via schema matching and lexicon extension. In *ACL*, 2013.

[11] L. Del Corro and R. Gemulla. Clausie: clause-based open information extraction. In *WWW*, 2013.

[12] L. Dong, F. Wei, M. Zhou, and K. Xu. Question answering over freebase with multi-column convolutional neural networks. In *ACL*, 2015.

[13] A. Fader, L. S. Zettlemoyer, and O. Etzioni. Paraphrase-Driven Learning for Open Question Answering. In *ACL*, 2013.

[14] R. V. Guha, D. Brickley, and S. Macbeth. Schema.org: evolution of structured data on the web. *Commun. ACM*, 2016.

[15] S. Hakimov, C. Unger, S. Walter, and P. Cimiano. Applying semantic parsing to question answering over linked data: Addressing the lexical gap. In *NLDB*, 2015.

[16] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING*, 1992.

[17] M. Joshi, U. Sawant, and S. Chakrabarti. Knowledge graph and corpus driven segmentation and answer inference for telegraphic entity-seeking queries. In *EMNLP*, 2014.

[18] D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *ACL*, 2003.

[19] T. Kwiatkowski, E. Choi, Y. Artzi, and L. S. Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. In *EMNLP*, 2013.

[20] P. Liang, M. I. Jordan, and D. Klein. Learning dependency-based compositional semantics. In *ACL*, 2011.

[21] P. Liang and C. Potts. Bringing Machine Learning and Compositional Semantics Together. *Annual Reviews of Linguistics*, 1, 2015.

[22] V. López, A. Nikolov, M. Sabou, V. S. Uren, E. Motta, and M. d'Aquin. Scaling up question-answering to linked data. In *EKAW*, 2010.

[23] V. López, P. Tommasi, S. Kotoulas, and J. Wu. Queriodali: Question answering over dynamic and linked knowledge graphs. In *ISWC*, 2016.

[24] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL*, 2009.

[25] S. Petrov, D. Das, and R. T. McDonald. A universal part-of-speech tagset. In *LREC*, 2012.

[26] J. Pound, P. Mika, and H. Zaragoza. Ad-hoc object retrieval in the web of data. In *WWW*, 2010.

[27] S. Reddy, M. Lapata, and M. Steedman. Large-scale semantic parsing without question-answer pairs. *TACL*, 2014.

[28] S. Reddy, O. Täckström, M. Collins, T. Kwiatkowski, D. Das, M. Steedman, and M. Lapata. Transforming dependency structures to logical forms for semantic parsing. *TACL*, 2016.

[29] D. Savenkov and E. Agichtein. When a knowledge base is not enough: Question answering over knowledge bases with external text data. In *SIGIR*, 2016.

[30] S. Shekarpour, E. Marx, A. N. Ngomo, and S. Auer. SINA: semantic interpretation of user queries for question answering on interlinked data. *J. Web Sem.*, 2015.

[31] M. Steedman. *The Syntactic Process*. 2000.

[32] T. Tran, P. Cimiano, S. Rudolph, and R. Studer. Ontology-based interpretation of keywords for semantic search. In *ISWC/ASWC*, 2007.

[33] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In *ICDE*, 2009.

[34] G. Tsatsaronis, M. Schroeder, G. Paliouras, Y. Almirantis, I. Androutsopoulos, E. Gaussier, P. Gallinari, T. Artieres, M. Alvers, M. Zschunke, and A.-C. Ngonga Ngomo. BioASQ: A challenge on large-scale biomedical semantic indexing and Question Answering. In *AAAI*, 2012.

[35] C. Unger, L. Bühmann, J. Lehmann, A. N. Ngomo, D. Gerber, and P. Cimiano. Template-based question answering over RDF data. In *WWW*, 2012.

[36] C. Unger and P. Cimiano. Pythia: Compositional meaning construction for ontology-based question answering on the semantic web. In *NLDB*, 2011.

[37] C. Unger, C. Forascu, V. López, A. N. Ngomo, E. Cabrio, P. Cimiano, and S. Walter. Question answering over linked data (QALD-5). In *CLEF*, 2015.

[38] R. Usbeck, A. N. Ngomo, L. Bühmann, and C. Unger. HAWK - hybrid question answering using linked data. In *ESWC*, 2015.

[39] A. D. Walker, P. Alexopoulos, A. Starkey, J. Z. Pan, J. M. Gómez-Pérez, and A. Siddharthan. Answer type identification for question answering - supervised learning of dependency graph patterns from natural language questions. In *JIST*, 2015.

[40] K. Xu, S. Reddy, Y. Feng, S. Huang, and D. Zhao. Question answering on freebase via relation extraction and textual evidence. In *ACL*, 2016.

[41] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum. Natural language questions for the web of data. In *EMNLP-CoNLL*, 2012.

[42] M. Yahya, K. Berberich, S. Elbassuoni, and G. Weikum. Robust question answering over the web of linked data. In *CIKM*, 2013.

[43] M. Yang, N. Duan, M. Zhou, and H. Rim. Joint relational embeddings for knowledge-based question answering. In *EMNLP*, 2014.

[44] Y. Yang and M. Chang. S-MART: novel tree-based structured learning algorithms applied to tweet entity linking. In *ACL*, 2015.

[45] X. Yao. Lean question answering over freebase from scratch. In *NAACL*, 2015.

[46] X. Yao and B. V. Durme. Information extraction over structured data: Question answering with freebase. In *ACL*, 2014.

[47] W. Yih, M. Chang, X. He, and J. Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *ACL*, 2015.

[48] P. Yin, N. Duan, B. Kao, J. Bao, and M. Zhou. Answering questions with complex semantic constraints on open knowledge bases. In CIKM, 2015.

[49] L. S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*, 2005.

[50] W. Zheng, L. Zou, X. Lian, J. X. Yu, S. Song, and D. Zhao. How to build templates for RDF question/answering: An uncertain graph similarity join approach. In *SIGMOD*, 2015.

[51] L. Zou, R. Huang, H. Wang, J. X. Yu, W. He, and D. Zhao. Natural language question answering over RDF: a graph data driven approach. In *SIGMOD*, 2014.