Online Schemaless Querying of Heterogeneous Open Knowledge Bases

Nikita Bhutani
nbhutani@umich.edu
University of Michigan, Ann Arbor

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

ABSTRACT

Applications that depend on a deep understanding of natural language text have led to a renaissance of large knowledge bases (KBs). Some of these are *curated* manually and conform to an ontology. Many others, called *open* KBs, are derived automatically from unstructured text without any pre-specified ontology. These open KBs offer broad coverage of information but are far more heterogeneous than curated KBs, which themselves are more heterogeneous than traditional databases with a fixed schema. Due to the heterogeneity of information representation, querying KBs is a challenging task.

Traditionally, query expansion is performed to cover all possible transformations and semantically equivalent structures. Such query expansion can be impractical for heterogeneous open KBs, particularly when complex queries lead to a combinatorial explosion of expansion possibilities. Furthermore, learning a query expansion model requires training examples, which is difficult to scale to diverse representations of facts in the KB. In this paper, we introduce an *online schemaless* querying method that does not require the query to exactly match the facts. Instead of exactly matching a query, it finds matches for individual query components and then identifies an answer by reasoning over the collective evidence. We devise an alignment-based algorithm for extracting answers based on textual and semantic similarity of query components and evidence fields. Thus, any representational mismatches between the query and evidence are handled online at query-time. Experiments show our approach is effective in handling multi-constraint queries.

CCS CONCEPTS

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

KEYWORDS

open knowledge bases, heterogeneity, schemaless querying

ACM Reference Format:
Nikita Bhutani and H V Jagadish. 2019. Online Schemaless Querying of Heterogeneous Open Knowledge Bases. 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/10.1145/3357384.3357874

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

ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3357874

1 INTRODUCTION

The success of cognitive applications and knowledge-centric services is attributed to the efficient and effective analysis of machine-readable knowledge bases (KBs). These KBs contain billions of ⟨*subject, predicate, object*⟩ triples about real-world entities and their relationships. Extensive efforts have been made to *curate* large KBs (e.g. Freebase [14], Yago [26], DBpedia [4], Wikidata [41]) using a pre-specified set of relations or an ontology. While the information in these KBs conforms to the ontology, they are far from complete. As new facts are discovered in non-static domains, manually expanding these KBs becomes impractical.

Open Knowledge Bases: The need to discover dynamically evolving relations has led to advances in Open-IE methods [12, 17, 34] that can extract tuples with arbitrary entities and relations from raw text without domain-specific knowledge engineering. Consequently, the extracted information provides high coverage but is also far more *heterogeneous* [25, 40] than in curated KBs.

Example 1. While a real-world fact has a unique representation in a curated KB, it can have diverse representations in an open KB. In example a) in Figure 1, it is not evident if the two triples refer to the same entities or even have the same relationship.

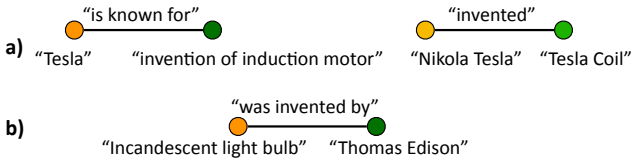

Figure 1: Facts in open KBs are heterogeneous

Facts in open KBs also exhibit variation in structure (such as n-tuple or nest-tuple) depending on the arity of the relation and contextual information [2, 12, 34]. For example, an Open IE method may extract the following tuples from a sentence *"The U.S. Supreme Court believed Tesla originally invented the Radio in 1897."*,

n-tuple t_1: ⟨*Tesla, originally invented, Radio, (in) 1897*⟩
nest-tuple t_2: ⟨*U.S. Supreme Court, believed,* t_1⟩

where the argument t_1 is a reference to the tuple t_1. Tuples can also differ in the ordering of entities, such as in ⟨*Radio, was invented by, Tesla*⟩ and ⟨*Tesla, invented, induction motor*⟩.

Querying: To retrieve information from KBs, a user can write a structured query (e.g., in SPARQL), which is answered by performing pattern matching over the KB. Clearly, the higher the heterogeneity of the KB, the greater the number of semantically equivalent structures a query has to cover. For instance, finding *"inventions of Nikola Tesla"* from the tuples in Figure 1 will require a complicated query containing multiple UNION operators.

```
SELECT ?x WHERE {
  {'Nikola Tesla'   'invented'   ?x.} UNION
  {?x   'was invented by'  'Tesla'.} UNION
  {'Tesla'   'is known for'   ?x.}
}
```

These query patterns would be far more complex if the user wanted to search for *"inventions of Serbian inventors"*.

Limitations: Traditionally, possible query transformations and semantically equivalent structures are mined [48, 52] in an offline manner, which are then used to expand a given query at query-time. Such query expansion can be impractical for heterogeneous open KBs, since there will be a combinatorial explosion of expansion possibilities, particularly for complex queries.

Example 2. Suppose the user wants to know *"which transformer did Nikola Tesla invent in 1891?"*. Figure 2 shows tuples from the open KB supporting the answer, *'Tesla Coil'*. The tuples a) and b) containing evidence differ not only in their structured relation patterns but also in the arguments. It is impractical to formulate a precise query or learn to expand a given query so it can exactly match the different expressions in the tuples.

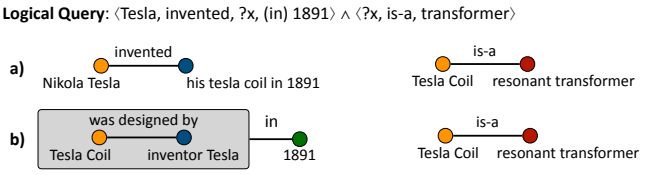

Figure 2: Heterogeneity in open KBs makes it difficult to access them via pattern matching

Approach. Instead of matching the entire query, finding matches for different query components is a much easier task. A simple keyword search [31, 42] can help find matches for 'invent', 'Nikola Tesla', 'in 1891'. These matches do not have to resemble the query specification. An answer to the query can then be found by reasoning over the collective evidence. In contrast to learning structured patterns or transformations from training examples or normalized fact representations, the alignment does not require any offline processing and can be done online at query-time.

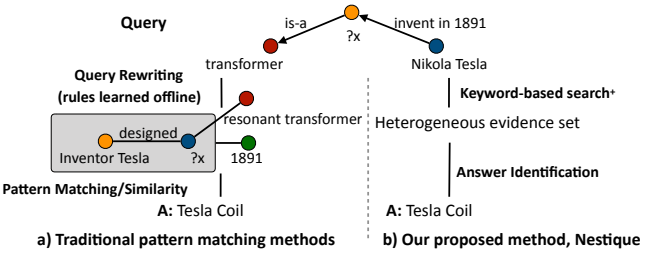

Figure 3: Pattern matching vs. our proposed method

Embodying these ideas, we propose an online *schemaless* querying method, Nestique, for accessing open KBs. Nestique is agnostic about query specification and can derive answers from facts that do not share the same representation as the query. It does so by evaluating the query in two distinct phases. It takes a complex query in a Datalog-like format, where each atom consists of a n-ary predicate and a vector of n arguments. In the first *evidence gathering* phase, it finds weak matches for each atom in the query such many relevant facts are retrieved successfully. In the second *evidence aggregation* phase, it uses more conservative reasoning to derive answers from the collective evidence. Figure 3 shows a comparison of our online, schemaless method with traditional pattern match-based querying. The two-step querying presents two major challenges:

Challenge 1: To support online processing, the evidence gathering from query components must be fast and efficient. It must achieve high recall, e.g., by considering evidence embedded in the context of tuples. Ideally, it should also join across components to ensure that precision is not too bad.

Challenge 2: How to find answers from the collective evidence? The evidence can be structurally and lexically different from the original query. The multiplicity of components in the query and arguments in the evidence involves a complex alignment problem.

Contributions: We make the following contributions:

- We propose a generalized setting for querying open knowledge bases with complex queries where the knowledge bases are heterogeneous and do not conform to a pre-defined ontology.
- We propose Nestique, an online and schemaless querying framework that does not require the user to write precise, complicated queries to access open knowledge bases. We propose a novel approach to match a complex query in parts rather than relying on exact pattern matching the whole query. Our framework is completely online and evaluates a query in two phases, namely, evidence gathering and evidence aggregation.
- We describe an efficient retrieval algorithm for collecting evidence that can find weak matches in different expressions for various query components in an online manner.
- We describe an alignment algorithm based on a novel bipartite graph that finds answers from aggregated evidence, handling any representational mismatches at query-time.
- We evaluate the proposed methods by conducting extensive experimental evaluations on three different query sets. We compare with state-of-the-art methods, OQA [23] and TAQA [50], for querying open knowledge bases.

To the best of our knowledge, we are the first to develop a systematic framework for online and schemaless querying of heterogeneous open knowledge bases. Nestique is designed to help users having complex information needs to access open knowledge bases. Our flexible framework is capable of finding good matches despite the high variance in fact representations in the knowledge base.

The rest of the paper is organized as follows. In Sec. 2, we introduce open KBs and formalize the problems studied in this paper. We describe evidence gathering in Sec. 3 and evidence aggregation for querying open KBs in Sec. 4. We briefly discuss how we handle front-end engineering issues to make querying open KBs more effective in Sec. 5. We evaluate our proposed method in Sec. 6, present a case study in Sec. 7 and describe related work in Sec. 8.

Id	Subject	Relation	Arguments
What team did Jordan play for when he played baseball?			
play_for(Jordan, ?x_Q, when play baseball), *is-a*(?x_Q, team)			
E_1	Michael Jordan	played	minor-league baseball
E_2	E_1	for	Birmingham Barons
E_3	Jordan	played	baseball,for Birmingham Barons
E_4	Barons	is-a	team
Which religious group settled in Delaware in 1638?			
settled(?x_Q, in Delaware, in 1638), *is-a*(?x_Q, religious group)			
E_5	Finns	settle	on shores of Delaware in 1638
E_6	Delaware	settle by	the Swedes
E_7	E_6	around	1638

Table 1: Example queries and evidence tuples in KBs.

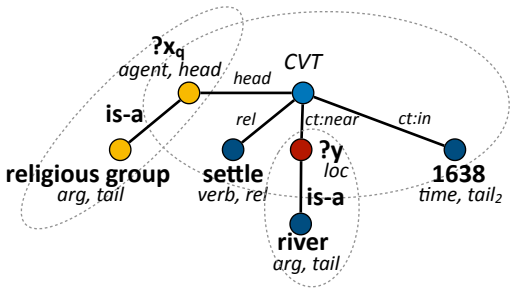

Figure 4: Example query graph and its sub-components

2 PRELIMINARIES AND OVERVIEW

Open Knowledge Bases. An open KB is a collection of n-tuples $K = \{V, E, L\}$ consisting of a set of arguments V and a set of edges E that are labeled by L. Each edge E is called an n-tuple $\langle v_h; r; v_{t_1}; \cdots; v_{t_n} \rangle$, where v_h is the head argument and v_{t_i} are tail arguments in V, and r is a relation name in L. Each edge is associated with metadata such as unique identifier, source and confidence score. For simplicity, we represent any nest-tuple as a set of n-tuples using unique identifiers for arguments. For example, tuples

t_1: ⟨*Tesla, originally invented, Radio, (in) 1897*⟩ and,

t_2: ⟨*U.S. Supreme Court, believed,* t_1⟩

represent a nest-tuple with t_1 and t_2 as identifiers. Lastly, V and L are not closed sets, i.e., they can contain multiple semantically equivalent arguments and relations, respectively.

Queries. A query Q is a Datalog-like program, consisting of a set of *RRules* (relational rules). Each *RRule* is of the form:

$$R_h(args) : R_1(args_1), R_2(args_2), \cdots, R_n(args_n)$$

We call R_h the head of the rule, and $R_1, R_2, \cdots, R_n$ the body of the rule. Each $R_i(args_i)$ is a relational atom that evaluates to true when the KB contains the tuple described by arguments $args_i$ and relation r_i. Every variable in the atom binds to values in the KB. A query can have more than one *RRule*, but has exactly one variable called query focus ?x_Q. Values that bind to ?x_Q form the answers to the query Q.

Traditional pattern-matching assumes that structured queries are formulated using the well-defined schema and vocabulary of the KB. We consider general queries that might not exactly follow the structure and semantic specifications of the KB. Specifically, we do not require each R_i and $args_i$ to exactly match a n-tuple in the KB. We only assume that the connections between different relational atoms are precisely specified.

Example 3. A complex query, *"which religious group settled near a river in 1638?"* can be posed as a set of *RRules*,

R_1(?x_Q,?y) : *settled*(?x_Q,?y) *is_a*(?y,river) *is_a*(?x_Q,religious group)
R_2(?x_Q,?y,1638) : R_1(?x_Q,?y), *settled_in*(?x_Q,1638), *settled_in*(?y,1638)
R(?x_Q) : R_2(?x_Q,?y,1638)

This query does not specify how to precisely match the relation atom *settled*(?x_Q, ?y). Here, it can match both *settle*(Roman Catholics, shores of Delaware, in 1638) and *settle_by*(Delaware, Swedes), which have different orderings and numbers of arguments.

Table 1 shows more examples of complex queries that do not match fact representations. We further represent each Datalog-style query Q as a query graph $G(Q)$.

Definition 1. (Query graph): Query graph $G(Q)$ of a query is an undirected, acyclic graph with query focus ?x_Q as the root. The vertices refer to constants and variables (e.g., Delaware, ?y). The edges refer to relations (e.g., settled, is_a) connecting the arguments.

A higher-arity relation is represented via an auxiliary vertex (called CVT) with edges to the relation and the arguments. We also annotate the query graph components with information such as their semantic roles and lemmatized values.

Definition 2. (Sub-component): A sub-component of Q is a relation edge and all its incident vertices in the query graph $G(Q)$. In case of a CVT vertex, the sub-component includes all edges connected to the vertex and their incident vertices.

Figure 4 shows a query graph with three sub-components. We find partial matches for the query by finding heterogeneous tuples from K that contain evidence for each sub-component of the query.

Definition 3. (Support Set): A support set $C_i(Q)$ is a collection of support items where each item is a maximal match for a sub-component of $G(Q)$. An item comprises tuples from K. The argument in the support set that matches ?x_Q is the answer.

2.1 Online Schemaless Querying

Figure 5 provides an overview of our proposed framework, Nestique, for online, schemaless querying of heterogeneous open knowledge bases. Given a query Q, Nestique represents it as a query graph $G(Q)$. It breaks down Q by identifying sub-components from $G(Q)$. It then finds weak matches for each sub-component from the KB and extracts an answer to the query Q from this aggregated evidence.

Evidence Gathering. To find evidence for the sub-components that could be encoded in heterogeneous tuples, it relies on fast and efficient keyword-based search over the KB. It finds tuples that contain similar terms as the sub-component. It further collects additional evidence in tuples related to the retrieved tuples.

Evidence Aggregation. Each support set collected contains an answer to the query. Nestique derives an answer a by analyzing the

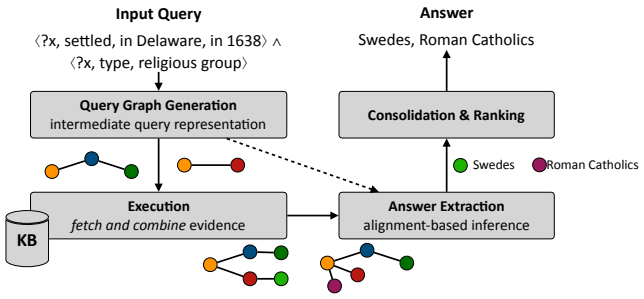

Figure 5: Overview of NESTIQUE schemaless querying.

components of the query graph and the support set, and includes a in the answer set A. It consolidates and ranks these answers using features extracted from various stages of evidence gathering.

NESTIQUE is *online* and does not rely on any offline process to learn transformation functions or equivalent patterns. It does not assume a fixed structure and representation of the tuples in the KB.

3 EVIDENCE GATHERING

3.1 Query Graph Representation

A query is specified in a DATALOG-like format. In contrast to semantic parsing, this query is not required to have a precise, formal interpretation in terms of the KB schema. For example, a user could specify a query as *settled(?x_Q, in Delaware, in 1638)* or *be_settled_by(Delaware, ?x_Q, in 1638)*, both encoding the same information.

Because our KB is unnormalized and contains only strings for arguments and relations, keyword matching and string similarity become primitive operations for matching query components. For these operations to have a high recall, we pre-process the query components: remove stop words, lemmatize, distinguish constraint modifiers from core entities. For example, the relation *settled_in* in $settled\_in(?x_Q, 1638)$ is transformed to a relation *settle* and a constraint modifier *in* for arguments $?x_Q$ and 1638. We also infer semantic roles for the query components using NLP4J [1]. This helps identify the answer argument from the evidence. We encode all this information into a query graph, as illustrated in Figure 4.

String literals in a query graph correspond to keyword-matching constraints on the tuples in the KB, while variables correspond to string-similarity join constraints. A query graph provides a general, high-recall solution to tackle the problem of minor surface-form variations. Due to high heterogeneity of the underlying open KB, simply keyword-matching the query components will result in a low recall. For example, a query

```
keyword-match(E0.r, 'settle') AND
keyword-match(E0.vt1, 'Delaware') AND
keyword-match(E0.vt2, '1638') AND
keyword-match(E1.r, 'is-a') AND
keyword-match(E1.vt1, 'religious group') AND
string-similarity(E0.vh, E1.vh) > 0.8
```

will match neither E_5 nor E_7 in Table 1. There is heterogeneity not only in the ordering of the arguments and vocabulary, but also in the structure. NESTIQUE addresses this problem of low recall by breaking down the query graph and finding matches for sub-components of a query. Specifically, for each sub-component, it finds a ranked list of tuples based on the terms mentioned in the tuples and those in the sub-component. Such relaxed query semantics helps achieve high recall despite heterogeneity in the KB.

3.2 Sub-Component Evaluation

Evaluating the queries in an online manner requires efficient retrieval of tuples containing information relevant to sub-components. A tuple $e = \langle v_h, r, v_{t_i} \rangle \in K$ is relevant if it contains similar terms as those in the arguments and relation of the sub-component. Since the KB could be very large, exhaustively finding relevant tuples for each sub-component will be expensive. Furthermore, relevant information could also be embedded in the context of a tuple (e.g., nest-tuple where one of v_h or v_{t_i} is a reference to another tuple). We, therefore, construct an inverted index using ElasticSearch [2] that includes all search terms with their corresponding tuple identifiers. For tuples that have references to other tuples, we include all terms from the referenced tuples. For each sub-component, we rank the retrieved tuples based on the following ranking function.

$$score(q, e) = queryNorm(q).coord(q, e). \sum_{t \in q} tf(t \in e).idf(t)^2$$

where *queryNorm* is the query normalization factor, tf is the term frequency and idf is the inverse document frequency. One key difference from a vector space model is the coordination factor (*coord*) that takes into account how many of the query terms are found in the tuple.

We only retain top-50 of the relevant tuples due to the size of the open KB. While such a simple approach can quickly find pieces of evidence encoded in different representations (triple, n-tuple, nest-tuple), there may be additional evidence embedded in the context of the retained tuples that might be useful for answering the query Q. We, therefore, additionally include tuples that a) are pointed to by the arguments in a matched tuple, b) point to a matched tuple, or c) share an argument with a matched tuple. To ensure the support sets are maximal matches for the sub-components of $G(Q)$, we filter out any contextual tuples that have no overlapping terms with $G(Q)$.

4 EVIDENCE AGGREGATION

We aggregate evidence for various sub-components of the query graph $G(Q)$ using a simple query optimizer that makes multiple queries to the inverted index and joins over multiple evidence from different sub-components. For joining the evidence pieces, we compute the join-key string similarity measured using the Levenshtein distance. This could return multiple support sets $C_i(Q)$ for the query Q. Note that it is possible for two support sets to share the same tuple or even the same answer. However, each support set contains a unique set of tuples as evidence for answering the query Q.

Even though keyword-based search is efficient and ensures high recall, it can sometimes find tuples with only a few terms overlap. Such tuples will increase the noise in the support sets for G as they contain little/no evidence. To alleviate this problem, we consider a support set C_i only if it matches the following criteria:

[1] https://emorynlp.github.io/nlp4j/

[2] https://www.elastic.co

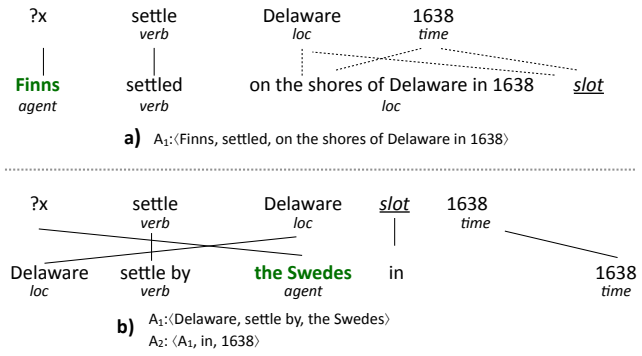

Figure 6: Alignment-based approach to extract answers from heterogeneous tuple representations

- C_i must include at least one term from each query component.
- at least one of the arguments satisfies constraints on answer type.
- *rel* in G must match to at least one *rel* in C_i.

A support set C_i for query graph G may not have the same representation. Figure 6 shows two support sets in different formats for a single query. In contrast to existing methods [23, 50], we do not rewrite or relax the query to handle such mismatches. We handle mismatches by inferring over the various items in a support set.

4.1 Answer Extraction

Our evidence gathering does not make any assumptions about the structure or schema of the tuples. As a result, a support set C_i and query graph G may have different representations. Figure 6 shows two support sets with different number and roles of items. These representational mismatches must be handled to infer an answer argument. We allow two types of mismatches between C_i and G:

- different number of components in G and items in C (e.g. 4 fields in G vs. 3 items in C)
- different roles of components in G and items in C (e.g. *Delaware* is a v_{t_1} in G vs. a v_h in C)

Given a support set C for G, the goal is to find an optimal alignment of fields in G to items in C. The field aligning to query focus $?x_Q$ in G is the answer. We model this as a maximum matching problem on a weighted bipartite graph. String literals and relational edges $f \in G$ constitute one type of nodes, and items $c \in C$ the other. In order to handle mismatches, we do not include any constraints on the alignment. We do not assume that any specific (f_i, c_j) pair will always align (e.g. v_h in query must always align to a v_h in a tuple). We include *dummy* nodes when G and C have different numbers of fields and items, respectively. Instead, we encode this information into how we assign weights to the edges connecting (f_i, c_j). We consider several indicators to derive the weight on an edge connecting (f_i, c_j):

Text Similarity, m_1**:** *tf-idf* of the lemmatized strings for f_i and c_j. This assigns high similarity to nodes that are minor surface-form variations (e.g., 'settle' vs. 'settle by')

Role Similarity, m_2**:** This is a boolean value indicating if f_i and c_j have same semantic role label (e.g., 'Delaware' has the same semantic role in the two different expressions).

Semantic Similarity, m_3**:** similarity of word2vec embeddings of (f_i, c_j). This captures semantic similarity of lexically different nodes (e.g., 'assassinate' vs. 'shot by').

Pattern Similarity, m_4**:** sum of functions p_i,

$p_1 = 1$ iff $(f_i$ is $?x_Q) \wedge (c_j$ is-of *answer type*)
$p_2 = 1$ iff $(f_i$ is $?x_Q) \wedge (c_j$ is a noun phrase)
$p_3 = 1$ iff $(f_i$ is $?x_Q) \wedge (c_j$ is a v_h or $v_t)$

These indicators ensure that an answer identity satisfies any constraints on its type (if specified in the query), is a noun-phrase, and is an argument in the tuple. The weight on an edge $e(f_i, c_j)$ is given by $w_e = f(m_1, m_2, m_3, m_4)$. The function $f()$ to compute relative weights for the different types of similarity could simply be set to compute the average, assigning equal weight to each type of similarity. It can also be tuned for optimal performance. For example, we could simply use weights for the different scoring functions as features and train a linear ranker on a query-answer dataset. We can then find an optimal alignment using the Hungarian algorithm and include the argument a aligning to $?x_Q$ in the answer set A.

4.2 Consolidation and Ranking

The answer set A will usually contain repeats: the same answer obtained with different support for its sub-components. We consolidate A using a set of features extracted from evidence gathering (e.g., number of components, relevance score of tuples, rank of retrieved tuple, extractor confidence) and answer extraction (e.g., alignment score, word count, average IDF of words). For each unique answer, we take the *best* value for each feature across the feature representations [9] and consolidate A. We score the consolidated answers using a log-linear model. We train the model using stochastic gradient descent on a set of query-answer pairs.

5 SYSTEM FRONT-END

NESTIQUE takes as input DATALOG-like queries, which are represented as query graphs. While the user can always write these queries directly, they can also be obtained by parsing a query in natural language or in any other structured format (e.g., SPARQL).

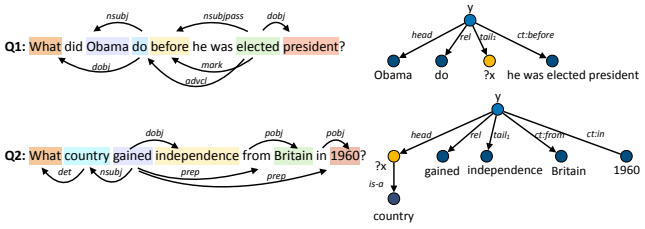

Figure 7: Example natural language queries, their dependency parse trees and query graphs

5.1 Natural Language Parsing

Since most related work for us to compare against starts with natural language questions, we need to build a natural language front end for a fair comparison. We provide a light-weight parser for translating user queries in natural language directly to query graphs to access the open knowledge bases. To generate a structured

query, a widely used approach is parsing the input natural language query into the syntactic dependency representation by employing NLP tools such as the Stanford Parser [21]. Based on the parsing result, a query skeleton is constructed [33, 49, 50, 54] depending upon target data format (e.g., relational database or RDF).

We build upon these ideas to generate a query graph for a NLQ with one difference: the relation-argument structure of the query graph is not biased towards any specific knowledge model. This is because the query graph has to be evaluated over a heterogeneous KB. However, the task is simplified because instead of precisely identifying query components, we only have to identify the sub-components. Figure 7 illustrates some example queries, their dependency structures and corresponding query graphs.

We first construct a dependency tree for the NLQ using NLP4J. A node in the tree is a word/phrase in the NLQ while an edge is a dependency relationship between two nodes. Next, we identify various components of the query graph from the parse tree:

Identify relation name, *rel*: If *root* is a copular verb, *rel* includes nodes with nsubj or attr relationship to the *root*. Otherwise, *rel* includes all nodes with any of cop, aux, auxpass relationships to the *root*. We exclude *qword* [3] in *rel*.

Identify head and tail arguments, *head* and $tail_i$: If a copular *rel* has a *qword* as a descendant, *head* is a query focus, $?x_Q$. Otherwise, we consider all nodes with either nsubj or nsubjpass relationships to *root* as a candidate c_{head} (e.g., as in Q_1). Each candidate c_{head} and nodes in the subtree rooted at c_{head} forms the *head*, except when c_{head} has a *qword* descendant. In that case, *head* is simply query focus $?x_Q$. Tail arguments $tail_i$ are identified similarly using nodes with either dobj or iobj relationships to the *root*.

Identify constraints, $c_j(target, mod, value)$: A constraint modifies either the *rel*, the *head* or any of the *tail* arguments. For each dependent node of *rel*, we include a constraint *c* with an incoming prep, advmod or mark edge as *mod* of the constraint, and the subtree of the dependent node as *value* of the constraint (e.g., as in Q_2 and Q_3). We also include constraints $c(?x, is\_a, type)$ where *type* is a subtree of node connecting *qword* and *root*.

Next, we construct the query graph with these components. The *rel*, *head*, $tail_i$ and $value_j$ form the vertices. A CVT node is used for a *rel* that has multiple constraints. We next add edges connecting *head* and $tail_i$ vertices to the *rel* vertex (in case *rel* is CVT), and *value* to *target* vertex for constraints (e.g., as in Q_2).

5.2 Paraphrasing

Users can formulate queries using informal and casual wordings and expressions. Queries having a significantly different vocabulary than the KB can result in low recall of support sets for the queries. They, therefore, must be reformulated so they share similar vocabulary and expressions as the open KB tuples. There are several works that study paraphrasing in relation to querying KBs: template-based paraphrasing, semantic parsers for curated KBs, paraphrases for neural question-answering models.

In this work, we demonstrate that a simple template-based paraphrasing technique to rewrite natural language queries can significantly boost the performance of a natural language end-point. We refer to the WikiAnswers paraphrase templates dataset [23, 24] and rewrite the NLQ using paraphrase operators. Each paraphrase operator comprises of source and target templates, such as:
`What disease killed ?a?` → `What did ?a die of?`
where ?*a* captures some argument. To use these simple templates to rewrite a complex NLQ, we ignore adverbial and prepositional modifiers in the NLQ when matching templates. For instance, we drop modifier "in 1960" to paraphrase the NLQ in Figure 7. We consider the top-k paraphrases based on the PMI score of operators and language model scores of the paraphrases. Each paraphrased NLQ can be translated to a query graph and evaluated against the KB for answers. While this can improve the recall of the answers, the candidate answers can still be consolidated and ranked as usual. Additional features, such as statistical features (e.g., PMI, co-occurrence count) and syntactical features (e.g., part-of-speech tags of template argument), can be included to make the ranking function take into account quality of paraphrases.

[3] Question Words: what, who, where, when, which

6 EXPERIMENTAL EVALUATION

We evaluate our proposed framework via an empirical study in this section. Section 6.1 describes our experimental set up, query sets and evaluation metrics. We demonstrate the effectiveness of our method by comparing with methods for querying open knowledge bases (Section 6.2) and curated knowledge bases (Section 6.3). We study how the components and KBs affect performance in Section 6.4.

6.1 System Settings

Open Knowledge Bases. We used several well-known open KBs.

- Open IE [23] is constructed using a family of open extractors that extract binary relationships from billions of web pages containing unstructured text (ClueWeb corpus). Open IE has a large set of relations. It contains many informal facts, typically not found in any curated KB.
- Nell [17] is a relatively small open KB with much fewer relation phrases. All facts are triples. Nell generally has high precision, but low recall.
- Probase [44] is an open KB with instances of only *is-a* relation extracted from 1.68B web pages. All facts are triples (e.g., ⟨star-fruit, is-a, fresh fruit⟩).
- nOKB [50] is an open KB containing n-tuple facts for higher-order relationships. These facts are extracted from web pages in English Wikipedia and the results of a commercial search engine.
- NestKB [12] is an open KB containing nest-tuple facts. Like nOKB these facts are extracted from web pages in Wikipedia and search snippets.

Table 2 summarizes these KBs. Combined these form a rich, heterogeneous knowledge source we used in our experiments.

Query Sets. In our experiments, we use three query sets: WebQuestions (WebQ), ComplexQuestions (CompQ-T), and ComplexWebQuestions (CompQ-M). Each query set is a collection of (natural language query, gold-standard-answer) pairs. The queries in natural language reflect natural, yet complex information needs of the users. They are not biased towards any specific open knowledge base. Table 3 shows statistics and example queries from these query sets.

Source	# Tuples	# Relations	KB model
Open IE	458M	6M	binary / triple
Probase	170M	1	triple
Nell	2M	300	triple
nOKB	120M	4.6M	n-tuple
NestKB	4M	0.5M	nest-tuple

Table 2: Open KBs used in our experiments

CompQ-T	what city did the Patriots play in before New England?
300 test	what country did France take over after World War 1?
	what money do they use in Spain before 2002?
CompQ-M	what is the government of France for 2010?
1300 train	when was the first Christmas celebrated in the US?
800 test	who was vice president when JFK was president?
WebQ	what do Jamaican people speak?
3778 train	who is Niall Ferguson's wife?
2032 test	what did George Orwell die of?

Table 3: Query Sets used in our experiments. CompQ-T and CompQ-M have complex queries, WebQ has simple queries.

- CompQ-T: This query set was released by the authors of [50]. The queries are crawled using Google Suggest API. Of the suggested queries, only queries containing at least one preposition (constraint) are included in the query set. These queries are not guaranteed to be answerable using a KB, curated or open.
- CompQ-M: This query set was released by the authors of [7]. The queries are selected from a query log of a practical search engine. While these queries are complex, containing multiple constraints, they are biased to be answerable using Freebase. They are selected such that each query mentions at least one entity from Freebase and has a Freebase concept as an answer.
- WebQ: This query set was released by the authors of [10]. The queries were generated from Google Autocomplete using a seed set of Freebase. Amazon Mechanical Turk users then provided answers in the form of Freebase concepts entities. Out of the three test sets, WebQ has the least number of complex queries (only 4% in the test set are multi-constraint). These queries are known *a priori* to be answerable using Freebase.

We want readers to keep a few things in mind. (1) Even though the queries come with gold-standard answers, the answer sets are not known to be complete. We evaluate a top-ranked answer manually if not already included in an answer set. We found an almost perfect inter-annotator agreement of 0.894 (Cohen's kappa) on such answers. (2) CompQ-M is answerable from Freebase and is included to compare querying methods designed specifically for curated KBs with querying methods for open KBs.

Metrics. Given a query, each method computes a ranked list of answers and returns the top-ranked one as the final answer or 'no answer'. Let $\#QA$ denote the total number of queries in a query set, $\#QC$ denote the number of queries that are correctly answered and $\#QF$ denote the number of queries with at least one answer. We report precision P ($\#QC/\#QF$), recall R ($\#QC/\#QA$) and F_1 scores of the querying methods. Then we define the precision P, recall R, and F_1 measure as follows.

$$P = \frac{\#QC}{\#QF}; \qquad R = \frac{\#QC}{\#QA}; \qquad F_1 = \frac{1}{1/P + 1/R}$$

Baseline Methods. There are several querying methods designed for curated KBs, but only a handful for open KBs.

- OQA [23] assumes the queries and KB facts are triples. It parses a NLQ into a structured, conjunctive query which is then evaluated via pattern matching. The authors of OQA have released their learned model and source code. We use OQA to parse input queries and evaluate them against the same KB as Nestique.
- TAQA [50] assumes the queries and KB facts are n-tuples. It parses a NLQ into an n-tuple query which is then evaluated via relaxed-pattern matching. The model and source code of TAQA is not publicly available. We compare with the results reported in their paper. This comparison is valid because TAQA uses the same KBs as Nestique, except for the nest-tuples that are extracted from the source as their KB. Their querying method cannot handle nest-tuples, so including them will not impact the performance.

When available, we use the trained models provided by the authors. For Nestique, we train the ranking model using the standard training set of WebQ with 3778 queries.

6.2 Effectiveness Evaluation: open KBs

Table 4 shows the performance of Nestique, and the two other baseline methods on the three query sets. In comparison to OQA, Nestique consistently achieves higher precision and recall on complex queries. OQA is designed to query triple facts by pattern-matching triple query templates. Lack of expressivity of the query model, in addition to restrictive pattern-matching, becomes a bottleneck for a complex query. This is reflected in the low recall and F_1 scores. Even when it is provided a background knowledge source with higher expressiveness (e.g., nOKB and NestKB), its querying mechanism cannot utilize the additional semantic information. Often the triple query template will not capture all the constraints in the input query. Pattern-matching such queries to KB tuples would ignore evidence in the n-ary argument or context of the tuples that could satisfy the constraints in the queries and prevent the system from finding erroneous matches.

In comparison to TAQA, Nestique achieves higher precision and recall on complex queries. Even though TAQA uses an expressive query language (n-tuple), its restrictive querying cannot extract answers from heterogeneous tuples. It enforces certain structural constraints (such as *head* and *rel* in a query and tuple should match) and does not take into account evidence embedded in the context of tuples. These constraints limit recall: 27.7% with no relaxed queries. To boost recall, it reformulates the queries, dropping certain constraints in the queries. While this improves the recall (39.3% with relaxed queries), the relaxed queries may not always capture the entire meaning of the question. For instance, dropping the constraint 'on the map' may help find matches for a question 'where is Vietnam on the map?' but will not generalize to questions such as 'who was the governor of Texas in 1934?'. Nestique does not enforce such structural constraints, enabling it to derive correct

Systems	CompQ-T			CompQ-M			WebQ		
	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1
Nestique	**55.9%**	**47.0%**	**51.1%**	**31.5%**	**21.5%**	**25.6%**	**38.3%**	26.0%	31.0%
OQA	26.3%	1.6%	3.1%	25.6%	2.7%	4.9%	28.4%	16.7%	21.0%
TAQA (No relaxation)	27.7%	27.7%	27.7%		–		32.3%	32.3%	32.3%
TAQA	39.3%	39.3%	39.3%		–		35.6%	**35.6%**	**35.6%**

Table 4: Performance of Nestique compared to other methods for querying open knowledge bases

Settings	CompQ-T	CompQ-M	WebQ
Full Model	51.1%	25.6%	31.1%
No context	28.2%	13.8%	14.9%
No paraphrasing	30.4%	17.1%	16.5%

Table 5: Component-wise contributions: F_1 score

Top-k	CompQ-T	CompQ-M	WebQ
1	55.9% (87.6%)	31.4% (70.5%)	38.8% (78.7%)
2-5	60.7% (95.1%)	37.9% (85.2%)	43.3% (87.8%)
6-10	62.3% (97.6%)	40.5% (91.0%)	45.7% (92.7%)
> 10	63.8% (100%)	44.5% (100%)	49.3% (100%)

Table 6: Rank Distribution of correct answers in top-k predictions. Numbers in parentheses are normalized to last row.

answers from tuples that are lexically and structurally different from the query.

In the WebQ query set, most of the queries are simple, single-relation queries answerable from Freebase. Thus, all methods can successfully formulate structured queries for these queries. While the restrictive representation and querying in OQA achieves reasonable precision, more flexible execution as in TAQA and Nestique achieves higher precision.

6.3 Effectiveness Evaluation: curated KBs

Freebase has attracted a lot of research attention. It is a curated KB with several querying methods and benchmark query sets that are guaranteed to be answerable from Freebase. A direct comparison to these querying methods using these query sets is not fair for two reasons: a) methods for querying curated KBs can rely on the accuracy and conciseness of the KB, b) these querying methods only have to learn to expand and reformulate logical queries for a closed set of predicates in Freebase. Methods for querying open KBs have to deal with higher heterogeneity and open vocabulary of the KB. We report the performance (F_1-scores) of a few advanced methods for querying Freebase as they provide interesting references.

We found ParaSempre [11] achieved a reasonable F_1(31%) on simple queries in WebQ known to be answerable from Freebase. However, for complex queries its performance varied depending upon whether the queries were guaranteed to be answerable from Freebase (5% on CompQ-T vs. 17% on CompQ-M). Other methods [7] report an average F_1 as high as (54% on WebQ and 42% on CompQ-M). These methods and query sets are biased towards queries that can be supported over Freebase. On the other hand, Nestique provides a mechanism to support generic, complex queries over open KBs.

6.4 Ablation Study

Table 5 shows the effects of removing different components from Nestique. It is not surprising that ignoring contextual tuples hurts performance on complex queries. Surprisingly, ignoring context for simple queries also affects the performance, implying that the correct answers are often derived using the context. We found ignoring paraphrases led to a significant drop in F_1 scores, especially for CompQ-T, which is not targeted towards any specific KB. Paraphrasing bridges the lexical gap between natural language queries and tuples. Simple keyword-based querying would not lead to similar quality results.

We found that the constraints we used to filter the support sets effectively eliminated bad support sets. When no constraints were enforced on the support set, the average number of support sets per query increased by 50%, making answer extraction slower. 84% of the newly discovered support sets didn't include a correct answer.

We also examined the ranking mechanism in Nestique and report the rank distribution of ground-truth answers in the top-k predictions. As shown in Table 6, we found that for most of the queries which could be correctly answered, the correct answer was within the top-5 predictions. For example, over 95% of true answers are among the top-5 candidates for CompQ-M.

Table 7 shows how different open KBs affect Nestique's performance. As expected, nOKB and NestKB, with rich representations significantly affect performance on complex queries. Probase is useful for simple queries in WebQ. Surprisingly, excluding Open IE does not lower the performance drastically. This is because Open IE tuples relevant to the query sets are included in nOKB and NestKB. Unlike other KBs, Nell had little effect on Nestique's performance.

7 CASE STUDY

Table 8 shows examples from test data where Nestique successfully (examples 1 and 2) or incorrectly (example 3 and 4) derived an answer. As shown in Example 1, Nestique doesn't require the query graph to exactly match a support set. It can derive the correct answer by analyzing the support set (e.g. 'do' and 'served' are semantically similar in example 1, even though these words are not themselves synonyms). As shown in Example 2, matching

Settings	CompQ-T	CompQ-M	WebQ
All KBs	55.9%	25.6%	31.1%
No Open IE	51.1%	25.4%	30.9%
No nOKB	23.6%	11.1%	12.5%
No NestKB	45.6%	19.2%	23.7%
No Probase	50.7%	25.3%	29.7%
No Nell	51.1%	25.6%	30.9%

Table 7: Contributions of open KBs: F_1 score

1	Q	what did Thomas Jefferson do before he was President?
	A	{*Secretary of State*}
	G(Q)	⟨Thomas Jefferson, do, ?x⟩ ∧ ⟨do, before, he was president⟩
	C(Q)	A_1:⟨President Thomas Jefferson, served, ϕ⟩
		A_2:⟨A_1, as, Secretary of State ⟩,
		A_3:⟨A_1, under, President Washington ⟩
2	Q	who did president Obama run against in 2008?
	A	{*John McCain, McCain*}
	G(Q)	⟨President Obama, run against, ?x⟩ ∧ ⟨run against, in, 2008⟩
	C(Q)	A_4:⟨Obama, run for president, against McCain, in 2008⟩
3	Q	what led to the split of the republican party in 1912?
	A	{*William Taft*}
	G(Q)	⟨?x, led to split, republican party⟩ ∧ ⟨led to split, in, 1912⟩
	C(Q)	A_5:⟨William Taft, lead, divided Republican Party, in 1912⟩
4	Q-A	where is french spoken most?
	A	{*England*}
	G(Q)	⟨?x, most spoke, French⟩
	C(Q)	A_4:⟨French, spoke, in England⟩

Table 8: Examples of successful and failed cases

sub-components with keyword queries helps combine evidence scattered across fields (e.g. 'Obama' in 'president' in two fields).

As shown in Example 3, Nestique can derive incorrect answers even if the lexical and structural gap is small because it fails to distinguish semantically different relation phrases (e.g. 'led to split of' and 'lead'). Future querying methods must explore the semantic similarity of queries and tuples to reduce such cases. As shown in Example 4, ignoring implicit constraints such as *aggregation* can also lead to incorrect answers. Identifying and treating these as explicit constraints is one way of validating them in the evidence.

8 RELATED WORK

There is ample work on querying knowledge bases (KBs), particularly curated KBs that can be modeled as RDF databases. In the literature on graph queries, the input is a structured query that is often presented as a query graph or pattern (e.g., [5, 8, 27]. The query pattern is evaluated against the KB by exact-matching. These methods emphasize efficiency and scale, and focus on fixed schemas and structures. As a result, a user must know the structure and vocabulary of the data being queried, and the exact values of the constants and data types. To relax such constraints of schema and structure, there has been a significant amount of research on approximate matching for graph pattern queries [28, 37, 38, 45, 52, 53] and keyword queries [46, 47] over the curated KBs. They learn semantically equivalent patterns and transformations offline and use these at query evaluation to allow for matches with similar structures or attributes to a given query. Learning such transformations or structural patterns offline is not possible for even more heterogeneous open KBs that do not have canonicalized entities and relations.

Natural language interfaces to access information in knowledge bases have also been widely studied, including template-based methods [1, 20, 39] and semantic parsers [10, 11, 15, 16, 22, 49]. They transform a natural language query into a structured query by employing templates or logical forms. They focus on learning how to map query phrases to elements in the KB and usually demand query-answer pairs as the training dataset, which makes them hard to scale [30, 35]. Furthermore, as queries become complex, deriving reliable translations becomes more challenging.

There is little work on querying open knowledge bases due to their flexible schema and open vocabulary. Querying systems for open KBs typically achieve robustness by rewriting and reformulating the query, in addition to a more relaxed semantics for matching query components [23, 24, 50]. Reformulating queries becomes infeasible as queries grow complex because the space of possible transformations grows. However, answers for complex queries can typically be found by decomposing the query [36, 51]. To the best of our knowledge, we are the first to demonstrate how such a formalism can be adopted to query heterogeneous open knowledge bases in a completely online manner.

Open KBs are derived automatically from unstructured text, which makes our work broadly related to other formalisms that query alternate structured representations of text or text directly. Corpus search approaches [13, 19] directly query the syntactic parse trees with tree-pattern queries. Alternatively, several approaches use semantic abstractions over text, such as AMR annotators [3], semantic views [29], to identify answers from the text. Techniques [18, 32] based on deep learning architectures [6, 43] learn embedding representations of text and queries to predict or generate answers. Learning these embeddings requires over 100,000 question-answer pairs.

9 CONCLUSION

We introduced Nestique, an online schemaless querying method for open KBs that leverages heterogeneity in KBs to support complex queries with multiple constraints. We described an algorithm that given a structured Datalog-like query, matches different components of the query, and finds evidence embedded in structurally diverse expressions. It can find answers from tuples that may not exactly match the query specification. Instead of learning query transformation functions offline, it handles these mismatches at query-time. Our experiments demonstrate that Nestique significantly outperforms state-of-the-art querying methods over open KBs in answering complex queries.

ACKNOWLEDGMENTS

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*. ACM, 1191–1200.

[2] Alan Akbik and Alexander Löser. 2012. KrakeN: N-ary Facts in Open Information Extraction. In *Proc. AKBC '12*. ACL, 52–56.

[3] Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage CCG Semantic Parsing with AMR. In *Proc. EMNLP '15*. ACL, 1699–1710.

[4] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *Proc. ISWC '07*. Springer, 722–735.

[5] Pablo Barceló Baeza. 2013. Querying graph databases. In *Proc. PODS '13*. ACM, 175–188.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR* abs/1409.0473 (2014).

[7] Jun-Wei Bao, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. 2016. Constraint-Based Question Answering with Knowledge Graph. In *Proc. COLING '16*. ACL, 2503–2514.

[8] Pablo Barceló, Leonid Libkin, and Juan L Reutter. 2011. Querying graph patterns. In *Proc. SIGMOD '11*. ACM, ACM, 199–210.

[9] Petr Baudis and Jan Sedivý. 2015. Modeling of the Question Answering Task in the YodaQA System. In *Proc. CLEF '15*, Vol. 9283. Springer, 222–228.

[10] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proc. EMNLP '13*. ACL, 1533–1544.

[11] Jonathan Berant and Percy Liang. 2014. Semantic Parsing via Paraphrasing. In *Proc. ACL' 14*. ACL, 1415–1425.

[12] Nikita Bhutani, H. V. Jagadish, and Dragomir R. Radev. 2016. Nested Propositions in Open Information Extraction. In *Proc. EMNLP '16*, Jian Su, Xavier Carreras, and Kevin Duh (Eds.). ACL, 55–64.

[13] Steven Bird, Yi Chen, Susan B. Davidson, Haejoong Lee, and Yifeng Zheng. 2006. Designing and Evaluating an XPath Dialect for Linguistic Queries. In *Proc. ICDE '06*, Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang (Eds.). IEEE Computer Society, 52.

[14] 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.

[15] Antoine Bordes, Sumit Chopra, and Jason Weston. 2014. Question Answering with Subgraph Embeddings. In *Proc. EMNLP '14*, Alessandro Moschitti, Bo Pang, and Walter Daelemans (Eds.). ACL, 615–620.

[16] 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.

[17] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. 2010. Toward an Architecture for Never-Ending Language Learning. In *Proc. AAAI '10*, Maria Fox and David Poole (Eds.). AAAI Press.

[18] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to Answer Open-Domain Questions. In *Proc. ACL '17*, Regina Barzilay and Min-Yen Kan (Eds.). ACL, 1870–1879.

[19] Pirooz Chubak and Davood Rafiei. 2012. Efficient indexing and querying over syntactically annotated trees. *Proc. VLDB '12* 5, 11 (2012), 1316–1327.

[20] 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.

[21] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation, LREC 2006, Genoa, Italy, May 22-28, 2006.*, Nicoletta Calzolari, Khalid Choukri, Aldo Gangemi, Bente Maegaard, Joseph Mariani, Jan Odijk, and Daniel Tapias (Eds.). European Language Resources Association (ELRA), 449–454.

[22] Li Dong, Jonathan Mallinson, Siva Reddy, and Mirella Lapata. 2017. Learning to paraphrase for question answering. (2017), 875–886.

[23] Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2014. Open question answering over curated and extracted knowledge bases. In *Proc. SIGKDD ' 14*, Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani (Eds.). ACM, 1156–1165.

[24] Anthony Fader, Luke S. Zettlemoyer, and Oren Etzioni. 2013. Paraphrase-Driven Learning for Open Question Answering. In *Proc. ACL '13*. ACL, 1608–1618.

[25] Luis Galárraga, Geremy Heitz, Kevin Murphy, and Fabian M. Suchanek. 2014. Canonicalizing Open Knowledge Bases. In *Proc. CIKM '14*, Jianzhong Li, Xiaoyang Sean Wang, Minos N. Garofalakis, Ian Soboroff, Torsten Suel, and Min Wang (Eds.). ACM, 1679–1688.

[26] 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.

[27] Jiewen Huang, Daniel J. Abadi, and Kun Ren. 2011. Scalable SPARQL Querying of Large RDF Graphs. *PVLDB* 4, 11 (2011), 1123–1134.

[28] Gjergji Kasneci, Fabian M. Suchanek, Georgiana Ifrim, Maya Ramanath, and Gerhard Weikum. 2008. NAGA: Searching and Ranking Knowledge. In *Proc. ICDE'08*, Gustavo Alonso, José A. Blakeley, and Arbee L. P. Chen (Eds.). IEEE, 953–962.

[29] Daniel Khashabi, Tushar Khot, Ashish Sabharwal, and Dan Roth. 2018. Question Answering as Global Reasoning Over Semantic Abstractions. In *Proc. AAAI '18*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 1905–1914.

[30] Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke S. Zettlemoyer. 2013. Scaling Semantic Parsers with On-the-Fly Ontology Matching. In *Proc. EMNLP '13*. ACL, 1545–1556.

[31] Wangchao Le, Feifei Li, Anastasios Kementsietsidis, and Songyun Duan. 2014. Scalable Keyword Search on Large RDF Data. *IEEE Trans. Knowl. Data Eng.* 26, 11 (2014), 2774–2788.

[32] Kenton Lee, Tom Kwiatkowski, Ankur P. Parikh, and Dipanjan Das. 2016. Learning Recurrent Span Representations for Extractive Question Answering. *CoRR* abs/1611.01436 (2016).

[33] Fei Li and HV Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *PVLDB* 8, 1 (2014), 73–84.

[34] Mausam, Michael Schmitz, Stephen Soderland, Robert Bart, and Oren Etzioni. 2012. Open Language Learning for Information Extraction. In *Proc. EMNLP-CoNLL '12*. ACL, 523–534.

[35] Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *TACL* 2 (2014), 377–392.

[36] Alon Talmor and Jonathan Berant. 2018. The Web as a Knowledge-Base for Answering Complex Questions. In *Proc. NAACL-HLT '18*. ACL, 641–651.

[37] Yuanyuan Tian, Richard C Mceachin, Carlos Santos, David J States, and Jignesh M Patel. 2006. SAGA: a subgraph matching tool for biological graphs. *Bioinformatics* 23, 2 (2006), 232–239.

[38] Yuanyuan Tian and Jignesh M Patel. 2008. Tale: A tool for approximate large graph matching. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*. IEEE, IEEE, 963–972.

[39] 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, ACM, 639–648.

[40] Shikhar Vashishth, Prince Jain, and Partha Talukdar. 2018. CESI: Canonicalizing Open Knowledge Bases using Embeddings and Side Information. In *Proc. WWW '18*. ACM, 1317–1327.

[41] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[42] Haixun Wang and Charu C Aggarwal. 2010. A survey of algorithms for keyword search on graph data. In *Managing and Mining Graph Data*. Springer, 249–273.

[43] Jason Weston, Sumit Chopra, and Antoine Bordes. 2014. Memory Networks. *CoRR* abs/1410.3916 (2014).

[44] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Qili Zhu. 2012. Probase: a probabilistic taxonomy for text understanding. In *SIGMOD Conference*. ACM, 481–492.

[45] Yinghui Wu, Shengqi Yang, and Xifeng Yan. 2013. Ontology-based subgraph querying. In *Proc. ICDE '13*. IEEE, IEEE, 697–708.

[46] Mohamed Yahya, Denilson Barbosa, Klaus Berberich, Qiuyue Wang, and Gerhard Weikum. 2016. Relationship Queries on Extended Knowledge Graphs. In *Proc. WSDM '16*. 605–614.

[47] Mohamed Yahya, Klaus Berberich, Maya Ramanath, and Gerhard Weikum. 2016. Exploratory querying of extended knowledge graphs. *Proc. VLDB '14* 9, 13 (2016), 1521–1524.

[48] Shengqi Yang, Yinghui Wu, Huan Sun, and Xifeng Yan. 2014. Schemaless and structureless graph querying. *Proc. VLDB '14* 7, 7 (2014), 565–576.

[49] Xuchen Yao and Benjamin Van Durme. 2014. Information Extraction over Structured Data: Question Answering with Freebase.. In *Proc. ACL '14*. ACL, 956–966.

[50] Pengcheng Yin, Nan Duan, Ben Kao, Junwei Bao, and Ming Zhou. 2015. Answering questions with complex semantic constraints on open knowledge bases. In *Proc. CIKM '15*. ACM, ACM, 1301–1310.

[51] 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.

[52] Weiguo Zheng, Lei Zou, Wei Peng, Xifeng Yan, Shaoxu Song, and Dongyan Zhao. 2016. Semantic SPARQL similarity search over RDF knowledge graphs. *Proc. VLDB '16* 9, 11 (2016), 840–851.

[53] Lei Zou, Lei Chen, and M Tamer Özsu. 2009. Distance-join: Pattern match query in a large graph database. *Proceedings of the VLDB Endowment* 2, 1 (2009), 886–897.

[54] 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, ACM, 313–324.