Answering Complex Queries with Heterogeneous Structured Knowledge Sources extracted from Text

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

ABSTRACT

Finding precise answers to user-issued queries, often posed in natural language (NL), has long been an elusive goal of DB and NLP researchers. Structured knowledge bases (KBs) have been widely adopted for this task. However, the diversity and complexity of queries are often limited by the inherent incompleteness of the KB. Every new textual data source first has to be curated and the KB can only retain information it is capable of representing. To address these limitations, *open* knowledge-based question-answering (KB-QA) is gaining popularity. These systems are powered by KBs that attempt to retain all the information in the textual data source.

Intuitively, a KB-QA system that can support NL queries should also be able to store the NL text with all its complexity and nuance. To this end, open information extraction (OPEN-IE) has made headway in identifying an array of tuple facts from NL text. However, existing methods still lose a great deal of contextual information critical to answering complex queries. We propose to encode knowledge in an nest-tuple format and describe a new OPEN-IE technique, NESTIE, to preserve the contextual information of facts. While such broad-coverage information paves the way to take KB-QA to the next level, tapping into this vast knowledge requires careful re-design of querying methods. Automatically extracted KBs have massive loosely-defined schema, which makes them harder to query with traditional pattern-matching methods adopted for manually curated KBs and databases. We describe a novel schemaless, online querying method, SOQ, that does not require the user query to exactly match the facts in the open KB. Experiments show that using our proposed extraction and querying techniques, a KB-QA system can effectively answer user queries of varying complexities.

PVLDB Reference Format:
Nikita Bhutani. Answering Complex Queries with Heterogeneous Structured Knowledge Sources extracted from Text. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION

Recent years have seen a shift in how people access information online. Users now prefer to get precise answers to their queries posed in natural language (NL) rather than a link to a document containing the information:

Q_1: What is the home of the Liverpool Football Club?

This has been made possible with question-answering (KB-QA) systems which are powered by a structured knowledge database (KB) [8, 2]. Curated manually or collaboratively from publicly available information such as Wikipedia, the KB contain factual information about real-world entities (e.g., *Liverpool F.C.*, *Anfield*) and their relations (e.g., *ground*) in well-structured schemas. A KB-QA system finds precise answers to a user's query by translating it to a structured KB query. For query Q_1, this is depicted with the following query formulation and execution.

$Q_1 \xrightarrow{\text{query}} \langle \textit{Liverpool F.C., ground, ?x} \rangle \xrightarrow{\textit{execute}} \langle \textit{Liverpool F.C., ground, Anfield} \rangle \xrightarrow{\textit{answer}} \textit{Anfield}$

The range and depth of NL queries that a KB-QA system can support is limited by incompleteness of the KB and the difficulty of query formulation. The incompleteness arises because every new textual data source has to be curated and the KB can only contain facts that can be expressed with a set of pre-defined relations. Formulating KB queries is challenging since several transformations are required to match phrases in the NL query to schema elements (e.g., *'home of'→ground*). Consider a complex NL query:

Q_2: Which religious group settled near a river in 1638?

As illustrated in Figure 1, the information to answer the complex query could be missing from the KB or encoded in such a complex schema that it is hard to formulate a precise KB query that can retrieve the correct answer. *Open* KBs have alleviated these two limitations in KB-QA and reformed how information is stored and retrieved in KBs. Open KBs attempt to retain all the information in the textual data source. These are constructed using open information extraction (OPEN-IE) techniques which automatically extract tuple facts about an unbounded set of relations from the NL text. Table 1 shows example facts extracted using OPEN-IE.

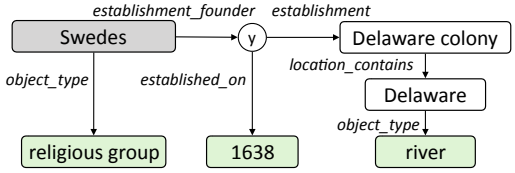

Figure 1: Snippet from curated KB with query components highlighted in green.

Since tuples are extracted automatically from NL text, their arguments and relations are simply strings. They can, therefore, be

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1	A few years later in 1638, some Swedish settlers established the first permanent settlement in Delaware. **Tuples**: ⟨Swedish settlers; established; settlement in Delaware⟩ ⟨Swedish settlers; established; settlement in 1638⟩
2	Finns settled on the shores of Delaware in 1638. **Tuples**: ⟨Finns; settled; on the shores of Delaware⟩ ⟨Finns; settled on the shores of Delaware; in 1638⟩

Table 1: Example text snippets and extracted tuple facts.

used directly to answer NL queries by string matchings instead of relation inference. However, the price paid is the tuple facts in an open KB are neither canonicalized nor semantically grounded like schema elements in a curated KB. Although synonymous, relations like *settled* and *established* will co-exist. However, such light-weight structuring of NL text is easy to derive (compared to manual curation) and easy to query (compared to raw text).

Although open KBs put open-domain question answering within reach, we are still far from supporting complex queries such as in traditional databases. First, existing OPEN-IE methods lose contextual information as they try to extract facts that are tuples of the form $t = \langle v_h, r, v_t \rangle$ where v_h and v_t are strings possibly denoting entities, and r is a string relation between them. This representation cannot accommodate contextual information about facts such as conditionals, attributions and temporal constraints. What is lost in the extraction process, cannot be queried. One way to mitigate the loss of information is to use a more expressive representation that could capture complex relations and sentence constructions, while still maintaining granularity in the information captured. We propose to extract facts as *nest-tuples* wherein v_h and v_t can be references to other extracted tuples. We develop an OPEN-IE technique, NESTIE, that learns to extract such nest-tuples given a small set of hand-crafted extraction patterns and a textual entailment dataset.

As mentioned earlier, facts in open KBs are not canonicalized. A same real-world fact can have multiple representations in the open KB. Simply modeling a NL query into a pre-defined query format that exactly matches a fact representation will result in low recall of the KB-QA system. Consider a n-tuple query for Q_2:

$Q_2 \xrightarrow{\text{query}} \langle$*?x, settled; in ?y; in 1638*$\rangle \wedge \langle$*?y; is-a; river*$\rangle$

The query will partially match tuples from example 2 in Table 1 which resemble the query specification. But it will not match tuples from example 1. Traditionally, such heterogeneity is handled by mining query transformations and semantically equivalent structures for expanding queries. This can quickly lead to combinatorial explosion of expansion possibilities, particularly for complex queries. However, since the arguments and relations in open KB facts are strings, it is easy to find matches for components of the query. We propose a schemaless querying technique, SOQ, that instead of matching the query specification as a whole, finds matches for different query components and computes the answer by reasoning over the collective evidence.

To summarize, we make the following contributions:

- We propose a more expressive fact representation, *nest-tuple*, to address the problems of loss of information and granularity in OPEN-IE. We describe an OPEN-IE technique, NESTIE for constructing open KBs. It learns broad-coverage domain-independent extraction patterns using a small set of seed extraction patterns and textual entailment dataset.

- We propose SOQ, an online and schemaless querying framework that does not require the user to write precise, complicated queries for accessing open KBs. We propose a novel approach to match a complex query in parts rather than relying on exact pattern matching the whole query. SOQ is completely online and evaluates a query in two phases, namely, evidence gathering and evidence aggregation.

- Using NESTIE and SOQ, we develop a KB-QA system that can dynamically ingest new raw NL text into open KBs and support diverse, open-domain, complex NL queries.

- We conduct experiments to demonstrate the effectiveness of our techniques on different text corpora and query sets.

2. PRELIMINARIES AND OVERVIEW

Open Knowledge Base. An open knowledge base (KB) is a collection of n-tuple facts $K = \{V, E, L\}$, extracted from a text corpora $\mathcal{D}$ by an OPEN-IE technique. V is a set of arguments and E is a set of edges that are labeled by L. Each edge E represents a n-tuple $\langle v_h; r; v_{t_1}, \ldots v_{t_n} \rangle$, where v_h is the head argument and v_{t_i} are tail arguments in V, and r is a relation name in L.

Definition 1. (Nest-tuple): A nest-tuple is a type of n-tuple where v_h and v_{t_i} can be references to other tuples.

Queries. A complex query Q is a DATALOG-like program, consisting of a set of $RRules$. Each $RRule$ is of the form:

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

where R_h is the head of the rule, and $R_1, .., R_n$ is the body of the rule. Each $R_i(args_i)$ is a relational atom that evaluates to true when K contains the tuple described by arguments and the relation. Any variables in the atom bind to values in K. A complex query can have multiple $RRule$ but has a variable called query focus $?x_Q$. Values that bind to $?x_Q$ form the answers to the query Q. Our running example query Q_2 from our running example can be posed as a set of rules,

$R_1(?x_Q,?y)$: *settle*$(?x_Q,?y)$ *is_a*$(?y,$river$)$ *is_a*$(?x_Q,$religious group$)$
$R_2(?x_Q,?y,1638)$: $R_1(?x_Q,?y)$,*settle_in*$(?x_Q,1638)$,*settle_in*$(?y,1638)$
$R(?x_Q)$: $R_2(?x_Q,?y,1638)$

The query does not have to specify how each relation atom must be matched. For example, *settle*$(?x_Q,?y)$ can match *settle*(Finns, shores of Delaware, in 1638) and *settle_by*(Delaware, Swedes).

2.1 Extracting tuples from NL text

To design an open KB-QA system that can answer complex queries, our first sub-task is to populate the open KB with automatically extracted facts, while retaining any contextual information. For this, we must learn an open information extractor that given a sentence can extract all possible n-tuple facts from the sentence. This problem has been previously addressed in [7]. The extractor is typically built on a set of domain-independent templates.

Definition 2 (Template)**.** A template maps a dependency parse-tree pattern to a tuple representation. Arguments in the templates are treated as a sequence of words to capture nominal modifiers.

We do not have access to a set of n-tuples to learn the templates. One way of generating a seed set of n-tuples is to hand-write templates and run them over a textual corpus. More equivalent parse-tree patterns for the hand-written templates can then be learned by bootstrapping over a textual entailment dataset. Embodying these ideas, we design an open information extractor, NESTIE. Figure 2 shows the system architecture of NESTIE. We describe how NESTIE learns patterns and extracts n-tuple facts in Section 3.

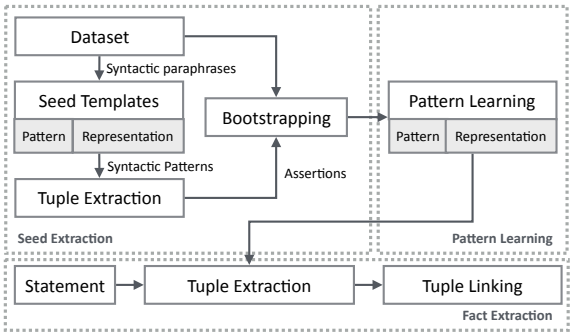

Figure 2: System Architecture of NESTIE.

2.2 Querying heterogeneous Open KB

OPEN-IE methods help construct broad-coverage open KBs that are also heterogeneous. The heterogeneity makes it difficult to formulate a precise query or expand a given query to match the different fact representations. This necessitates more forgiving database access modalities that can handle heterogeneity not only in the ordering of the arguments in tuple facts and vocabulary but also in the structure of the tuples. One way to support complex queries over open KBs is to break them down into sub-components and match them with string matchings. For identifying sub-components, a query graph $G(Q)$ is constructed for a query Q.

Definition 3. (Query graph): Query graph $G(Q)$ of a query is an undirected, acyclic graph with query focus $?x_Q$ as the root. The vertices denote constants and variables (e.g., Delaware, $?y$). The edges denote relations (e.g., settled, is_a) connecting the arguments.

A higher-arity relation is an auxiliary vertex (called CVT) with edges to the relation and the arguments. The query graph can also be annotated with semantic roles and lemmatized values of the components. Figure 4 shows a query graph with sub-components.

Definition 4. (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.

We develop an online, schemaless querying framework SOQ (Figure 3) for supporting complex queries over open KBs. In an evidence gathering phase, it retrieves tuples containing evidence for the sub-components. In an evidence aggregation phase, the collected evidence is aggregated to construct support sets.

Definition 5. (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.

Each support set contains an answer to the query. The answer is derived by analyzing the components of the query and the support set. We describe the details of SOQ in Section 4.

3. EXTRACTING TUPLES FROM NL TEXT

We wish to learn an open extractor for identifying nest-tuples without any direct supervision, i.e. without relation-specific, handcrafted extraction patterns or domain-specific knowledge engineering. To achieve this, we have to learn extraction patterns that can map the many ways of expressing complex relations in the text to

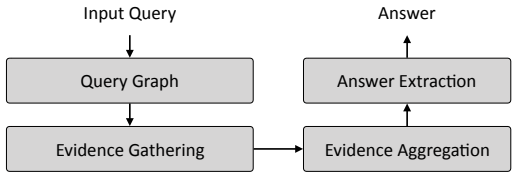

Figure 3: Overview of SOQ schemaless querying.

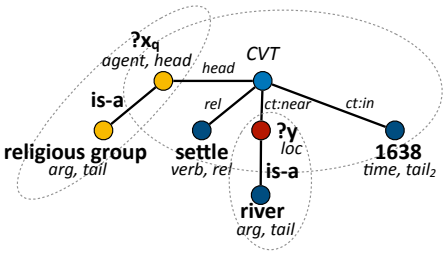

Figure 4: Example query graph and its sub-components

corresponding nest-tuple representations. In practice, it is infeasible to simply enumerate all different extraction patterns as relations, especially since complex, n-ary and multi-verb relations, could be expressed in several different ways in the text. Also, the complexity of the templates cannot be increased indefinitely as the instances in the training data that could support such templates would become sparser. NESTIE uses a multi-stage solution:

a) construct a seed set of facts with little or no nesting,
b) bootstrap sentences describing the seed facts and learn extraction patterns,
c) extract tuples from unseen sentences using learned patterns, and link extracted tuples to capture any missing information.

3.1 Constructing Seed Set

We first write a set of 13 templates, each encoding a sub-tree of the dependency parse connecting relation phrases and argument phrases. A subset of these templates is shown in Figure 5. Intuitively, we want these templates to capture the simple, common sentence constructions. Since the tuples extracted using these templates would form the basis of training, these templates and tuples must be clean and precise. The set of hypotheses in a textual entailment dataset typically exhibit these desirable properties for constructing seed set of facts for bootstrapping. The hypotheses are simple sentence constructions; their dependency parses having similar structures. We iteratively create templates until at least one tuple could be extracted from each hypothesis. We generate a seed set of facts by matching the templates against the hypotheses.

Using the seed set of tuples, we can learn the different ways of expressing them in complex sentence constructions by referring to the statements entailing the hypotheses. While a statement and hypothesis often share words, there is a class of words (e.g., prepositions, a subset of adverbs, determiners, verbs etc.) that do not modify the meaning of the hypothesis or the statement. We ignore such words while constructing the seed set.

Example 1. Consider a statement-hypothesis pair,
Statement: A few years later in 1638, some Swedish settlers established the first permanent settlement in Delaware.
Hypothesis: Delaware colony was established by Swedish settlers.
The hypothesis is entailed in the statement. The seed templates extract the following tuples from the hypothesis: ⟨*Swedish settlers; established; Delaware colony*⟩.

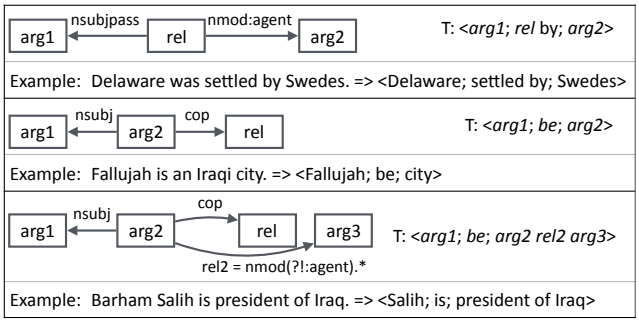

Figure 5: Seed templates and corresponding representation.

3.2 Extraction Pattern Learning

We next have to learn the various syntactic patterns that can encode the same information as the seed patterns and hence can be mapped to the same representation. We extend the bootstrapping techniques designed for binary-relations [20] to handle n-ary and complex relations. Our seed templates include patterns and corresponding representations for n-ary, complex relations (see Figure 5). This allows us to learn dependency parse-tree patterns connecting all the argument and relation phrases in a seed template. Instead of learning different ways of encoding two arguments and a relation in a tuple, we learn how all different components in a template might be expressed. This achieves higher coverage of context for the facts and prevents the arguments/relations from being over-specified and/or uninformative. To mitigate sparsity while bootstrapping, we ignore the implicit relations (e.g., nominal modifier) that can be deduced from the dependency parse. This allows to learn templates that map paraphrases such as '*Mary gave John a car*' and '*Mary gave a car to John*' to the same representation.

Specifically, NESTIE learns relation-independent, dependency-parse tree patterns for the nest-tuple representations using the set of (statement-tuples) pairs as training data. We use the Stanford dependency parser [11] to parse a statement and identify the path connecting the words of the corresponding tuple. If such a path exists, we retain the syntactic constraints on nodes and edges in the path and ignore the surface forms of nodes in the path. This helps generalize the learned patterns to unseen relations and arguments. In this manner, NESTIE could learn 183 templates from the 13 seed templates. Figure 6 shows a subset of these patterns.

Example 2. Consider the dependency sub-trees of the statement and hypothesis from Example 1,

Statement: settlers $\overset{\text{nsubj}}{\longleftarrow}$ established $\overset{\text{dobj}}{\longrightarrow}$ settlement $\overset{\text{nmod:in}}{\longrightarrow}$ Delaware

Hypothesis: Delaware $\overset{\text{nsubjpass}}{\longleftarrow}$ established $\overset{\text{agent}}{\longrightarrow}$ settlers

A seed extraction pattern maps the parse-tree of the hypothesis to the representation, ⟨*arg2; rel; arg1*⟩, returning tuple, ⟨*settlers; established; Delaware*⟩. With bootstrapping, the extraction pattern from the statement is mapped to the same representation.

3.3 Fact Extraction

Once the extraction patterns are learned, we use these patterns to extract nest-tuples from unseen sentences. We first parse a new sentence and match the patterns against the parse tree of the sentence. As the patterns only capture the heads of the arguments and relations, we expand the extracted argument and relation phrases to increase the coverage of context as in the original sentence (e.g. *Swedish* for the argument *settlers*). We refer to the parse-tree and expand the arguments on *nmod, amod, compound, nummod, det, neg* edges. We expand the relations on *advmod, neg, aux, auxpass,*

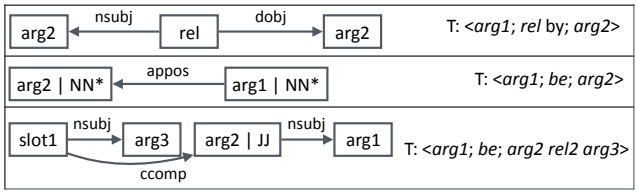

Figure 6: Syntactic Patterns learned using bootstrapping.

cop, nmod edges. Only the dependency edges not captured in the pattern are considered for expansion. Also, the order of words from the original sentence is retained in the argument phrases.

The context of extracted tuples could include condition, attribution, belief, order, reason and more. Since it is not possible to generate or learn patterns that can express complex facts as a whole, we link the various tuples from the previous step to generate nest-tuples that are complete and closer in meaning to the original sentence. We use the following rules to link the tuples.

- The relation of tuple T_1 has a dependency edge to the relation of tuple T_2.

Consider tuples, T_1: ⟨*Native Americans; inhabited; Delaware*⟩ and T_2: ⟨*Swedish settlers; established, colonies*⟩. Using dependency edge *nmod:before*, we construct a nest-tuple ⟨T_1*; before;* T_2⟩.

- Tuple T_1 is argument in tuple T_2.

Given tuples, T_1: ⟨*Chinese; invented; metal currency*⟩ and T_2: ⟨*Historians; believe;*ϕ⟩), we update T_2 to ⟨*Historians; believe;*T_1⟩.

- In a nested representation with multiple nest-tuples, a nest-tuple is replaced with a more descriptive tuple.

These rules are based on common sentence constructions, where *ccomp* edge indicates a clausal complement, an *advcl* edge indicates a conditional and a *nmod* edge indiciates a relation modifier. For conditionals and nominal modifiers, a new tuple is constructed with source and target tuples as argument based on rule 1. For clausal complements, the empty argument in the source tuple is updated with the target tuple based on rule 2.

4. QUERYING OPEN KB

While a real-world fact has a unique representation in a curated KB, it can have diverse representations in an open KB. Querying amid such heterogeneity in fact representations in open KBs can often be challenging. For instance, finding *"settlers of Delaware"* from the tuples in Figure 7 will require a complicated query containing multiple UNION operators.

```
SELECT ?x WHERE {
  {?x   'settled'   'shores of Delaware'.} UNION
  {'Delaware'   'was settled by'  ?x.}
}
```

The higher the heterogeneity of the KB, the greater the number of semantically equivalent structures a query has to cover. For instance, the query patterns will be far more complex for *"religious groups which settled around a river"* or *"religious groups which settled around a river in 1638"*. Formulating such complex query patterns is challenging from the perspective of both the user and the system. Traditionally, heterogeneity in curated KB is handled by expanding the query using query patterns and transformations which are mined offline [30, 27]. This can be impractical for complex queries and even more heterogeneous open KBs.

Since arguments and relations in tuples are simply strings, finding matches for query components is relatively easier in an open KB. For instance, a simple keyword search [22, 18] can help find

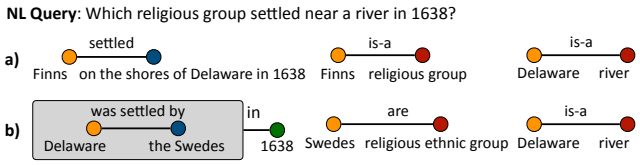

Figure 7: Heterogeneity in open KBs makes it difficult to access them via pattern matching.

matches for 'settle', 'river', 'religious group', 'in 1638'. These matches do not have to resemble the query specification. Instead of evaluating a query as a whole to find an answer, it is possible to find the answer by collecting evidence for its components. In contrast to learning patterns or transformations from training examples, this approach does not require any offline processing or adaptation.

Embodying these ideas, we develop an online schemaless querying framework, SOQ, for complex queries. These complex queries can be represented in a DATALOG-like format and may not resemble the fact representation. SOQ first transforms a query to a query graph for identifying sub-components of the query that can be matched independently over the KB. Since the primitive operations for matching queries are keywords and string similarity, it pre-processes the query components: remove stop words, lemmatize, distinguish constraint modifiers from core entities and include semantic role information.

4.1 Evidence Gathering

Next, SOQ must gather evidence in the form of tuples that contain information relevant to the various sub-components of the query. The retrieval must be efficient since the queries are evaluated in an online manner. Since the KB could be very large, exhaustively finding relevant tuples for each sub-component is expensive. Furthermore, relevant information could also be embedded in the context of a tuple (e.g., nest-tuple where one of head or tail arguments is a reference to another tuple). We, therefore, construct an inverted index that includes all search terms with corresponding tuple identifiers and use it to retrieve tuples for a query based on the terms mentioned in the query and the tuples. We only retain top-50 relevant tuples due to the size of the KB. This simple approach can quickly find pieces of evidence encoded in different representations (triple, n-tuple, nest-tuple). We further include contextual tuples of the retrieved tuples based on their overlap with the query terms.

4.2 Evidence Aggregation

In the next step, SOQ has to aggregate the evidence for various sub-components of the query graph $G(Q)$. We use a simple query optimizer that makes multiple queries to the inverted index and joins over multiple evidence from different sub-components and constructs support sets. 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. While multiple support sets could share the same tuple or even the same answer, but each support set contains a unique set of tuples as evidence. To avoid noise and computation overhead, we further prune the support sets based on the number of query components matched such as the number of relations and arguments that contain the query terms.

4.2.1 Answer Extraction

Since the evidence gathering does not make any assumptions about the structure of the tuples, a support set C_i and query graph G may have different representations (as shown in Figure 8). These

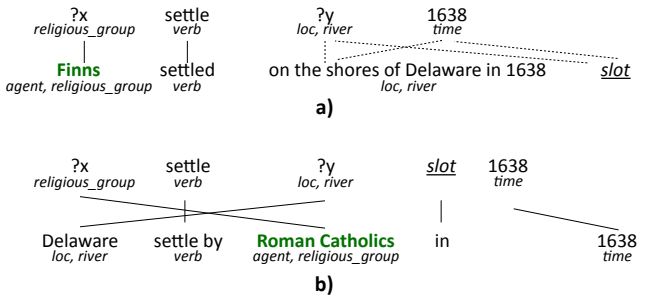

Figure 8: Alignment-based approach to extract answers from heterogeneous tuple representations

representational mismatches must be handled to infer an answer argument. To find the argument in support set C that corresponds to query focus $?x_Q$ in G, we need to find an optimal alignment of components in G to items in C. This can be modeled 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. No constraints are enforced on the alignment to accommodate mismatches i.e. no specific (f_i, c_j) pair is assumed to always align. A (f_i, c_j) pair is likely to align if:

- f_i and c_j are surface-form variations ('settle' vs. 'settle by').
- f_i and c_j have same semantic role label ('Delaware' has the same semantic role in the two different expressions)
- f_i and c_j are synonymous or semantically similar ('assassinate' vs. 'shot by').

The weight on an edge $e(f_i, c_j)$ is given by a function $f()$ over these indicators. The function $f()$ 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.

4.2.2 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 [5] 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.

4.3 Front-End

SOQ 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).

4.3.1 Natural Language Parsing

We provide a light-weight parser for translating NL query to query graphs. A widely used approach is to parse the NLQ into a syntactic dependency representation. A query skeleton is then generated depending upon the target data format [19, 32]. We build upon these ideas to generate 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. 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. We then identify various components of the query graph from the parse tree, namely relation name, head and tail arguments and constraints.

4.3.2 Paraphrasing

Users can formulate queries using informal, casual wordings and expressions. NLQs having significantly different vocabulary than the KB can result in low recall of support sets. NLQ, therefore, must be paraphrased so they use vocabulary similar to that of the 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 QA 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 [15, 14] and rewrite the NLQ using paraphrase operators. Each paraphrase operator comprises of source and target templates, such as:
What disease killed $?a$*?* $\rightarrow$ *What did* $?a$ *die of?*
where $?a$ captures some argument. To rewrite complex NLQ using such simple templates, we drop adverbial and prepositional modifiers in the NLQ when matching templates. We consider the top-k paraphrases based on the PMI score of operators and language model scores of the paraphrases. Each paraphrased NLQ is then translated and evaluated against the KB for answers.

5. OVERALL SYSTEM

An open KB-QA system can answer a broad set of user-issued complex queries provided the KB preserves the richness of the textual data sources. Since natural language has great variation, no single structure can be capable of representing all of the rich variations. In relational databases, and in RDF stores, it is commonplace to have a single fact partitioned into a set of tuples, which can be joined together to obtain the complete fact. Even if a representation for a tuple is simple, the information is enriched with connections between tuples, and we can, therefore, represent the potentially complex information in a natural language sentence with a set of interconnected tuples. Based on this idea, our open-information extractor NESTIE, captures the richness of NL text using a set of interconnected nest-tuples and equips a KB-QA system with the necessary background knowledge to answer complex queries.

Even though rich, open knowledge bases, are naturally more heterogeneous than their carefully curated counterparts. There is a long line of work on *curated* KB-QA systems that focus on understanding NL queries in terms of a well-defined schema of the KB. The major challenge for these systems is to learn to translate the NL queries into exact database queries, using templates or complex query transformations. With open KBs, the focus has shifted to accommodate different ways in which the query components could be expressed across facts in the KB. The querying modalities for open KBs, therefore, have to deviate from the conventional query mechanisms where the database query (such as SPARQL) is issued against the KB and tuples that match the patterns specified in the query are retrieved. With heterogeneous fact representations, it is not possible to enumerate and expand different ways in which query components could be captured in the tuple facts. Query execution, thus, needs to follow a retrieval-based approach to gather evidence to answer a query and then compute an answer from the collected evidence. Based on this idea, our schemaless, online querying framework, SOQ, helps answer complex queries when the schema of the underlying KB is not known in advance.

6. EXPERIMENTS

We evaluate the effectiveness of techniques we proposed for designing an open KB-QA system for answering complex queries. First, we demonstrate that NESTIE can convert an NL sentence into a set of tuples that capture the meaning of the sentence without losing granularity of information captured. Second, we demonstrate the SOQ can answer complex, multi-constraint queries with no prior knowledge of the schema or structure of facts in the KB.

6.1 NestIE

We used two datasets released by [12] in our experiments: 200 random sentences from Wikipedia, and 200 random sentences from New York Times (NYT). We compared NESTIE against three OPEN-IE systems: REVERB, OLLIE and CLAUSIE. Since the source code for each of the extractors was available, we independently ran the extractors on the two datasets. For fair comparison, we configured the extractors to generate triple facts. Two CS graduate students labeled each triple for correctness (0 or 1) and minimality (0 or 1). For each sentence, they label the set of triples for informativeness (0-5) based on the coverage of information in the sentence.

The results of our experimental study are summarized in Table 2 which shows the number of correct and minimal triples, as well as the total number of triples for each extractor and dataset. We found moderate inter-annotator agreement: 0.59 on correctness and 0.53 on minimality for both the datasets. As can be seen, NESTIE produced many more triples and more informative triples than other systems. There seems to be a trade-off between informativeness and correctness (which are akin to recall and precision, respectively). CLAUSIE achieves results closest to ours. However, the nested representation and tuple-linking used by NESTIE produce substantially more (1.7-1.8 times more) minimal triples than CLAUSIE, which generates triples from the constituents of the clause. Learning non-verb mediated extraction patterns and tuple-linking also increase the syntactic scope of relation expressions and context. This is also reflected in the average informativeness score of the triples. NESTIE achieves 1.1-1.9 times higher informativeness score than other systems.

We believe that nested representation directly improves minimality, independent of other aspects of extractor design. To explore this idea, we analyze the triples of OLLIE that are labeled correct but not minimal and find triples that can be made minimal and informative with a nested representation. We found 73.75% of the non-minimal correct triples could further be reduced, improving the minimality score of OLLIE by 17.65%.

6.2 SOQ

To demonstrate the effectiveness of SOQ on querying heterogeneous fact representations, we used several well-known open KBs. (1) OPEN-IE [14] is constructed using a family of open extractors that extract binary relationships. NELL [9] is a relatively small open KB with much fewer relation phrases. (3) PROBASE [23] is an open KB with instances of only *is-a* relations. (4) nokb [29] is an open KB containing n-tuple facts for higher-order relationships. (5) NESTKB [7] is an open KB containing nest-tuple facts. We compared SOQ against two querying mechanisms for open KBs, OQA [14] and TAQA [29]. OQA assumes the queries and KB facts are triples and evaluates queries via pattern matching. TAQA assumes queries and KB facts are n-tuples and evaluates queries via

Dataset		ReVerb	Ollie	ClausIE	NestIE
NYT	Informativeness	1.437/5	2.09/5	2.32/5	**2.762/5**
	Correct	187/275 (0.680)	359/529 (0.678)	527/882 (0.597)	469/914 (0.513)
	Minimal	161/187 (0.861)	238/359 (0.663)	199/527 (0.377)	**355/469 (0.757)**
Wikipedia	Informativeness	1.63/5	2.267/5	2.432/5	**2.602/5**
	Correct	194/258 (0.752)	336/582 (0.577)	453/769 (0.589)	415/827 (0.501)
	Minimal	171/194 (0.881)	256/336 (0.761)	214/453 (0.472)	**362/415 (0.872)**

Table 2: Informativeness and number of correct and minimal tuple-assertions as fraction of total number of assertions.

Systems	CompQ-T			CompQ-M			WebQ		
	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1
SOQ	**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 3: Performance of SOQ compared to other methods for querying open knowledge bases

relaxed-pattern matching. We report the precision, recall and F_1 scores of answers retrieved by each method.

Table 3 shows the performance of SOQ, and the two other baseline methods. In comparison to OQA, SOQ consistently achieves higher precision and recall on complex queries. OQA lacks expressivity in the query model, in addition to restrictive pattern-matching. This becomes a bottleneck for a complex query. 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. In comparison to TAQA, SOQ 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 and does not take into account evidence embedded in the context of tuples. These constraints limit recall: 27.7% with no relaxed queries. SOQ does not enforce such structural constraints, enabling it to derive correct 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 SOQ achieves higher precision.

7. RELATED WORK

There is ample work on querying knowledge bases (KBs), particularly curated KBs that can be modeled as RDF databases. The input is a structured query that is often presented as a query graph or pattern (e.g., [4, 3, 16], which 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, approximate matching for graph pattern queries [24, 30, 17, 31] and keyword queries [25, 26] have been adopted. Learning equivalent structural patterns and transformations 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 KBs has also been widely studied, including template-based methods [21, 1, 10] and semantic parsers [6, 28, 13]. They transform a NL 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. Furthermore, as queries become complex, deriving reliable translations becomes more challenging.

There is little work on querying open KBs due to their flexible schema and open vocabulary. Open KB-QA systems rewrite and reformulate the query, in addition to a more relaxed semantics for matching query components [15, 14, 29]. Reformulating queries becomes infeasible as queries become complex. However, answers for complex queries can typically be found by decomposing the query. To the best of our knowledge, we are the first to demonstrate how such a formalism can be adopted to query heterogeneous open KBs in a completely online manner.

8. CONCLUSION

Open-IE has revolutionized how NL text is structured with little/no human effort in structured open KBs. Yet, the state of the art in open KB-QA involves very simple questions from a DB perspective. Evaluating more complex queries efficiently needs the extraction process to preserve the rich information in NL text and the querying process to adapt to the flexible schema of the open KB. In this work, we have developed NestIE and SOQ to accomplish this goal.

9. REFERENCES

[1] A. Abujabal, M. Yahya, M. Riedewald, and G. Weikum. Automated template generation for question answering over knowledge graphs. In *Proc. WWW '17*, pages 1191–1200. ACM, 2017.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. *The semantic web*, pages 722–735, 2007.

[3] P. B. Baeza. Querying graph databases. In *Proc. PODS '13*, pages 175–188. ACM, 2013.

[4] P. Barceló, L. Libkin, and J. L. Reutter. Querying graph patterns. In *Proc. SIGMOD '11*, pages 199–210. ACM, ACM, 2011.

[5] P. Baudis and J. Sedivý. Modeling of the question answering task in the yodaqa system. In *Proc. CLEF '15*, volume 9283, pages 222–228. Springer, 2015.

[6] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *Proc. EMNLP '13*, pages 1533–1544. ACL, 2013.

[7] N. Bhutani, H. V. Jagadish, and D. R. Radev. Nested propositions in open information extraction. In J. Su, X. Carreras, and K. Duh, editors, *Proc. EMNLP '2016*, pages 55–64. ACL, 2016.

[8] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proc. SIGMOD '2008*, pages 1247–1250. AcM, 2008.

[9] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell. Toward an architecture for never-ending language learning. In M. Fox and D. Poole, editors, *Proc. AAAI '2010*. AAAI Press, 2010.

[10] W. Cui, Y. Xiao, H. Wang, Y. Song, S.-w. Hwang, and W. Wang. Kbqa: learning question answering over qa corpora and knowledge bases. *Proc. VLDB'17*, 10(5):565–576, 2017.

[11] M.-C. De Marneffe, B. MacCartney, C. D. Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proc. LREC '2006*, volume 6, pages 449–454, 2006.

[12] L. Del Corro and R. Gemulla. Clausie: clause-based open information extraction. In *Proc. IW3C2 '2013*, pages 355–366, 2013.

[13] L. Dong, J. Mallinson, S. Reddy, and M. Lapata. Learning to paraphrase for question answering. pages 875–886, 2017.

[14] A. Fader, L. Zettlemoyer, and O. Etzioni. Open question answering over curated and extracted knowledge bases. In S. A. Macskassy, C. Perlich, J. Leskovec, W. Wang, and R. Ghani, editors, *Proc. SIGKDD ' 2014*, pages 1156–1165. ACM, 2014.

[15] A. Fader, L. S. Zettlemoyer, and O. Etzioni. Paraphrase-driven learning for open question answering. In *Proc. ACL 2013*, pages 1608–1618. ACL, 2013.

[16] J. Huang, D. J. Abadi, and K. Ren. Scalable SPARQL querying of large RDF graphs. *PVLDB*, 4(11):1123–1134, 2011.

[17] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. NAGA: searching and ranking knowledge. In G. Alonso, J. A. Blakeley, and A. L. P. Chen, editors, *Proc. ICDE'08*, pages 953–962. IEEE, 2008.

[18] W. Le, F. Li, A. Kementsietsidis, and S. Duan. Scalable keyword search on large RDF data. *IEEE Trans. Knowl. Data Eng.*, 26(11):2774–2788, 2014.

[19] F. Li and H. Jagadish. Constructing an interactive natural language interface for relational databases. *PVLDB*, 8(1):73–84, 2014.

[20] M. Schmitz, R. Bart, S. Soderland, O. Etzioni, et al. Open language learning for information extraction. In *Proc. EMNLP-CoNLL '2012*, pages 523–534, 2012.

[21] C. Unger, L. Bühmann, J. Lehmann, A.-C. Ngonga Ngomo, D. Gerber, and P. Cimiano. Template-based question answering over rdf data. In *Proc. WWW '12*, pages 639–648. ACM, ACM, 2012.

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

[23] W. Wu, H. Li, H. Wang, and K. Q. Zhu. Probase: a probabilistic taxonomy for text understanding. In *Proc. SIGMOD '2012*, pages 481–492. ACM, 2012.

[24] Y. Wu, S. Yang, and X. Yan. Ontology-based subgraph querying. In *Proc. ICDE '13*, pages 697–708. IEEE, IEEE, 2013.

[25] M. Yahya, D. Barbosa, K. Berberich, Q. Wang, and G. Weikum. Relationship queries on extended knowledge graphs. In *Proc. WSDM '16*, pages 605–614, 2016.

[26] M. Yahya, K. Berberich, M. Ramanath, and G. Weikum. Exploratory querying of extended knowledge graphs. *Proc. VLDB '14*, 9(13):1521–1524, 2016.

[27] S. Yang, Y. Wu, H. Sun, and X. Yan. Schemaless and structureless graph querying. *Proc. VLDB '14*, 7(7):565–576, 2014.

[28] X. Yao and B. Van Durme. Information extraction over structured data: Question answering with freebase. In *Proc. ACL '14*, pages 956–966. ACL, 2014.

[29] P. Yin, N. Duan, B. Kao, J. Bao, and M. Zhou. Answering questions with complex semantic constraints on open knowledge bases. In *Proc. CIKM '2015*, pages 1301–1310. ACM, ACM, 2015.

[30] W. Zheng, L. Zou, W. Peng, X. Yan, S. Song, and D. Zhao. Semantic sparql similarity search over rdf knowledge graphs. *Proc. VLDB '16*, 9(11):840–851, 2016.

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

[32] 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 *Proc. SIGMOD '2014*, pages 313–324. ACM, ACM, 2014.