

Answering Complex Questions Using Curated and Extracted Knowledge Bases

by

Nikita Bhutani

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2019

Doctoral Committee:

Professor Hosagrahar V. Jagadish, Chair
Associate Professor Michael J. Cafarella
Assistant Professor Walter S. Lasecki
Dr. Yunyao Li, IBM Research
Associate Professor Qiaozhu Mei
Professor Rada Mihalcea

Nikita Bhutani

nbhutani@umich.edu

ORCID iD: [0000-0002-6687-2579](https://orcid.org/0000-0002-6687-2579)

© Nikita Bhutani 2019

To my family and partner

ACKNOWLEDGEMENTS

To achieve something great, one must dare to fail greatly too. If I have come this far, it is because of those who believed in me when I did not and those who lent support when I needed the most:

H V Jagadish, my Ph.D. advisor and mentor, for giving me an opportunity I thought was far bigger than my talents and skills as an undergrad with no prior research experience. He provided an intellectually stimulating environment, boundless enthusiasm and guidance, which shaped my journey in graduate school. He always encouraged me to chart unfamiliar territories and seek ambitious and adventurous projects. His research philosophy and ethics have had a profound impact on how I think about and conduct research today. One of the most important lessons I learned was that defining problems and articulating why they matter are just as vital as technical innovations. He is my first-ever research mentor, and I'll always hold him in high regard.

My parents, Promila and Prem, for their moral encouragement, insistent optimism and unwavering support in every path I took. I still remember how scared I was when I made the career leap to study computer science. But they believed in me and motivated me to embrace the road less taken. I am also incredibly grateful to have two wonderful sisters, Neha and Bhavika, who taught me not to take my failures personally and to always look at the bright side of things.

Yunyao Li, my two-times research mentor at IBM and also my thesis committee member, for providing a great environment to work on very cool research. A brilliant

research manager and friend, she taught me how to find impactful research problems and think about the usability of the solutions we build.

Wang-Chiew Tan and Yoshihiko Suhara, my mentors at Megagon Labs, for introducing me to deep learning in natural language processing. I've learned from them how to ask hard-hitting questions and be more systematic in my research practices.

My thesis committee members Rada Michalcea, Michael Cafarella, Walter Lasecki, and Qiaozhu Mei, for spending their valuable time to provide insightful comments and feedback which helped improve this dissertation.

My research collaborators and reviewers at various points: Alon Halevy, Dragomir Radev, Xinyi Zheng, Qian Kun, Fei Li, Mauricio Hernandez, IBM's Scalable Knowledge Intelligence Group, and Megagon Labs family.

All my hilarious friends and fellow occupants of BBB4945: Pallavi Moghe, Ankita Chaudhari, Niket Prakash, Helen Hagos, Shweta Khushu, Aniket Deshmukh, Abraham Addisie, Ameer Rahmati, Earlence Fernandes, Jie Song, Zhongjun Jin, Abolfazl Asudeh, Kevin Eykholt, Chris Baik. A big shout out to all the members of the DB group for incessantly coming to all my practice talks and providing valuable feedback.

My Group-X instructors, especially Tatiana and Caymen, who made sure the day-to-day grind and work deadlines didn't hijack my much-needed gym time.

Finally, Jayadevan, for being my work-mentor and now closest friend and ally. I probably would never have chosen to pursue a Ph.D. if it wasn't for him. He walked me to the door and stood by me through the ups and downs as I walked through it.

TABLE OF CONTENTS

DEDICATION	ii
ACKNOWLEDGEMENTS	iii
LIST OF FIGURES	viii
LIST OF TABLES	x
ABSTRACT	xii
CHAPTER	
I. Introduction	1
1.1 Challenges and Strategies	5
1.1.1 Lossy Open Information Extraction	5
1.1.2 Heterogeneity in Extracted Knowledge Bases	7
1.1.3 Noisy and Non-Canonicalized Facts in Extracted Knowl- edge Bases	8
1.2 Summary of Contributions	9
1.3 Outline of the Dissertation	10
II. Related Work	11
2.1 Question-Answering	11
2.2 Querying Text	11
2.3 Knowledge Acquisition	13
2.4 Querying Knowledge Bases	16
2.4.1 Querying Curated Knowledge Bases	16
2.4.2 Querying Extracted Knowledge Bases	18
2.4.3 Querying Multiple Knowledge Sources	19
III. Identifying Facts in Text with Open Information Extraction	21

3.1	NestIE	24
3.1.1	Constructing Seed Set	25
3.1.2	Extraction Pattern Learning	26
3.1.3	Tuple Extraction	28
3.1.4	Tuple Linking	28
3.1.5	Comparison with Ollie	30
3.2	Experiments	30
3.2.1	Experimental Setup	30
3.2.2	Experimental Results	32
3.2.3	Error Analysis	33
3.3	Conclusions	34
3.4	Facts from Conversational Question-Answer Data	35

IV. Online Schemaless Querying of Extracted Knowledge Bases 37

4.1	Introduction	37
4.2	Preliminaries and Overview	41
4.2.1	Online Schemaless Querying	44
4.3	Evidence Gathering	46
4.3.1	Query Graph Representation	46
4.3.2	Sub-Component Evaluation	47
4.4	Evidence Aggregation	48
4.4.1	Answer Extraction	49
4.4.2	Consolidation and Ranking	51
4.5	System Front-End	51
4.5.1	Natural Language Parsing	51
4.5.2	Paraphrasing	53
4.6	Experimentals	54
4.6.1	System Settings	54
4.6.2	Effectiveness Evaluation: extracted KBs	58
4.6.3	Effectiveness Evaluation: curated KBs	59
4.6.4	Ablation Study	60
4.7	Case Study	62
4.8	Conclusion	63

V. Querying Curated Knowledge Bases with Query Composition 65

5.1	Introduction	65
5.2	Background	69
5.3	Solution Overview	70
5.4	Query Composition	72
5.4.1	Partial Query Candidate Generation	74
5.4.2	Query Composition and Execution	76
5.5	Semantic Matching Model	78
5.5.1	Model Architecture	78

5.5.2	Implicit Supervision	80
5.6	Experiments	83
5.6.1	Datasets and Baseline Systems	83
5.6.2	Experimental Setup	85
5.6.3	Results and Discussion	86
5.6.4	Ablation Study	89
5.6.5	Qualitative Analysis	91
5.6.6	Error Analysis	92
5.7	Conclusion	93
VI. Querying Curated and Extracted Knowledge Bases		94
6.1	Introduction	94
6.2	Task and Overview	98
6.3	Partial Query Candidate Generation	99
6.4	Semantic Matching	101
6.4.1	Model Architecture	102
6.4.2	Relation Alignment	104
6.4.3	Implicit Supervision	104
6.5	Query Composition	105
6.6	Experiments	106
6.6.1	Experimental Setup	106
6.6.2	Results and Discussion	108
6.7	Conclusion	112
VII. Concluding Remarks and Future Work		113
BIBLIOGRAPHY		116

LIST OF FIGURES

Figure

1.1	Snippet from a curated KB and an example query	3
3.1	Pattern learning and tuple extraction for nest-tuples in NESTIE. . .	24
3.2	Seed templates and corresponding representation in NESTIE.	25
3.3	Example syntactic Patterns learned using bootstrapping.	27
4.1	Facts in extracted KBs are heterogeneous	37
4.2	Heterogeneity in extracted KBs makes it difficult to access them via pattern matching	39
4.3	Pattern matching vs. our proposed method	40
4.4	Example query graph and its sub-components	44
4.5	Overview of NESTIQUE schemaless querying.	45
4.6	Alignment-based approach to extract answers from heterogeneous tu- ple representations	48
4.7	Example natural language queries, their dependency parse trees and query graphs	52
5.1	Example queries with constrained main relation (1) and multiple main relations (2 and 3). A main relation connects a topic T to an intermediate answer I or query answer A . The relation could be a n-ary relation, indicated in grey.	66
5.2	Example computation plan indicating how to construct the complex query given the partial queries	70

5.3	System Architecture of <code>TEXTRAY</code>	71
5.4	Action space showing how to generate candidates for partial query. .	74
5.5	Staged candidate generation for a partial query in <code>TEXTRAY</code>	75
5.6	Staged generation for a subsequent partial query in <code>TEXTRAY</code> . . .	76
5.7	Semantic Matching Model for comparing question representation to partial query representation.	77
6.1	Simple vs Complex questions.	95
6.2	Partial queries and derivations.	97
6.3	System Architecture of <code>MULTIQUE</code>	99
6.4	Example Candidate Generation for the running example 1.	100
6.5	Semantic Matching Model for collective reasoning over diverse rela- tion forms.	101

LIST OF TABLES

Table

3.1	Extracted tuples from Open-IE systems: REVERB, OLLIE and NESTIE.	22
3.2	Informativeness and number of correct and minimal tuples as fraction of total number of tuples.	31
4.1	Example queries and evidence tuples in KBs.	43
4.2	Extracted KBs used in our experiments	55
4.3	CompQ-T and CompQ-M have complex queries, WebQ has simple queries.	56
4.4	Performance of NESTIQUE and other systems on complex questions.	58
4.5	Performance of NESTIQUE and other systems on simple questions. .	59
4.6	Contributions of various components of NESTIQUE : F_1 score	60
4.7	Correct answers in top-k predictions. Numbers in parentheses are normalized to last row.	61
4.8	Contributions of extracted KBs: F_1 score	62
4.9	Examples of successful and failed cases in NESTIQUE	63
5.1	Average F_1 scores and Precision@1 on CompQWeb.	87
5.2	Average F_1 scores and Precision@1 on WebQSP dataset. * are results on the WebQ dataset.	88
5.3	Upper bound F_1 scores for candidate generation.	88

5.4	Percentage of questions with the highest F_1 score in the top-k candidate derivations, and the average best F_1	89
5.5	Component-wise ablation results (Average F_1) of TEXTRAY.	89
5.6	Average F_1 for different ranking variants in TEXTRAY.	90
5.7	Example failed queries from CompQWeb	93
6.1	Average F_1 / precision@1 of baseline systems and MULTIQUE in different configurations.	109
6.2	Ablation results, average F_1 / precision@1, of MULTIQUE (cKB+oKB).111	
6.3	Percentage of questions with the highest F_1 score in the top-k derivations, and the average best F_1	111

ABSTRACT

The web is awash in textual data. As users struggle to navigate this textual data, the art and science of search engines have changed dramatically in recent years. Search engines like Google are now focusing on providing concise answers in response to user questions asked in natural language, instead of delivering an assortment of links to other websites. This has been made possible with the renaissance of large-scale knowledge bases (KBs), which contain facts about real-world entities and relations between them. Curated manually or extracted automatically from textual data, KBs have helped unlock the value in abundant unstructured textual sources. Despite the tremendous progress in question-answering over a knowledge base (KB-QA), existing systems still struggle to answer a wide variety of questions, especially complex questions where multiple pieces of information have to be combined to conclude the answer. This dissertation studies the design of KB-QA systems that can answer complex questions by reasoning over knowledge bases, curated manually or extracted automatically.

KB-QA systems face two challenges. The first challenge is the loss of information at knowledge acquisition: how do we prevent extraction systems from losing contextual information critical to answering complex questions. We describe open information techniques that are robust to complex textual data and can encode a complex fact as a set of linked simple facts. The extracted facts can be used to populate a KB automatically. The second challenge is querying: how do we translate complex questions to queries to access the information in the KB. The difficulty of this task and the coverage of the KB-QA system depend on the target KB. Auto-

matically extracted KBs offer a high coverage of information but use many different patterns to express the information, making them hard to query. Manually curated KBs, on the other hand, suffer from low coverage due to their restricted schema, but can support compositional reasoning since they are constructed precisely for querying. We describe querying techniques for complex question-answering over each type of KB. We also describe a KB-QA system that can benefit from combining high-quality curated knowledge with broad-coverage automatically extracted facts for answering complex questions.

CHAPTER I

Introduction

With more information accessible today than ever, there is a shift in how people find information. Traditionally, web search engines presented a ranked list of relevant documents or short texts in response to users' information needs. They relied heavily on the user to spend a couple of minutes reading each document to get the desired information. Ranked links or snippets have become less effective with the increasing richness of information on the Web and growing complexity of information needs. Consider the following question,

Q_1 : What is the capital city of Brazil?

Users now prefer to get direct answers (Brasilia) to their questions instead of sifting through documents. This has drawn tremendous interest in question-answering (KB-QA) systems that can reason over real-world conceptual knowledge to give out concise results to user questions. To help spur development of commercial QA systems and products such as the Google Assistant, Amazon Alexa, and Apple Siri, several large-scale structured knowledge bases (KBs) have been created recently, including Google's Knowledge Graph, Freebase [20], YAGO2 [48]. Curated from publicly available information such as Wikipedia, these KBs contain factual information about popular real-world entities (e.g. Brazil, Brasilia) and the relations between them (e.g.,

capital) in well-structured schemas. A KB-QA system interacts with the KB to retrieve answers by first translating the input question to a structured query and then performing a subgraph match over the structured information in the KB. For Q_1 , this is depicted with the following query formulation and execution:

$$\langle \text{Brazil}, \text{capital}, ?x \rangle \xrightarrow{\text{query}} \langle \text{Brazil}, \text{capital}, \text{Brasilia} \rangle \xrightarrow{\text{answer}} \text{Brasilia}$$

Despite the tremendous progress in knowledge-based question answering (KB-QA), existing systems still struggle to answer complex questions about any topic:

Q_2 : Which countries import fish from Brazil?

The difficulty in building a QA system that can answer complex questions about a variety of topics can be traced to two problems: knowledge acquisition and query formulation. KB-QA systems rely predominantly on knowledge bases built using manually-defined ontologies. The ontology determines how the knowledge in the text could be encoded as concepts and relations between the concepts. Consequently, information extraction (IE) systems only learn to populate the KB with relations that are already defined in the ontology. The sparsity in terms of the number of facts becomes even more acute as questions become informal and complex. For Q_2 , the KB might model popular concepts (e.g. *president*, *capital*) but miss out the long-tail concepts (e.g. *import*) critical to answering the question.

Even when the knowledge base might contain the information to support a complex question, it often requires a complex query to retrieve it:

Q_3 : What college did the author of The Hobbit attend?

As shown in Figure 1.1, a KB-QA system has to infer several schema elements from the phrases in the question (e.g. 'author of' \rightarrow *books.written*) to construct the complex query. Learning these mappings from natural language phrases to the vocabulary of KB is a non-trivial task.

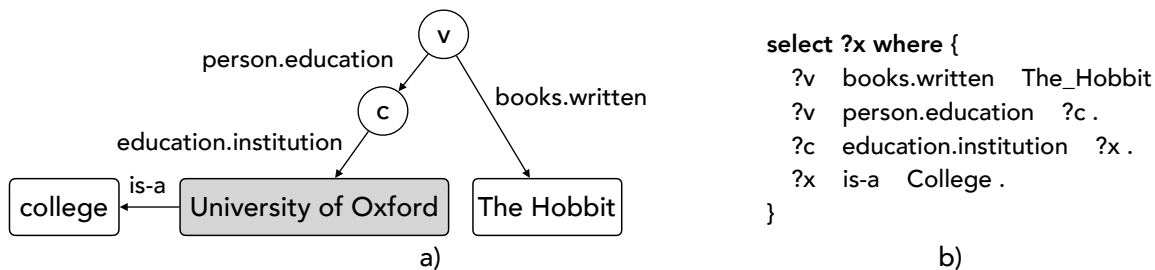


Figure 1.1: Snippet from a curated KB and an example query

As curating and querying KBs are beginning to hit scalability limits, another approach to gain insights into the textual data is to infer answers to the questions directly from the raw text. Recent advances in neural network-based machine comprehension models have stirred interest in this direction. However, these systems require a lot of training data and often target domain-specific questions. Furthermore, structured KBs are more appropriate for compositional reasoning since they are constructed precisely to be queried. Learning to seek and combine evidence for answering complex questions is much harder over raw text.

The two types of data offer competing benefits for designing a broad-coverage QA system. Curated KBs are expensive to build and often sparse but provide declarative access to information for answering complex questions. On the other hand, the raw text has the highest coverage of information, but the different textual patterns are hard to query. These limitations have led to a new wave in IE research: *open information extraction*. Open-IE methods aim to model all possible relations in the raw text without requiring any manually-defined ontologies or relation-specific training data. They model facts in an input sentence as tuples of the form $t = \langle e_i, r_{i,j}, e_j \rangle$ where $r_{i,j}$ is a textual relation between entity phrases e_i and e_j .

Sentence: Brazil was officially discovered in 1500.

Open-IE tuple: $\langle \text{Brazil}; \text{be discovered in}; 1500 \rangle$

The facts from a text corpus are then compiled into a structured KB, referred to

as *extracted* knowledge base. In contrast to manual curation, building an extracted KB is inexpensive. Also, the extracted tuples offer broad coverage of information in the raw text. While an extracted KB is an attractive choice for KB-QA, existing systems primarily focus on curated information and very simple questions. Usually, a single fact is sought, with a few (typically one or two) constraints.

To enable KB-QA systems to answer complex questions, we study various aspects of knowledge representation in curated and extracted KB. One major issue in knowledge base construction is the loss of information at extraction. What is lost at extraction cannot be queried. We investigate two potential contributing factors: inexpressive output tuple representation and complex input text format. How the KB is constructed also impacts how it is queried. A curated KB uses a fixed schema and thus can be accessed with pattern-matching or retrieval based methods. On the other hand, an extracted KB uses heterogeneous representations of facts or in other words, a flexible schema. We, therefore, have to reassess how we access this type of KB to support complex questions. Lastly, where a curated knowledge base contains information on some topic, we should generally prefer it to “best effort” information automatically extracted from the text. If we could design a querying method that can leverage both a curated KB and an extracted KB, we will be able to answer complex questions more reliably.

We seek answers to the following three questions:

- How can we minimize loss of information by retaining contextual information in extracted facts and handling complex input formats?
- Given several *extracted* KBs with massive, heterogeneous schema, what kind of query mechanisms are needed to find answers to complex questions in natural language?
- Can we design a KB-QA system that can additionally leverage the available

high-quality, curated information?

We next discuss some of the observations we made from existing Open-IE techniques and KB-QA systems. We present key technical challenges and general strategies to address the questions above. We then conclude with the key contributions and provide an outline of the work presented in this dissertation.

1.1 Challenges and Strategies

1.1.1 Lossy Open Information Extraction

Data curation processes have to resolve competing demands of affordable curation, and breadth and complexity of schema. A vast, complex schema helps serve complex queries involving several selection predicates and joins. However, extracting information from text that conforms to a pre-defined schema is difficult and expensive, requiring extensive human involvement in the form of hand-crafted extraction rules or hand-tagged training examples. For instance, a curated KB could have information about several relations (e.g., *language*, *currency*) of an entity (e.g, **Brazil**):

Brazil: *capital*(Brasilia), *currency*(Brazilian Real), *language*(Portuguese)

These facts can be queried simultaneously for answering complex questions like: ‘Which Portuguese-speaking country uses Real as their currency?’. Constructing such a KB, however, is expensive. Alternatively, Open-IE systems extract information at scale with significantly lower effort. They offer light-weight structuring of text, which is easier to derive than schema elements in a curated KB and is easier to query compared to the raw text. Inspired by these potential benefits, the first Open-IE systems were designed towards emitting triple facts, such as:

Sentence: Brazil was officially discovered in 1500.

Open-IE tuple: ⟨Brazil; be discovered in; 1500⟩

However, as facts become complex, triple representation begins to lose contextual information (e.g., arguments of an n-ary relation, links between nested relations) or fuse it in lengthy arguments. Neither of these is suitable for downstream tasks.

Sentence: Portuguese colonized Brazil under the direction of Pedro Alvares Cabral.

Open-IE tuple: ⟨Portuguese; colonized; Brazil⟩

Such a tuple can provide a precise answer to a question ‘Who colonized Brazil?’, but not ‘Who commanded the colonization of Brazil?’. To preserve granularity of information captured and minimize the loss of information, we propose to factor complex facts into simple facts and to store them as interconnected triples, which we call nest-tuples. In nest-tuples, arguments can optionally be references to other triples. This makes the representation more expressive for complex facts. We designed an Open-IE system, NestIE, that can extract nest-tuples facts from the text.

The vast majority of previous work on Open-IE extracts structured information from individual sentences. However, there are alternative sources of textual data that provide rich information to build knowledge bases. One such source is conversational question-answer (cQA) data. Often, cQA data contains precisely the knowledge that users care about and can provide a goal-directed method for constructing knowledge bases. For example, many community question answering websites answer questions which may require local knowledge or particular expertise. The information required to answer these questions may not be present elsewhere in web documents. Following examples illustrate the kind of data we aim to extract:

Question: What does Brazil export?

Answer: Major export is seafood.

As can be seen from the example, harvesting facts from cQA data presents significant challenges because now the information is scattered across multiple sentences.

An extractor must interpret information collectively between the questions and answers to find useful tuple facts. Without knowing the question, an extractor can often either fail to or incorrectly interpret the answer. We designed an end-to-end Open-IE system, NEURON that can extract fact triples from utterances in cQA data.

1.1.2 Heterogeneity in Extracted Knowledge Bases

Knowledge bases, curated or extracted, use large schemas and terminologies to describe the information they contain. Consequently, it is tedious for a user, who lacks detailed knowledge about the structure of the data, to access the data in the KBs. On the other hand, users are more comfortable expressing their information needs in natural language. A controlled natural language query interface can hide the complexity from the user while maintaining expressivity. The idea of querying structured databases with meaning representation languages that rely on natural language is not new. However, it is still an active area of research owing to the challenges imposed by ambiguity and variability in natural language.

Such a natural language query interface is both a requirement and a viable solution to access an extracted KBs. Assertions expressing the same real-world fact can have different representations, with differences in the use of vocabulary and schema. These differences arise from variations in input textual data and in knowledge representation of Open-IE extractors (e.g., tuple, nary-tuple, nest-tuple). Formulating a query or expanding an input query to match all possible representations is non-trivial and challenging. The complexity of this task grows as queries become complex. Consider the following question, its query and relevant facts from an extracted KB:

Question: Which countries import fish from Brazil?

Query: $\langle ?x; \text{import fish from}; \text{Brazil} \rangle \langle ?x; \text{is-a}; \text{country} \rangle$

Relevant Facts: $t_1 \langle \text{Brazilian fish}; \text{is imported by}; \text{USA} \rangle,$

$t_2 \langle \text{USA}; \text{imports}; \text{fish}; \text{Brazil} \rangle,$

$t_3\langle\text{Brazil; supplies; fish}\rangle, t_4\langle:t_3; \text{to; United States}\rangle,$
 $t_5\langle\text{USA; is a; country}\rangle$

Exactly matching the query specification in this case will yield no results because the tuple facts differ in textual relations (e.g., **supplies** vs **imports**), implicit constraints (e.g. **Brazilian fish**) and structured representations (e.g., **Brazil** is an *arg* in the query but a *subj* in tuple t_3). Even though it is possible to learn query transformations offline, such an approach is not suited for complex questions where the queries can have several possible transformations.

A major impediment in harnessing the broad coverage of an extracted KB to answer complex natural language questions is its massive, heterogeneous schema. However, since the arguments and relations in an extracted KB are strings in natural language, they can naturally be queried by string matchings (e.g., relation phrase **import fish from** and textual relation **is imported by** have several tokens in common). We designed a schemaless querying framework, NESTIQUE, which does not require a precise query to match the heterogeneous fact representations. Instead of matching a query as a whole, it matches sub-components of the query and collects evidence which could be different representation than the query specification. It then finds answers by reasoning over the collective evidence. Such an approach is flexible to representation model of the extracted KB.

1.1.3 Noisy and Non-Canonicalized Facts in Extracted Knowledge Bases

While a KB-QA system can benefit from the broad coverage of information in an extracted KB, the information is not as accurate and precise as the information curated manually. This is because a lot of effort has been put at curation to make sure the information is accurate and also pliable for compositional reasoning. Intuitively, curated information when available should be preferred over extracted information for answering complex, compositional questions. As such, a superior strategy to design

a KB-QA system is to combine high-quality curated knowledge with broad-coverage extracted facts.

Combining information from multiple sources offers two benefits: evidence scattered across a curated and extracted KB can be aggregated, and evidence from different KBs can be used to complement each other. Consider the example complex question Q_3 : 'What college did the author of *The Hobbit* go to?'. Inference over ontological relation *books.written* from a curated KB can benefit from textual relation *written by* from an extracted KB. On the other hand, evidence matching for *college attended* may exclusively be in the curated KB.

One possible solution to answer complex questions over multiple knowledge bases is to break down inference into multiple simple queries, each targeting a specific KB. The complex query can then be constructed by joining the simple queries. This framework is flexible to target a specific KB exclusively or different types of KB. We study this *decompose-execute-join* querying formalism in two scenarios: background knowledge source is a curated KB (with our system TEXTRAY), and background knowledge source includes a curated and an extracted KB (with our system MULTIQUE). Both our systems use a neural-network based model that can learn to find simple queries using implicit supervision available in the form of answers to complex questions.

1.2 Summary of Contributions

This dissertation studies the challenges in enabling KB-QA systems to answer complex questions in natural language. To answer complex questions, such a system first needs to acquire and represent knowledge from the raw text in a structured format that supports faster querying and inference. To minimize the loss of information at extraction and maintain the granularity of knowledge, we present a scalable, open-domain information technique that encodes the knowledge in a nested format. Additionally, we briefly describe techniques to extract information when it could be

scattered across multiple sentences.

To access the knowledge stored in several such automatically extracted KBs, we propose a schemaless querying mechanism that given one structural query for a natural language question can derive answers from relevant assertions with different knowledge representations.

Lastly, we propose a query mechanism for a KB-QA system that benefits from high-quality knowledge of curated KBs and broad-coverage of extracted KBs. To leverage information from multiple KBs, we construct query patterns for complex questions using simple queries, each targeting a specific KB. We propose a neural-network based model that finds simple queries using implicit supervision from answers to complex questions. We describe techniques to further equip the model for performing collective inference over diverse relation forms from multiple KBs.

1.3 Outline of the Dissertation

The rest of the dissertation is organized as follows. In chapter II, we survey related information extraction and question answering research concerning the automatic acquisition and querying of knowledge from textual data. In chapter III, we describe our open information extractor, NESTIE, for extracting nest-tuples from natural language text. We additionally describe our open information extractor, NEURON, for extracting tuples from conversational utterances. In chapter IV, we describe a schemaless querying technique, NESTIQUE, for answering complex questions over an extracted KB. In chapter V, we describe a KB-QA system, TEXTRAY, for answering complex questions over a curated KB. In chapter VI, we describe a KB-QA system, MULTIQUE, for answering complex questions over a curated and extracted KB. Finally, we outline the future work directions and conclude the dissertation in chapter VII.

CHAPTER II

Related Work

While our proposed solution techniques are new, and based on novel inter-subdisciplinary approaches, the problems we seek to address have a long history of excellent work, which we briefly survey below.

2.1 Question-Answering

There are two major paradigms of question answering: information-retrieval-based methods and knowledge-based methods. IR-based question answering systems rely on the textual information on the web. Given a question in natural language, they use information retrieval techniques to find relevant documents and passages and then use reading-comprehension algorithms to read the retrieved passages and identify an answer directly from the text. In knowledge-based question answering, a system maps a question to a logical representation which is then used to query structured databases of facts. We describe IR-based approaches in the next section, followed by sections of knowledge acquisition and knowledge-based question answering approaches.

2.2 Querying Text

The goal of IR-based question answering is to answer a users question by finding fragments of text in a collection of documents. Traditionally, corpus search approaches

[29, 19] were used to query the syntactic parse trees from the text using tree-pattern queries. Alternatively, other semantic abstractions over the text such as abstract meaning representations [5] and semantic views [52] have also been used.

The most popular formulation of IR-based question answering is reading comprehension. There has been considerable progress in reading comprehension models, owing to the success of new learning architectures like attention-based and memory-augmented neural networks [7, 94]). Several new large-scale training datasets like SQUAD [72] and WikiReading [47] based on Wikipedia have been released. Many successful reading comprehension systems [27, 76, 37] have been developed. These systems compute an embedding for the question and an embedding for each token in the text, and then select spans whose embeddings are close to the question embedding. The systems are trained end-to-end. Obtaining such training data, however, can be expensive. As a result, often the models do not generalize beyond their training domains [95, 38]. Furthermore, the questions that appear in reading comprehension tend to focus more on deep understanding of the retrieved passage, including coreference resolution and textual entailment. The techniques proposed in this dissertation are not designed to handle this level of text understanding.

There are a number of other QA approaches that use either the Web or Wikipedia as the background knowledge. This includes systems such as AskMSR [23], QuASE [83], DeepQA [45] and YodaQA [14]. Most of these systems are very sophisticated, relying on carefully curated ontologies and linguistic analyses of questions and candidate answers. Instead, our focus is on providing an easy-to-use natural language query end-point to textual data, which with little training can support complex information needs. Satisfying these information needs can often require the ability to do reasoning across multiple documents/passages, which is challenging over unstructured data [86]. We, therefore, adopt a knowledge-based question answering approach in this dissertation to answer complex questions.

2.3 Knowledge Acquisition

Information available as natural language text in open-domain corpora such as from the Web and Wikipedia becomes much easier to query and analyze if it is stored in a structured database. Therefore, many have studied how to build large-scale knowledge bases (KBs) of facts about real-world entities, and relations between them, from natural language text. There are two main approaches to construct knowledge bases, namely curation and extraction. Curated KBs are typically constructed collaboratively using manually-defined rules. A curated KB has a fixed schema, defined by its ontology. Extracted KBs are constructed using information extraction systems that transform the text into structured facts. The two KBs differ in comprehensiveness and accuracy. Curated KBs have precise information about certain domains, but lack information about others. Extracted KBs can have broad coverage about a variety of domains, but are prone to errors from extraction.

Several initiatives have been taken over the years to leverage the online community for curating high-quality knowledge bases. Cyc [57] is one of the first efforts to assemble open-domain, commonsense knowledge using a formal ontology. DB-Pedia [6] is constructed with information from Wikipedia infoboxes. YAGO [82] is constructed using Wikipedia infoboxes and WordNet. Freebase [20] is constructed using an interactive tool for editing the ontology and writing new facts into the KB. Wikidata [89] is yet another collaboratively edited knowledge base that can be read and edited by both humans and machines. Other prominent endeavors include NELL [26], KNOWITALL [39], DEEPDIVE [67]), and GOOGLE KNOWLEDGE GRAPH [78].

The fixed schema and inherent incompleteness of curated knowledge bases led researchers to consider completing the knowledge base with little human effort. One approach is to predict the missing relations in the KB, also known as knowledge base completion. Several techniques [93, 92, 62] have been proposed recently to learn embeddings for entities and relations in the KB to identify the missing relations/facts.

However, these are limited by the relations described in the ontology of the KB. The most promising approach to identify open-domain relations and facts is to extract them from the text. Traditionally, information extraction systems learned an extractor for each target relation from labeled training examples [79, 73]. Such an approach does not scale when the number of target relations is large, or where the set of target relations cannot be specified in advance. Several efforts have been made to reduce the manual labor such as by requiring only a small set of labeled instances per relation [24, 2]. However, substantial expertise is required in obtaining training data, making them unsuitable for large-scale extraction.

Relaxing the limitations of the pre-specified ontology of relations and labeled corpora has received considerable attention in the last few years. The extraction systems can be broadly classified into Preemptive IE [77], weak supervision for IE [65, 49] and Open-domain IE [8, 96]. Preemptive IE eliminates the need for relation-specific extractors but relies on clustering entities and documents which is expensive to do at a web-scale. Distant supervision methods incorporate supervision from curated KBs, such as Freebase [20], to generate training data for learning relation-specific extractors. Even though these approaches drastically reduce the manual effort, the extracted information is limited to use the relations from the curated KB.

Open-IE systems require no relation-specific training data or pre-specified ontology of relations. These systems make a single extraction pass over the corpus and automatically discover relations of interest using shallow syntactic features and extract tuples from sentences, each comprising a relation phrase and arguments related by that relation. Pioneered by `TEXTRUNNER` [105] that used hand-written rules to extract binary relations mediated by verbs, subsequent works [96, 41, 28, 40, 36, 101, 4, 12] extended the scope of relations to a wider range of syntactic entities. However, the first-generation Open-IE systems [105, 41, 36] suffer from two key major drawbacks: a) they miss out additional attributes such as reification, conditionals, and multiple

arguments a relation may have, b) they determine argument and relation boundaries heuristically which often results in loss of granularity in information captured. Neither of these is suitable for downstream tasks such as answering complex questions.

Instead of triples, n-ary tuple representation has been used [3, 28], but this representation still cannot capture contextual information such as attributions, conditionals, and beliefs. OLLIE [75] made the first attempt by additionally extracting an attribution contextual information with a tuple, when available. Semantic role labeling (SRL) has also been used to discover multi-verb relation phrases in semantic frames, and construct binary, n-ary, and nest-tuples [28]. However, SRL relations do not always correspond to Open-IE relations (e.g., SRL identifies ‘took’ as a relation from a sentence ‘The event took place in the Levis stadium.’ whereas an Open-IE system would identify ‘took place in’ as the relation phrase). Recently, short entailment and clusters of semantically related constituents from longer utterances have been used to address the problem of long and uninformative argument and relation phrases [11, 4]. These methods reduce the clusters to triples by mapping them to known relation types or by using a set of hand-crafted rules.

Since we target answering complex questions, we need an extraction system to minimize loss of information because of inexpressive representation without sacrificing granularity in extracted information. Our techniques described in chapter III are inspired by the use of expressive representations in closed-domain extraction [66, 11, 82] to capture temporal, geospatial and prepositional context information. Our bootstrapping and pattern learning approaches is motivated from a large body of work - DIPRE [24], SNOWBALL [2], NELL [26], and OLLIE [75] - that bootstrap training data based on seed instances of a relation and then learn patterns for extraction.

The vast majority of the previous work on Open-IE extracts structured information (e.g., triples) from individual sentences. However, they struggle to extract useful information from more complex input formats, such as conversational question-

answer data. Motivated by the recent success in Open-IE systems based on end-to-end frameworks, such as sequence tagging [81] or sequence-to-sequence generation [30], we briefly describe an end-to-end framework to extract information from question-answer pairs in chapter III.

2.4 Querying Knowledge Bases

Given a knowledge source, a knowledge-based question answering (KB-QA) system has to learn how to map natural language questions to queries that can be executed over the knowledge source. How a system performs this task depends largely on the type of the knowledge source, i.e. curated or extracted. We summarize the two types of KB-QA systems in the next sections.

2.4.1 Querying Curated Knowledge Bases

The goal of a KB-QA system that uses a curated KB as the background knowledge source is to formulate queries that align with the fixed schema of the KB [87, 59, 100, 25, 16, 55, 9, 113, 69, 21, 74]. The main challenge in answering questions in this paradigm is learning how to map textual phrases from the question to ontological relations in the KB. Typically, systems would learn these mappings from a training set of question-answer pairs.

KB-QA systems using a curated KBs can be broadly classified into three categories: retrieval-based methods, template-based methods, and semantic parsing-based methods. Most of these approaches target simple questions involving a single relation in the KB. The retrieval-based methods rely on information extraction techniques to answer natural language questions. Typically, a set of candidate answer entities from the KB are retrieved using relation extraction [104, 44] or distributed representations [22, 99], based on which the final answer entity is identified. These methods cannot handle compositional questions that require identifying multiple entities and

relations.

Template-based methods [58, 87, 1, 31] map the input question into a query using templates. Traditionally, templates were defined manually to handle complex query logic [87, 113], but these suffered from limited coverage. More recently, templates can be learned automatically from question-answer pairs [1]. Complex questions are answered using a series of binary factoid questions answerable using the learned templates.

Semantic parsing-based methods aim to learn semantic parse trees or equivalent query graphs representing the semantic structures of the questions. Conventional approaches [61, 16, 15, 91] relied on domain-independent semantic structures to generate logical forms that could be transformed into queries over the KB. While the semantic structures could be complex, these methods struggle to correctly map them to relations in the KB to construct complex queries. Recent advancements in neural-network models have revived the interest in semantic-parsing methods. These methods [63, 106, 109] first collect candidate queries using bottom-up parsing or staged query generation, and then rank them based on their semantic similarity to the question. The semantic similarity function relies on continuous representations of question and query candidates and is learned using question-answer pairs. These methods have shown great promise in answering simple questions that target one main relation in the KB.

Obtaining fully annotated queries for complex questions is expensive. Complex questions can, however, be answered by decomposing complex intents into multiple related simpler questions [51, 85]. Learning a semantic parser to answer the simple questions will require supervision in the form of answers to the simple questions. Obtaining answers for each simple question related to a complex question can be cumbersome. In chapter V, we describe techniques that can learn a semantic parser using only answers to the complex question. Motivated by the previous works, we

focus on answering a complex question using a series of simple queries, each matching one main relation at a time. This approach is broadly related to structured output prediction [70] and path finding [98, 34] methods that learn to navigate the search space of queries using question-answer pairs as the evidence. These work well on simple questions where the search space is small, and the search quality can be reliably approximated. However, when the queries become complex, the search space for finding correct paths grows, and the supervision signals become sparse [60].

2.4.2 Querying Extracted Knowledge Bases

KBs constructed with open information extraction tend to have information about a broad set of topics, but tend to use textual relations that are unnormalized natural language, in contrast to ontological relations in a curated KB. As a result, an extracted KB tends to contain more redundant, noisy facts than a curated KB. This can be understood as a trade-off between generality and power of knowledge representation. Ontological relation representation in a curated KB naturally supports compositional queries, but incomplete knowledge limits its generality. Textual relation representation in an open KB is harder to query but can support open-domain questions.

Owing to the unnormalized, heterogeneous knowledge representation, querying an extracted KB is fundamentally different from querying a curated KB. PARALEX system [42] was the first KB-QA system to operate over an extracted KB containing only triple facts. It is based on lexicons, learned from a question paraphrase corpus, that directly maps questions to triple queries against the extracted KB. A KB-QA system, however, can achieve higher robustness to variability in natural language questions and heterogeneity in the extracted KB by using a pipeline of operations for question paraphrasing, parsing, query rewriting, and execution [43]. Powered by triple facts and simple query language, early KB-QA systems struggle to answer a

question with complex semantic constraints.

The complexity of questions that can be supported over an extracted KB requires a) a more expressive knowledge representation in the KB, and b) a querying mechanism that can aggregate evidence across multiple facts in the KB [53, 108]. However, most existing KB-QA systems assume a fixed representation of the facts in the KB (triple or n-ary) and use a query language tailored for that representation. As a result, they can only find answers from facts that match the query specification. Since this can result in a low recall, these systems rewrite or relax the query to match the facts. While this works for simple questions, learning query transformations for complex questions can quickly lead to combinatorial explosion. In chapter IV, we describe schemaless querying techniques that do not assume a fixed knowledge representation model and can find answers even when facts do not exactly match the query specification.

2.4.3 Querying Multiple Knowledge Sources

Over time, the two streams of knowledge base question answering - KB-QA on extracted KBs derived from unstructured data and KB-QA on curated KBs - have evolved rather independently. An important but under-explored KB-QA paradigm is where multiple knowledge sources are exploited together [100, 43, 110, 33]. Such combination is attractive because curated KBs have precise, compositional facts and the extracted KBs contain millions of facts not present in the curated KBs. A lot of KB-QA systems rely on the raw text itself instead of the extracted information. [32] uses memory networks and universal schema to support inference on the union of KB and text. [84] enriches KB subgraphs with entity links from text documents and formulates KB-QA as a node classification task. The key limitations of these methods are that a) they cannot handle highly compositional questions and b) they ignore the relational structure between the entities in the text.

Most of the existing state-of-the-art systems [83, 99] use one primary knowledge source, typically a curated KB. They use the secondary KB to improve the quality of answers derived from the primary KB. This is effective when the primary KB contains sufficient information to answer the question. Generalizing to questions which can not be answered using a single knowledge source alone requires a uniform query model [43] or a uniform structured representation [33] over the two types of KBs. The techniques described in chapter VI build upon these ideas of collective reasoning over different relation forms. However, we not only exploit any complementary evidence across the KBs but also combine evidence scattered across multiple KBs.

CHAPTER III

Identifying Facts in Text with Open Information Extraction

Conventional Open-IE systems [105, 41] were designed towards emitting binary tuples from individual sentences. Such binary representation isn't always sufficient for expressing factual information, especially for higher-order relations and nested facts. Often only a part of the information is captured, resulting in tuples that are incomplete, uninformative or incoherent. Consider Example 1 in Table 3.1, where contextual information is either ignored or is subsumed in over-specific argument and/or relation phrases. Such errors are difficult to repair in a post-processing step and affect downstream applications like knowledge-based question answering that rely on correctness and completeness of the tuples.

Furthermore, the extraction patterns used by existing Open-IE systems to generate the tuples only identify relations mediated by verbs or a subset of verbal patterns. However, relations in the text can be mediated by nouns and have long-range dependencies that cannot be captured using a small set of patterns [75, 68]. For instance, consider Example 2 in Table 3.1 that asserts facts, 'Rozsa Hill is the third hill near the river', 'Rozsa Hill is Rose Hill' and 'Rozsa Hill lies north of Castle Hill', which are not mediated by verbs. A verb-mediated extraction pattern, such as in REVERB, would extract tuple A_1 , which is not useful for answering questions about the location of

1. After giving 5,000 people a second chance at life, doctors are celebrating the 25th anniversary of Britain’s first heart transplant.	
REVERB	A ₁ : ⟨doctors; are celebrating the 25th anniversary of; Britain ’s first heart transplant⟩
OLLIE	A ₁ : ⟨doctors; are celebrating; the 25th anniversary of Britain’s first heart transplant⟩
NESTIE	A ₁ : ⟨doctors; are celebrating; the 25th anniversary of Britain’s first heart transplant⟩
	A ₂ : ⟨doctors; giving; second chance at life⟩ A ₃ : ⟨A ₁ ; after; A ₂ ⟩
2. Rozsa (Rose) Hill, the third hill near the river, lies north of Castle Hill.	
REVERB	A ₁ : ⟨the third hill; lies north of; Castle Hill⟩
OLLIE	A ₁ : ⟨the third hill; lies north of; Castle Hill⟩
NESTIE	A ₁ : ⟨Rozsa; lies; north of Castle Hill⟩
	A ₂ : ⟨Rozsa Hill; is; third hill near the river⟩ A ₃ : ⟨Rozsa Hill; is; Rose⟩
3. A senior official in Iraq said the body, which was found by U.S. military police, appeared to have been thrown from a vehicle.	
REVERB	A ₁ : ⟨Iraq; said; the body⟩
	A ₂ : ⟨the body; was found by; U.S. military police⟩
OLLIE	A ₁ : ⟨A senior official in Iraq; said; the body which was found by U.S. military police⟩
NESTIE	A ₁ : ⟨body; appeared to have been thrown; ∅⟩
	A ₂ : ⟨A ₁ ; from; vehicle⟩
	A ₃ : ⟨A senior official in Iraq; said; A ₂ ⟩
	A ₄ : ⟨U.S. military police; found; body⟩

Table 3.1: Extracted tuples from Open-IE systems: REVERB, OLLIE and NESTIE.

‘Rozsa Hill’. In contrast, tuple ⟨Rozsa; lies; north of Castle Hill⟩, is more informative but is not mediated by a verb in the original sentence. Focused on extracting verb-mediated facts, much factual information from the sentence is often lost. For instance, A₁ from REVERB cannot answer questions of the form ‘What is the other name of Rozsa Hill?’, ‘Where is Rozsa Hill located?’ and ‘Which is the third hill near the river?’. On the other hand, these questions can be answered from a more complete set of tuple facts {A₁, A₂, A₃} from NESTIE.

Another problem faced by Open-IE systems is that they tend to extract tuples with too large argument or relation phrases, in an attempt to encode information as binary tuples and to achieve high recall. Such tuples are difficult to fuse, link and

aggregate for downstream applications. For instance, the argument phrase, ‘body which was found by U.S. military police’, is less likely to be useful than the argument phrase, ‘body’ in Example 3 in Table 3.1.

Intuitively, complex facts can be factored into a set of interconnected tuples.

Definition 1 (Nest-Tuple). A nest-tuple representation is of the format $t_{i,j} = \langle e_i, r_{i,j}, e_j \rangle$ where e_i and e_j are strings representing entities or tuple references and $r_{i,j}$ is the relation connecting e_i and e_j .

This representation can potentially solve the problem of loss of information. To identify nest-tuple facts from the text, we need to learn extraction patterns that can map the many ways of expressing complex relations in the text to corresponding nest-tuple representations. In practice, it is infeasible to simply enumerate all different extraction patterns as relations, especially 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.

We propose an Open-IE system, NESTIE, which uses a bootstrapping algorithm that can learn broad-coverage relation-independent patterns for complex sentence constructions given a seed set of extraction patterns. We demonstrate that these patterns can be learned from a limited amount of data containing sentence equivalence pairs. To tackle the problem, NESTIE uses a multi-stage solution:

- a) construct a seed set of tuples with little or no nesting,
- b) bootstrap sentences that mention the seed facts and learn extraction patterns,
- c) extract tuples from unseen sentences using learned patterns,
- d) link extracted tuples to capture any missing information not captured previously using the learned patterns.

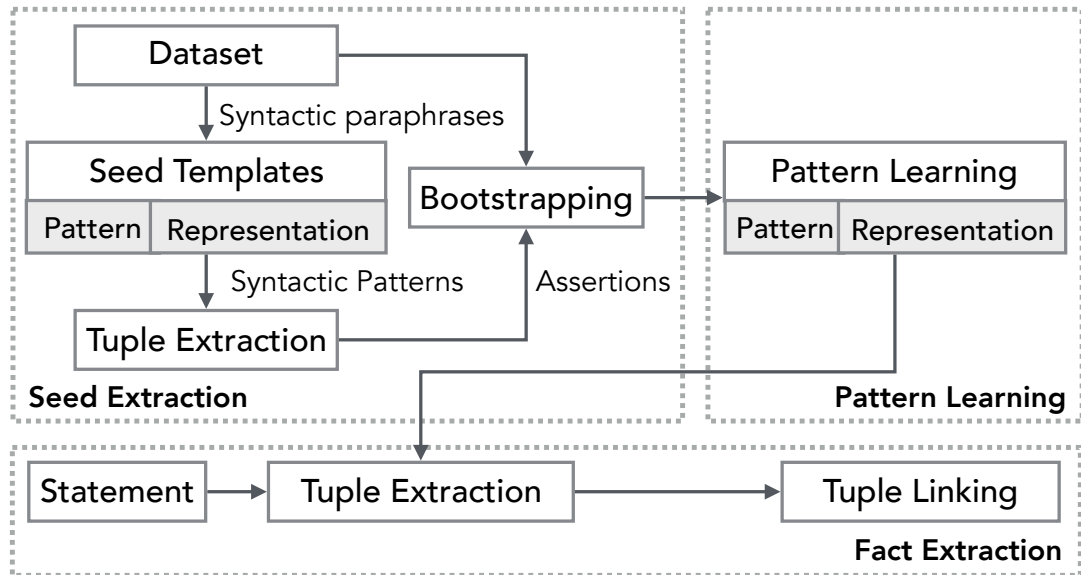


Figure 3.1: Pattern learning and tuple extraction for nest-tuples in NESTIE.

3.1 NestIE

Figure 3.1 illustrates the system architecture of NESTIE. Training includes creation of seed facts, bootstrapping and pattern learning over dependency parse-trees. First, a set of high-precision seed templates is used to extract tuples.

Definition 2 (Template). A template maps a dependency parse-tree pattern to a tuple representation: a triple such as $\langle \text{arg1}; \text{rel}; \text{arg2} \rangle$ for capturing binary relations, a nest-tuple such as $\langle \langle \text{arg1}; \text{rel}; \text{arg2} \rangle; \text{rel2}; \text{arg3} \rangle$ for capturing n-ary relations. Arguments in the templates are treated as a sequence of words, such as $[\text{arg2} \text{ rel2} \text{ arg3}]$, to capture nominal modifiers.

NESTIE bootstraps over a textual entailment dataset to learn equivalent parse-tree patterns for the templates in the seed set. In the extraction phase, NESTIE performs pattern matching using learned templates and parse-based expansion to construct tuples. These extracted tuples are then linked to generate more complete tuple facts.

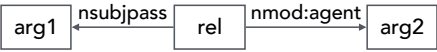

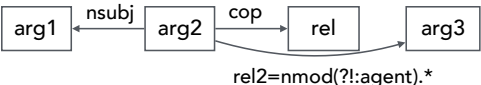
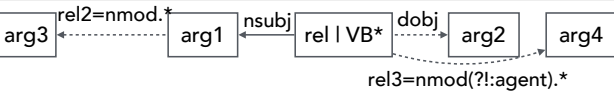
Template	Example
Pattern: 	A body has been found by police.
Representation: T: $\langle arg1; rel \text{ by}; arg2 \rangle$	$\langle \text{body}; \text{found by}; \text{police} \rangle$
Pattern: 	Fallujah is an Iraqi city.
Representation: T: $\langle arg1; \text{be}; arg2 \rangle$	$\langle \text{Fallujah}; \text{is}; \text{city} \rangle$
Pattern: 	Ghazi al-Yawar is president of Iraq.
Representation: T: $\langle arg1; \text{be}; (arg2, rel2, arg3) \rangle$	$\langle \text{Yawar}; \text{is}; \text{president of Iraq} \rangle$
Pattern: 	10,000 people in Africa died of Ebola.
Representation: T ₁ : $\langle (arg1, rel2, arg3); rel; arg2 \rangle$ T ₂ : $\langle T_1; rel3; arg4 \rangle$	A ₁ : $\langle \text{people in Africa}; \text{died}; \emptyset \rangle$ A ₂ : $\langle A_1; \text{of}; \text{Ebola} \rangle$

Figure 3.2: Seed templates and corresponding representation in NESTIE.

3.1.1 Constructing Seed Set

Since nest-tuples are not readily available, we first write a set of 13 templates, each encoding a sub-tree of the dependency parse connecting relation phrases and argument phrases of a sentence. A subset of these templates is shown in Figure 3.2. 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, therefore, iteratively create templates until at least one tuples could be extracted from each hypothesis. We then generate a seed set of tuple-facts by matching these templates against the set of hypotheses.

Using the seed set of tuples from the hypotheses, we can learn the different ways of expressing them in complex sentence constructions by referring to the statements

entailing the hypotheses. While a statement entailing a hypothesis share many words, there is a closed 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 and can be considered auxiliary. We ignore such words while constructing the seed set.

Example 1. Consider a statement-hypothesis pair,

Statement: Paul Bremer, the top U.S. civilian administrator in Iraq, and Iraq’s new president, Ghazi al-Yawar, visited the northern Iraqi city of Kirkuk.

Hypothesis: Ghazi al-Yawar is the president of Iraq.

The hypothesis is entailed in the statement. The seed templates extract the following tuples from the hypothesis: $\langle \text{al-Yawar; is; president} \rangle$, $\langle \text{al-Yawar; is; president of Iraq} \rangle$, and $\langle \text{al-Yawar; is president of; Iraq} \rangle$.

3.1.2 Extraction Pattern Learning

The biggest challenge in information extraction is the multitude of ways in which information can be expressed in unstructured text. When facts are complex, it is not possible to enumerate all the different syntactic variations of the fact. We, therefore, need to learn the various syntactic patterns that can encode the same information as the seed patterns and hence can be mapped to same representation.

We extend the bootstrapping techniques designed for binary-relations [75] to handle n-ary and complex relations. Our seed templates include patterns and corresponding representations for n-ary, complex relations (see 3.2). This allows us to learn dependency parse-tree patterns connecting the heads of all the argument and relation phrases in a seed template. Instead of learning different ways of encoding two arguments and a relation for a triple representation, we learn how all different arguments and relations in a template might be expressed. This allows us to achieve higher coverage of context for the facts and prevents the arguments/relations from


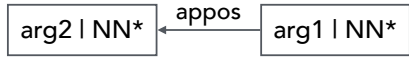
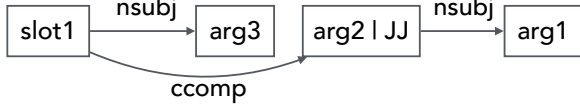
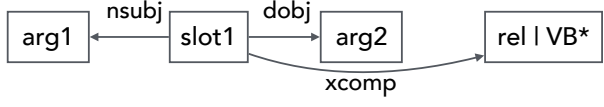
Template	Learned Patterns
Pattern:	
Representation:	T: <arg1; rel by; arg2>
Pattern:	
Representation:	T: <arg1; be; arg2>
Pattern:	
Representation:	T: <arg1; be; (arg2, rel2, arg3)>
Pattern:	
Representation:	T ₁ :<(arg1, rel2, arg3); rel; arg2>, T ₂ : <T ₁ ; rel3; arg4>

Figure 3.3: Example syntactic Patterns learned using bootstrapping.

being over-specified and/or uninformative. To mitigate sparsity while bootstrapping, we ignore the relations that are implicit (e.g., nominal modifier) and can be deduced from the type of dependency. 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 [35] to parse a statement and identify the path connecting the words of the corresponding tuples. If such a path exists, we retain the syntactic constraints on the nodes and edges in the path and ignore the surface forms of the nodes in the path. This helps generalize the learned patterns to unseen relations and arguments.

Example 2. Consider dependency parse-subtree of the statement and hypothesis from Example 1,

Statement: Iraq $\xrightarrow{\text{poss}}$ president $\xrightarrow{\text{appos}}$ al-Yawar

Hypothesis: al-Yawar $\xleftarrow{\text{nsubj}}$ president $\xrightarrow{\text{of}}$ Iraq

A seed extraction pattern maps the parse-tree of the hypothesis to the representation, $\langle \text{arg1}; \text{be}; \text{arg2} \rangle$, returning tuple, $\langle \text{al-Yawar}; \text{is}; \text{president of Iraq} \rangle$. With bootstrapping, the syntactic pattern from the statement is mapped to the same representation.

In this manner, NESTIE could learn 183 templates from the 13 seed templates. Figure 3.3 shows a subset of these patterns.

3.1.3 Tuple 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.

Example 3. In the statement from Example 1, the extraction patterns capture the dependency path connecting the head words: `Iraq`, `administrator` and `Paul Bremer`. However, to capture the contextual information, we need to further qualify the argument node, `administrator`.

Following this observation, 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`, `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.

3.1.4 Tuple Linking

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

a complex fact as a whole, NESTIE links the various tuples from the previous step to generate nest-tuples that are complete and closer in meaning to the original sentence.

We assume that the information to link the various extracted tuples can be inferred from the dependency parse-tree of the sentence from which the tuples were extracted. We use the following rules to link the tuples:

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

Consider the statement, ‘The accident happened after the chief guest had left the event.’ and tuples, T_1 : $\langle \text{accident}; \text{happen}; \phi \rangle$ and T_2 : $\langle \text{chief guest}; \text{had left, event} \rangle$. Using dependency edge `nmod:after`, we construct a nest-tuple $\langle T_1; \text{after}; T_2 \rangle$.

- Tuple T_1 is argument in tuple T_2 .

Consider the statement, ‘A senior official said the body appeared to have been thrown from a vehicle.’ and tuples, T_1 : $\langle \text{body}; \text{appeared to have been thrown from}; \text{vehicle} \rangle$ and T_2 : $\langle \text{senior official}; \text{said}; \phi \rangle$. We update T_2 to $\langle \text{senior official}; \text{said}; T_1 \rangle$.

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

We use dependency parse patterns to link tuples. We find correspondences between: a `ccomp` edge and a clausal complement, an `advcl` edge and a conditional, a `nmod` edge and a relation modifier. For clausal complements, a null argument in the source tuple is updated with the target tuple. For conditionals and nominal modifiers, a new tuple is constructed with the source and target tuples as arguments. The relation of the new tuple is derived from the target of the `mark` edge from the relation head of target tuple.

3.1.5 Comparison with Ollie

Our approach to learn dependency-parse tree based syntactic patterns is similar to that of OLLIE and WOE. However, there are significant differences. First, OLLIE and WOE rely on tuples from REVERB and Wikipedia info-boxes respectively for bootstrapping. Most of these relations are binary. On the contrary, we rely on high-confidence seed templates to construct seed set of tuples. These templates are more complex and expressive, allowing us to learn different ways in which a complex fact as a whole could be expressed. Though the arguments in OLLIE can be expanded to include the n-ary arguments, NESTIE encodes them in the seed templates and learns different extraction patterns for these arguments. Also, similar to OLLIE, NESTIE can extract tuples that are not just mediated by verbs.

3.2 Experiments

We conducted an experimental study to compare NESTIE to other state-of-the-art extractors. We found that it achieves higher informativeness and produces more correct and minimal tuples than other extractors.

3.2.1 Experimental Setup

We used two datasets released by [36] in our experiments: 200 random sentences from Wikipedia, and 200 random sentences from New York Times (NYT). We compared NESTIE against three OIE 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.

Next, to make the tuples comparable, we configured the extractors to generate triple tuples. REVERB and CLAUSIE tuples were available as triples by default. OLLIE extends its triple representation. So, we generated an additional tuple for each of the

Dataset		REVERB	OLLIE	CLAUSIE	NESTIE
	Informativeness	1.437/5	2.09/5	2.32/5	2.762/5
NYT	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)
	Informativeness	1.63/5	2.267/5	2.432/5	2.602/5
Wikipedia	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 3.2: Informativeness and number of correct and minimal tuples as fraction of total number of tuples.

possible extensions of a tuple. NESTIE uses a nested representation. So, we simply extracted the innermost tuple in a nested representation as a triple and allowed the subject and the object in the outer tuple to contain a *reference* to the inner triple. By preserving references the context of a tuple is retained while allowing for queries at various granularity levels.

We manually labeled the tuples obtained from all extractors to 1) maintain consistency, 2) additionally, assess if they were informative and minimal. Some extractors use heuristics to identify arguments and/or relation phrase boundaries, which leads to over-specific arguments that render the tuples unusable for other downstream applications. To assess the usability of tuples, we evaluated them for minimality [11]. Furthermore, the goal of our system is to extract as many tuples as possible and lose as little information as possible. We measure this as *informativeness* of the set of the tuples for a sentence. Since computing informativeness as a percentage of text contained in at least one extraction could be biased towards long tuples, we used an explicit rating scale to measure informativeness.

Two CS graduate student labeled each tuple for correctness (0 or 1) and minimality (0 or 1). For each sentence, they label the set of tuples for informativeness (0-5). A tuple is marked correct if it is supported in the text and correctly captures the contextual information. A tuple is considered minimal if the arguments are not over-

specified i.e. they don't subsume another tuple or have conjunctions or are excessively long. Lastly, they rank the set of tuples on a scale of 0-5 (0 for bad, 5 for good) based on the coverage of information in the original sentence. We measured the agreement between labelers as Cohens Kappa.

3.2.2 Experimental Results

The results of our experimental study are summarized in Table 3.2 which shows the number of correct and minimal tuples, as well as the total number of tuples for each extractor and dataset. For each dataset, we also report the macro-average of informativeness reported by the labelers. We found moderate inter-annotator agreement: 0.59 on correctness and 0.53 on minimality for both the datasets. Each extractor also includes a confidence score for the tuples. But since each extractor has its unique method to find confidence, we compare the precision over all the tuples instead of a subset of high-confidence tuples.

NESTIE produced many more tuples, and more informative tuples than other systems. There appears to be a trade-off between informativeness and correctness (which are akin to recall and precision, respectively). CLAUSIE is the system with results closer to NESTIE than other systems. However, the nested representation and tuple-linking used by NESTIE produce substantially more (1.7-1.8 times more) minimal tuples than CLAUSIE, which generates tuples 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 tuples. NESTIE achieves 1.1-1.9 times higher informativeness score than the other systems.

We believe that nested representation directly improves minimality, independent of other aspects of extractor design. To explore this idea, we conducted experiments on OLLIE, which does not expand the context of the arguments heuristically unlike

other extractors. Of the tuples labeled correct but not minimal by the annotators on the Wikipedia dataset, we identified tuples that satisfy one of: 1) has an argument for which there is an equivalent tuple, 2) shares the same subject with another tuple whose relation phrase contains the relation and object of this tuple, 3) has an object with conjunction. Any such tuples can be made minimal and informative with a nested representation. 73.75% of the non-minimal correct tuples met at least one of these conditions, so by a post-processing step, we could raise the minimality score of OLLIE by 17.65%, from 76.1% to 93.75%.

3.2.3 Error Analysis

We did an analysis of the errors made by NESTIE. We found that in most of the cases (about 33%-35%), extraction errors were due to incorrect dependency parsing. This is not surprising as NESTIE relies heavily on the parser for learning extraction patterns and linking tuples. An incorrect parse affects NESTIE more than other systems which are not focused on extracting finer grained information and can trade-off minimality for correctness. An incorrect parse not only affects the pattern matching but also tuple-linking which either fails to link two tuples or produces an incorrect tuple.

Example 4. Consider the statement, ‘A day after strong winds stirred up the Hauraki Gulf and broke the mast of Team New Zealand, a lack of wind caused Race 5 of the America’s Cup to be abandoned today.’. The statement entails following facts:

A₁: strong winds stirred up the Hauraki Gulf

A₂: strong winds broke the mast of Team New Zealand

A₃: a lack of wind caused Race 5 of the America’s Cup to be abandoned

A₁ and A₂ are parsed correctly. A₃ is parsed incorrectly with Race 5 as object of the verb caused. Some extractors either don’t capture A₃ or return an over-specified tuple, ⟨a lack of wind; caused; Race 5 of the America ’s Cup to be abandoned today⟩.

Such a tuple is correct but not minimal.

To maintain minimality, NESTIE aims to extract tuples, A_1 : ⟨Race 5 of the America’s Cup; be abandoned; ϕ ⟩ and A_2 : ⟨a lack of wind; caused; A_1 ⟩. However, it fails because of parser errors. It extracts incorrect tuple, A_3 : ⟨a lack of wind; caused; Race 5⟩ corresponding to A_3 and links it to tuples for A_1 and A_2 . Linking an incorrect tuple generates more incorrect tuples which hurt the system performance.

However, we hope this problem can be alleviated to some extent as the parsers become more robust. Another approach could be to use clause segmentation to first identify clause boundaries and then use NESTIE on reduced clauses. As the problem becomes more severe for longer sentences, we wish to explore clause processing for complex sentences in future.

Another source of errors was under-specified tuples. Since our nested representation allows null arguments for intransitive verb phrases and for linking tuples, failure to find an argument/tuple results in an under-specified tuple. We found that 27% of the errors were because of null arguments. However, by ignoring tuples with null arguments we found that precision increases by only 4%-6% (on Wikipedia). This explains that many of the tuples with empty arguments were indeed correct, and need special handling. Other sources of errors were: aggressive generalization of an extraction pattern to unseen relations (24%), unidentified dependency types while parsing long, complex sentences (21%), and errors in expanding the scope of arguments and linking tuples (20%).

3.3 Conclusions

We presented NESTIE, a novel open information extractor that uses nested representation for expressing complex relations and inter-tuple relationships. It can be seen as a step towards a system that has a greater awareness of the context of tuples

and provides informative, complete tuples for KB-QA systems.

3.4 Facts from Conversational Question-Answer Data

Most existing Open-IE systems focus on identifying facts from individual sentences, and ignore richer forms of textual data such as conversational question-answer (cQA). Following examples illustrate the kind of data:

Example 5. Q: Does the hotel have a gym?

A: It is located on the third floor and is 24/7.

Tuple: ⟨gym, is located on, third floor⟩

Example 6. Q: What time does the pool open?

A: 6:00am daily.

Tuple: ⟨pool, open, 6:00am daily⟩

As can be seen from these examples, harvesting facts from cQA data presents significant challenges. In particular, the system must interpret information collectively between the questions and answers. In this case, it must realize that ‘third floor’ refers to the location of the ‘gym’ and that 6:00am refers to the opening time of the pool. Open-IE systems that operate over individual sentences ignore the discourse and context in a QA pair. Without knowing the question, they either fail to or incorrectly interpret the answer.

We develop NEURON, an end-to-end system for extracting information from cQA data. We cast Open-IE from cQA as a multi-source sequence-to-sequence generation problem to explicitly model both the question and answer in a QA pair. We propose a multi-encoder, constrained-decoder framework that uses two encoders to encode each of the question and answer to an internal representation. The two representations are then used by a decoder to generate an output sequence corresponding to an extracted tuple. For example, the output sequence of Example 6 is:

$\langle arg_1 \rangle$ pool $\langle /arg_1 \rangle \langle rel \rangle$ open $\langle /rel \rangle \langle arg_2 \rangle$ 6:00am daily $\langle /arg_2 \rangle$

While encoder-decoder frameworks have been used extensively for machine translation and summarization, there are two key technical challenges in extending them for information extraction from cQA data. First, it is vital for the translation model to learn constraints such as, arguments and relations are sub-spans from the input sequence, output sequence must have a valid syntax (e.g., $\langle arg_1 \rangle$ must precede $\langle rel \rangle$). These and other constraints can be integrated as *hard* constraints in the decoder. Second, the model must recognize auxiliary information that is irrelevant to the KB. Since existing facts in the KB are representative of the domain of the KB, this prior knowledge can be incorporated as *soft* constraints in the decoder to rank various output sequences based on their relevance. NEURON uses a novel multi-encoder constrained-decoder method that explicitly models both the question and the answer of a QA pair. It incorporates vocabulary and syntax as *hard* constraints and prior knowledge as *soft* constraints in the decoder. A more detailed description of our system and experimental results can be found in [18]. NEURON can be seen as a complementary extraction system to NESTIE which can gather facts from conversational question-answer data on the web.

CHAPTER IV

Online Schemaless Querying of Extracted Knowledge Bases

In this chapter, we focus on the problem of massive, loosely-defined, heterogeneous schemas of *extracted* knowledge bases (KBs) that impose significant limitations on querying the ‘broad-coverage’ information they contain. In the context of complex queries, we study challenges in and provide techniques to answer *multi-constraint* questions in a natural language using an array of extracted KBs having different representations and expressiveness.

4.1 Introduction

While a real-world fact has a unique representation in a curated KB, it can have diverse representations in an extracted KB. In example a) in Figure 4.1, it is not evident if the two triples refer to the same entities or even have the same relationship.

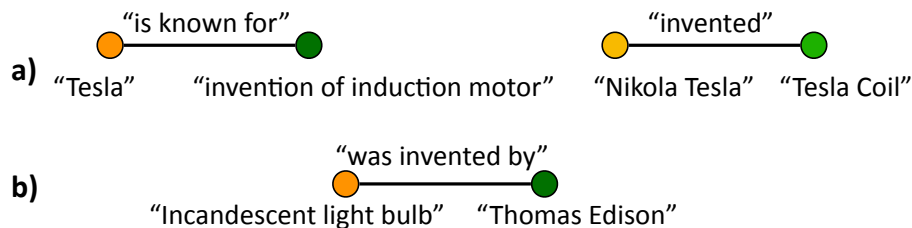


Figure 4.1: Facts in extracted KBs are heterogeneous

Facts in extracted KBs also exhibit variation in structure (such as n-tuple or nest-tuple) depending on the arity of the relation and contextual information [3, 75, 17]. 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 : $\langle \text{Tesla, originally invented, Radio, (in) 1897} \rangle$

nest-tuple t_2 : $\langle \text{U.S. Supreme Court, believed, } t_1 \rangle$

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 $\langle \text{Radio, was invented by, Tesla} \rangle$ and $\langle \text{Tesla, invented, induction motor} \rangle$.

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 4.1 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 will be far more complex if the user wanted to search for “inventions of Serbian inventors”.

Traditionally, possible query transformations and semantically equivalent structures are mined [111, 102] in an offline manner, which are then used to expand a given query at query-time. Such query expansion can be impractical for heterogeneous extracted KBs, since there will be a combinatorial explosion of expansion possibilities, particularly for complex queries.

Logical Query: $\langle \text{Tesla, invented, ?x, (in) 1891} \rangle \wedge \langle \text{?x, is-a, transformer} \rangle$

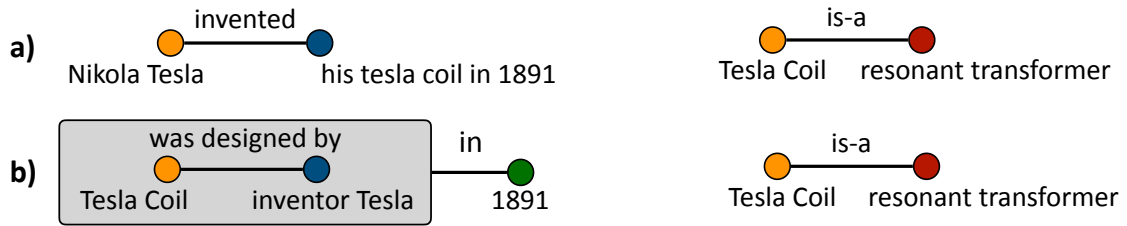


Figure 4.2: Heterogeneity in extracted KBs makes it difficult to access them via pattern matching

Example 7. Suppose the user wants to know “which transformer did Nikola Tesla invent in 1891?”. Figure 4.2 shows tuples from the extracted 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.

Instead of matching the entire query, finding matches for different query components is a much easier task. A simple keyword search [90, 56] 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.

Embodying these ideas, we propose an online *schemaless* querying method, NESTIQUE, for accessing extracted 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 that the recall is high. In the second *evidence aggrega-*

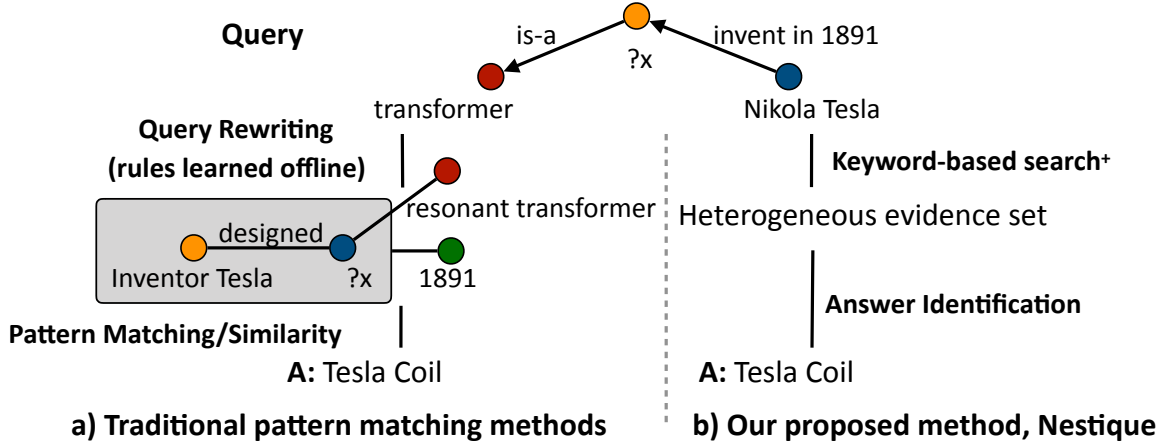


Figure 4.3: Pattern matching vs. our proposed method

tion phase, it uses more conservative reasoning to derive answers from the collective evidence. Figure 4.3 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 a 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 query. The multiplicity of components in the query and arguments in the evidence involves a complex alignment problem.

We make the following contributions in this chapter:

- We propose a generalized setting for querying extracted 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 extracted 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 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 [43] and TAQA [108], for querying extracted knowledge bases.

To the best of our knowledge, we are the first to develop a systematic framework for online and schemaless querying of heterogeneous extracted knowledge bases. NES-TIQUE is designed to help users having complex information needs access extracted 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 chapter is organized as follows. In Sec. 4.2 we introduce extracted KBs and formalize the problems studied in this chapter. We describe evidence gathering in Sec. 4.3 and evidence aggregation for querying extracted KBs in Sec. 4.4. We briefly discuss how we handle front-end engineering issues to make querying extracted KBs more effective in Sec. 4.5. We evaluate our proposed method in Sec. 4.6 and present a case study in Sec. 4.7.

4.2 Preliminaries and Overview

Extracted Knowledge Bases. An extracted 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 a n-tuple $\langle v_h; r; v_{t_1}, \dots, v_{t_n} \rangle$, 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 : $\langle \text{Tesla, originally invented, Radio, (in) 1897} \rangle$ and,

t_2 : $\langle \text{U.S. Supreme Court, believed, } t_1 \rangle$

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:

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

We call R_h the head of the rule, and R_1, R_2, \dots, 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 . Any variables in the atom bind to values in the KB. A query can have more than one $RRule$, but has a 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 8. A complex query, “which religious group settled near a river in 1638?”

Id	Subject	Relation	Arguments
What team did Jordan play for when he played baseball?			
	play_for(Jordan, ?x _Q , when play baseball), isa(?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), isa(?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 4.1: Example queries and evidence tuples in KBs.

can be posed as a set of *RRules*,

$$R_1(?x_Q, ?y) : \text{settled}(?x_Q, ?y) \text{ is_a}(?y, \text{river}) \text{ is_a}(?x_Q, \text{religious group})$$

$$R_2(?x_Q, ?y, 1638) : R_1(?x_Q, ?y), \text{settled_in}(?x_Q, 1638), \text{settled_in}(?y, 1638)$$

$$R(?x_Q) : R_2(?x_Q, ?y, 1638)$$

This query does not specify how to precisely match the relation atom `settled(?xQ, ?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 4.1 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 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 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

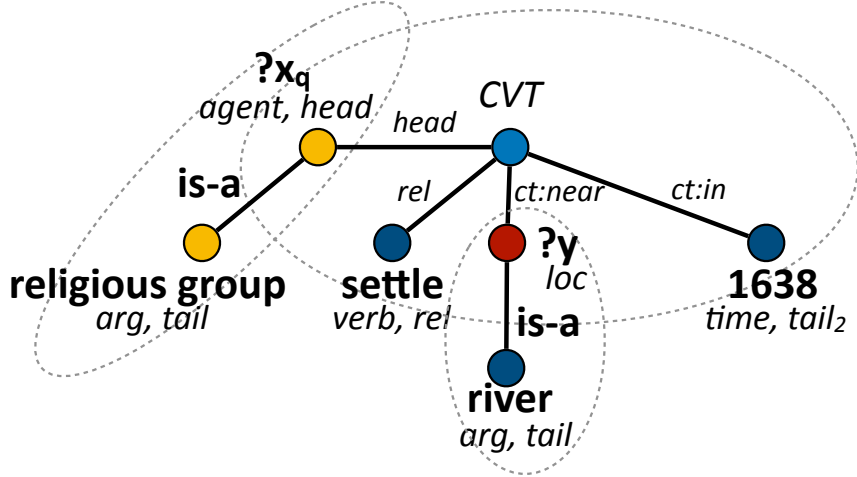


Figure 4.4: Example query graph and its sub-components

to the relation and the arguments. We also annotate the query graph components with information such as their semantic roles and lemmatized values.

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.

Figure 4.4 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 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.

4.2.1 Online Schemaless Querying

Figure 4.5 provides an overview of our proposed framework, NESTIQUE, for online, schemaless querying of heterogeneous extracted 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

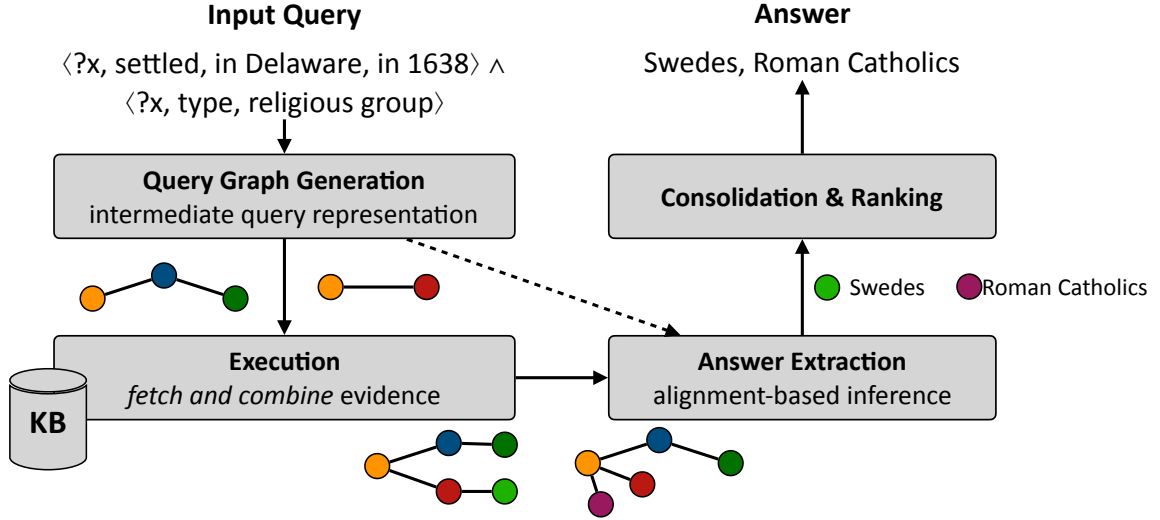


Figure 4.5: Overview of NESTIQUE schemaless querying.

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 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.

4.3 Evidence Gathering

4.3.1 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(?xQ, in Delaware, in 1638)` or `be_settled_by(Delaware, ?xQ, 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(?xQ, 1638)` is transformed to a relation `settle` and a constraint modifier `in` for arguments `?xQ` and `1638`. We also infer semantic roles for the query components using NLP4J¹. This helps identify the answer argument from the evidence. We encode all this information into a query graph, as illustrated in Figure 4.4.

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 the high heterogeneity of the underlying extracted KB, simply keyword-matching the query components will result in a low recall. For example, a query

```
keyword-match( $E_0.r$ , 'settle') AND  
keyword-match( $E_0.v_{t_1}$ , 'Delaware') AND  
keyword-match( $E_0.v_{t_2}$ , '1638') AND  
keyword-match( $E_1.r$ , 'is-a') AND
```

¹<https://emorynlp.github.io/nlp4j/>

keyword-match($E_1.v_{t_1}$, 'religious group') AND
string-similarity($E_0.v_h$, $E_1.v_h$) > 0.8

will match neither E_7 nor E_8 in Table 4.1. 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.

4.3.2 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

²<https://www.elastic.co>

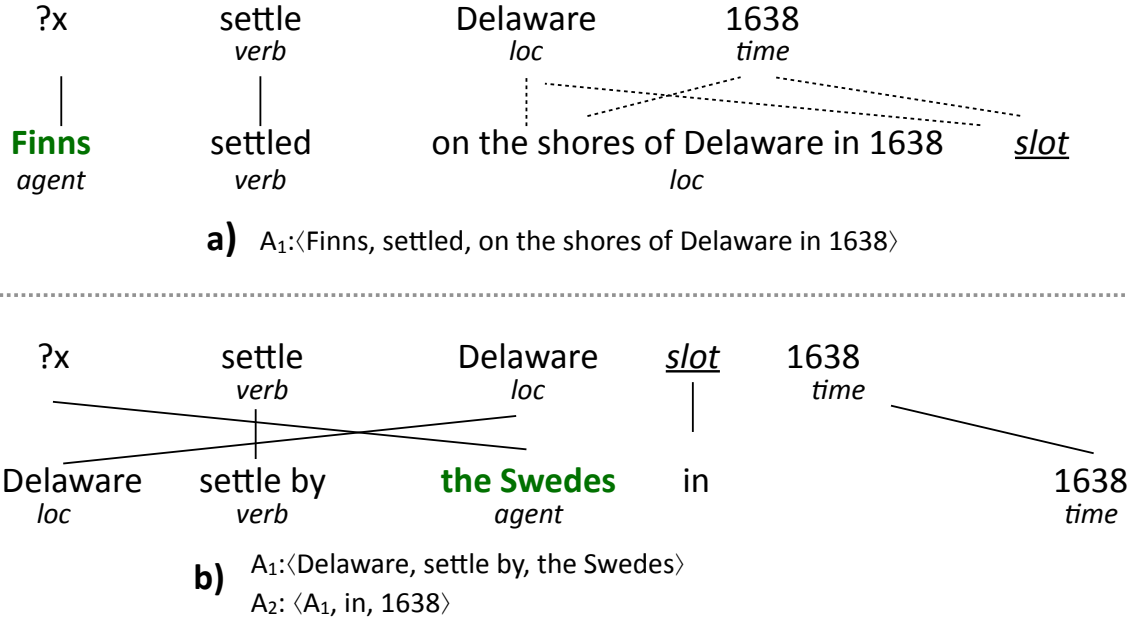


Figure 4.6: Alignment-based approach to extract answers from heterogeneous tuple representations

are found in the tuple.

We only retain top-50 of the relevant tuples due to the size of the extracted 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 $G(Q)$.

4.4 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 a 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:

- 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 4.6 shows two support sets in different formats for a single query. In contrast to existing methods [43, 108], we do not rewrite or relax the query to handle such mismatches. We handle mismatches by inferring over various items in a support set.

4.4.1 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 4.6 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. We instead, 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 ,

$$p_1 = 1 \text{ iff } (f_i \text{ is } ?x_Q) \wedge (c_j \text{ is-of } \textit{answer type})$$

$$p_2 = 1 \text{ iff } (f_i \text{ is } ?x_Q) \wedge (c_j \text{ is a noun phrase})$$

$$p_3 = 1 \text{ iff } (f_i \text{ is } ?x_Q) \wedge (c_j \text{ is a } v_h \text{ or } v_t)$$

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 .

4.4.2 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 [14] 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.

4.5 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).

4.5.1 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 extracted knowledge bases. To generate a structured query, a widely used approach is parsing the input natural language query into the

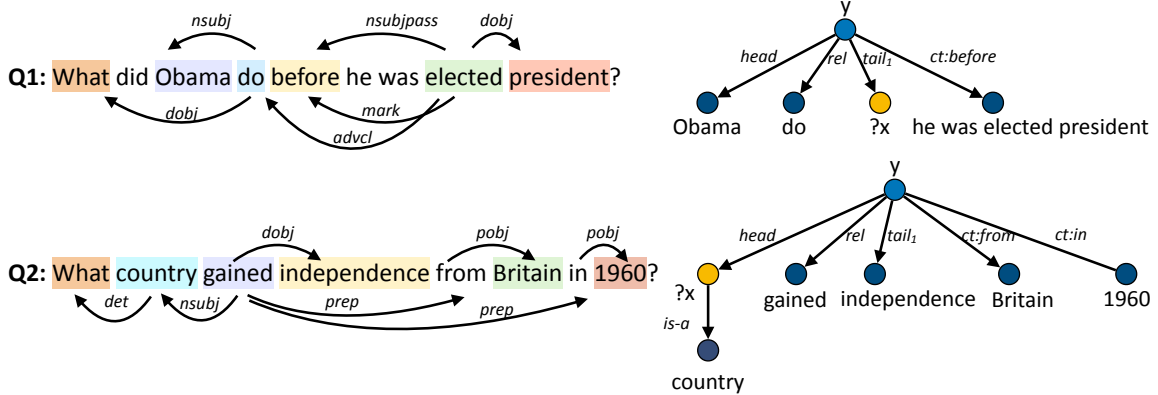


Figure 4.7: Example natural language queries, their dependency parse trees and query graphs

syntactic dependency representation by employing NLP tools such as the Stanford Parser [35]. Based on the parsing result, a query skeleton is constructed [58, 113, 104, 108] depending upon target data format (e.g., relational database or RDF).

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. Figure 4.7 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*.

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

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).

4.5.2 Paraphrasing

Users can formulate queries using informal and casual wordings and expressions. Queries having 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 extracted 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.

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 [42, 43] 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 4.7. 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.

4.6 Experimentals

We evaluate our proposed framework via empirical study in this section. Section 4.6.1 describes our experimental set up, query sets and evaluation metrics. We demonstrate effectiveness of our method by comparing with methods for querying extracted knowledge bases (Section 4.6.2) and curated knowledge bases (Section 4.6.3). We study how the components and KBs affect performance in Section 4.6.4.

4.6.1 System Settings

Extracted Knowledge Bases. We used several well-known extracted KBs.

- Open-IE [43] 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 [26] is a relatively small extracted KB with much fewer relation phrases. All facts are triples. NELL generally has high precision, but low recall.
- PROBASE [97] is an extracted KB with instances of only *is-a* relation extracted from 1.68B web pages. All facts are triples (e.g., $\langle \text{star-fruit, is-a, fresh fruit} \rangle$).
- NOKB [108] is an extracted 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 [17] is an extracted KB containing nest-tuple facts. Like NOKB these facts are extracted from web pages in Wikipedia and search snippets.

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 4.2: Extracted KBs used in our experiments

Table 4.2 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 extracted knowledge base. Table 4.3 shows statistics and example queries from these query sets.

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 does Jamaican people speak?
3778 train	who is Niall Ferguson’s wife?
2032 test	what did George Orwell die of?

Table 4.3: CompQ-T and CompQ-M have complex queries, WebQ has simple queries.

- **CompQ-T:** This query set was released by the authors of [108]. 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 extracted.
- **CompQ-M:** This query set was released by the authors of [10]. 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 [16]. 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 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 for curated KBs with querying methods for extracted 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 , recall R and F_1 scores of the querying methods:

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

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

- OQA [43] 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 [108] 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.

Systems	CompQ-T			CompQ-M		
	Precision	Recall	F_1	Precision	Recall	F_1
NESTIQUE	55.9%	47.0%	51.1%	31.5%	21.5%	25.6%
OQA	26.3%	1.6%	3.1%	25.6%	2.7%	4.9%
TAQA (No relaxation)	27.7%	27.7%	27.7%	-	-	-
TAQA	39.3%	39.3%	39.3%	-	-	-

Table 4.4: Performance of NESTIQUE and other systems on complex questions.

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.

4.6.2 Effectiveness Evaluation: extracted KBs

Table 4.4 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

Systems	WebQ		
	Precision	Recall	F ₁
NESTIQUE	38.3%	26.0%	31.0%
OQA	28.4%	16.7%	21.0%
TAQA (No relaxation)	32.3%	32.3%	32.3%
TAQA	35.6%	35.6%	35.6%

Table 4.5: Performance of NESTIQUE and other systems on simple questions.

certain structural constraints (such as *head* and *rel* in a query and tuple should match) and does not take into account evidence embedded in 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 recall (39.3% with relaxed queries), it hurts precision. NESTIQUE does not enforce such structural constraints, enabling it to derive correct answers from tuples that are lexically and structurally different from the query.

Table 4.5 shows the performance of the systems on the WebQ query set. 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.

4.6.3 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

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 4.6: Contributions of various components of NESTIQUE : F_1 score

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 extracted 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 [15] 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 [10] 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 extracted KBs.

4.6.4 Ablation Study

Table 4.6 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 also affected performance, especially for CompQ-T, which is not targeted towards any specific KB. Paraphrasing bridges the lexical gap between

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 4.7: Correct answers in top-k predictions. Numbers in parentheses are normalized to last row.

natural language queries and tuples.

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 4.7, 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 4.8 shows how different extracted 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.

Being automatically extracted, tuples in an extracted KB may not always be correct. To investigate the impact of data quality on the overall system performance, we examined the tuples returned as evidence along with the answers for 100 randomly

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 4.8: Contributions of extracted KBs: F_1 score

selected questions from the test set. We found 12% of the tuples were incorrect and contribute to the errors made by the system. Although NESTIQUE attempts to mitigate errors in the data itself by considering metadata information of the tuples (such as extractor confidence, frequency) when ranking the answers to a question, these errors can further be reduced if the data quality were improved (e.g. via a human-in-the-loop approach).

4.7 Case Study

Table 4.9 shows examples from test data where NESTIQUE successfully (examples 1 and 2) or incorrectly (examples 3 and 4) derived an answer. 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 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

1	Q	what did Thomas Jefferson do before he was President?
	A	{Secretary of State}
	G(Q)	$\langle \text{Thomas Jefferson, do, ?x} \rangle \wedge \langle \text{do, before, he was president} \rangle$
	C(Q)	$A_1: \langle \text{President Thomas Jefferson, served, } \phi \rangle$ $A_2: \langle A_1, \text{ as, Secretary of State } \rangle,$ $A_3: \langle A_1, \text{ under, President Washington } \rangle$
2	Q	who did president Obama run against in 2008?
	A	{John McCain, McCain}
	G(Q)	$\langle \text{President Obama, run against, ?x} \rangle \wedge \langle \text{run against, in, 2008} \rangle$
	C(Q)	$A_2: \langle \text{Obama, run for president, against McCain, in 2008} \rangle$
3	Q	what led to the split of the republican party in 1912?
	A	{William Taft}
	G(Q)	$\langle \text{?x, led to split, republican party} \rangle \wedge \langle \text{led to split, in, 1912} \rangle$
	C(Q)	$A_3: \langle \text{William Taft, lead, divided Republican Party, in 1912} \rangle$

Table 4.9: Examples of successful and failed cases in NESTIQUE

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.

4.8 Conclusion

We introduced NESTIQUE, an online schemaless querying method for extracted KBs that leverages heterogeneity in KBs to support complex queries with multiple constraints. We described algorithms that, given a structured DATALOG-like query, matches different components of the query and finds evidence embedded in structurally diverse expressions. It can, thus, find answers from tuples that may not exactly match the query specification. Instead of learning query transformation functions of-

fine, it handles these mismatches at query-time. Our experiments demonstrate that NESTIQUE significantly outperforms state-of-the-art querying methods over extracted KBs in answering complex queries.

CHAPTER V

Querying Curated Knowledge Bases with Query Composition

To design a KB-QA system that can combine both curated and extracted knowledge bases, we must first understand the techniques for querying curated KBs and the complexity of questions they can support. In this chapter, we focus on the design of a KB-QA system that exclusively uses a curated KB as the background knowledge source. We propose a novel *decompose-execute-join* to construct complex query patterns using a set of simple queries. It uses a semantic matching model which is able to learn simple queries using implicit supervision from question-answer pairs, thus eliminating the need for complex query patterns.

5.1 Introduction

An important direction in KB-QA over curated KBs is based on semantic parsing, where a question is mapped to a structured query over the KB. Consider the simple question 1 in Fig. 5.1 and its corresponding query,

```
select ?x where {  
  Martin_Luther_King person.education ?c .  
  ?c education_institution ?x .
```

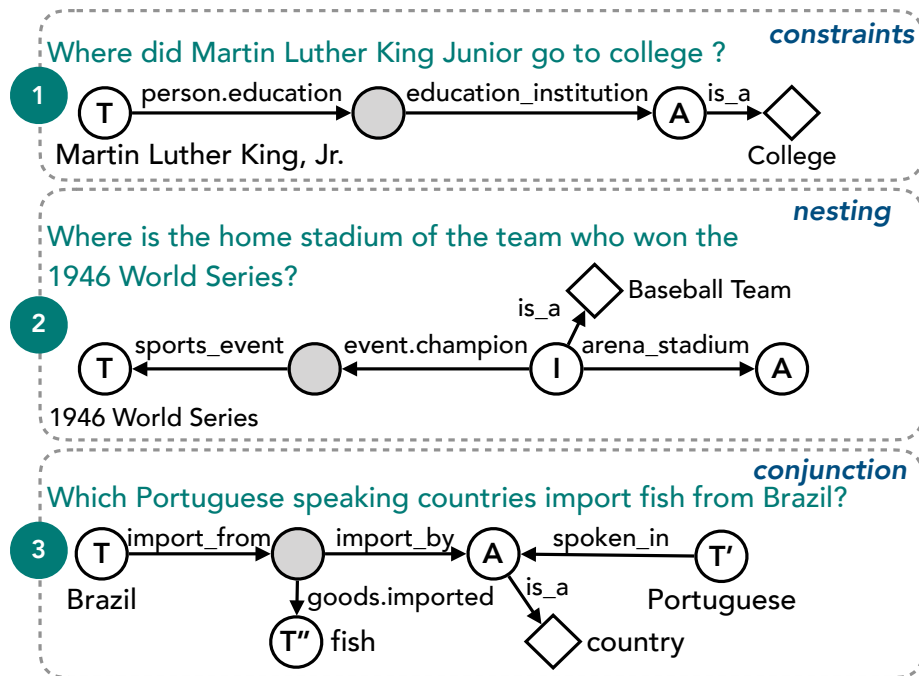


Figure 5.1: Example queries with constrained main relation (1) and multiple main relations (2 and 3). A main relation connects a topic T to an intermediate answer I or query answer A . The relation could be a n-ary relation, indicated in grey.

```
?x is_a College . }
```

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

A key challenge in KB-QA is learning how to bridge the gap between natural language expressions in the questions and the complex schema of the KB using only question-answer pairs. As a result, KB-QA systems have focused on simple questions which can be answered by querying a single relation (or path) in the KB. Little effort has been made to support compositional questions where queries involve joining multiple relations. We call such questions *complex questions* throughout this chapter. Consider example questions 2 and 3 in Fig. 5.1. Their corresponding queries have multiple query components: multiple topic entities (`Brazil` and `Portuguese`) and more

than one main relation (`import_from-import_by` and `spoken_in`). Generating such complex queries is much harder due to the structural complexity of the query patterns and the many expressions in the questions mapping to query components.

Traditionally, KB-QA systems handled compositionality by using query templates [87, 113, 1, 31]. While templates can encode complex query patterns, they inevitably have limited coverage. Modern KB-QA systems [10, 50, 106, 16] offer better coverage by modeling semantic parsing as a query generation task, where the goal is to construct a query pattern for a question using likely candidates for its query components. Candidates for query patterns are collected using bottom-up parsing or staged generation methods. The query candidates are ranked based on their semantic similarity to the question, and the best candidate is executed over the KB. Although more robust, these KB-QA systems face two major challenges: a) searching for good candidate query patterns, and b) learning the semantic matching function.

When a query becomes complex with many query components and joins, the number of possible candidates grows exponentially, making the search for good query candidates significantly harder. Consider the following example where the topic `1946_World_Series` is 3 hops from the answer. Given the topic entity, collecting and scoring all 3-hop paths to find the most likely path is too expensive if not infeasible.

Example 9. Where is the home stadium of the team who won the 1946 World Series?

Partial Queries:

q_1 : ?a event.champion ?c . ?c sports_event 1946_World_Series .

q_2 : ?b arena_stadium ?x .

Join: q_1 join_{?a=?b} q_2

Execute: ans = Busch Stadium

We hypothesize that complex query patterns can be constructed by joining a set of simple queries. Since simple queries have fewer query components, they also have fewer candidates. In the example above, q_1 and q_2 have fewer components than the

original query. Scoring the candidates for a simple query and executing the best query (q_1) can yield intermediate answers (**Cardinals** for $?a$), which can restrict the search space for subsequent simple queries ($?b$ in q_2 binds to answers of q_1). The process of decomposing the complex query into simple queries makes the search for complex query patterns tractable.

A natural question then is how to decompose the query generation process and learn the semantic matching function for simple queries. A simple approach is to annotate the linguistic parse tree of a question with query components and learn to map the tree elements to a query pattern [58, 112]. The mapping can be learned from example questions annotated with complex query patterns. Obtaining complex query patterns, however, can be cumbersome and error-prone. Recent works [16, 70] have shown that the mapping can instead be learned using distant supervision from question-answer pairs. While this works for simple questions, it is challenging for complex questions where only answers (e.g., **Busch Stadium**) to complex questions are available and not the simple queries (e.g., **St. Louis Cardinals** for q_1) that constitute the complex query. We propose that by restricting each simple query to a single relation path and by leveraging some prior domain knowledge, a semantic parser can be trained to answer simple queries using only implicit supervision for the simple queries.

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

quality of the weak, implicit supervision signals. To summarize this chapter makes the following contributions:

- We present a new KB-QA system, `TEXTRAY`, that automatically translates a given complex, compositional question to the matching query over a knowledge base (Section 5.3) .
- We propose a novel *decompose-execute-join* approach to construct a complex query pattern from partial queries. This enables efficient search for good candidates for a complex query (Section 5.4).
- We present a neural-network based semantic matching model that learns to score candidates for partial queries using implicit supervision from question-answer pairs. It uses an effective scoring strategy for candidates that combines the quality of full-query derivations of a candidate with domain knowledge (Section 5.5).
- We provide an extensive evaluation of our system on multiple QA datasets, where it significantly outperforms previous approaches on complex questions (Section 5.6).

We introduce key concepts in Sec.5.2. We outline problems studied in this chapter and our approach in Sec.5.3 before providing details in Sec.5.4-5.5. We present experiments in Sec.5.6.

5.2 Background

Our goal is to design a KB-QA system that can map a complex question Q in natural language to a matching query G , which can be executed against a knowledge base \mathcal{K} to retrieve answers to Q .

Knowledge Base. A knowledge base \mathcal{K} is a collection of triples of the form of (s, r, o) , where s , r and o denote subject, relation and object respectively. A triple can also be interpreted as a directed edge from s to o labeled with relation r , and \mathcal{K} as a directed graph. Higher-order relations are expressed using special mediator nodes.

Complex Question. A complex question Q in natural language corresponds to a query G over the \mathcal{K} such that G involves joining multiple main relations in the \mathcal{K} . G has a single query focus $?x$ that corresponds to the answer to Q . G can be interpreted as a sequence of simple partial queries $G = (G_1, G_2, \dots, G_o)$ connected via different join conditions. Each partial query corresponds to a main relation in G . Partial queries may share the query focus $?x$ (e.g. example 3 in Fig. 5.2) or a variable (e.g. example 2 in Fig. 5.2).

Computation Plan. A computation plan C is a tree that decides how a complex query G is constructed and executed from the partial queries. It uses two main functions: *simQA* and *join*. *simQA* denotes search and execution of likely partial query candidates. *join* denotes the join condition for two partial queries. Fig. 5.2 shows computation plans for running examples 2 and 3.

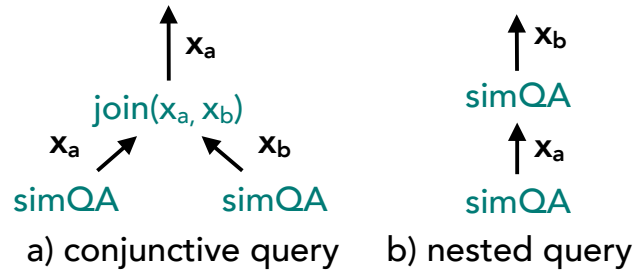


Figure 5.2: Example computation plan indicating how to construct the complex query given the partial queries

5.3 Solution Overview

Fig. 5.3 shows the design of a system that answers complex questions with simple queries. There are several key steps in the process.

Given a question Q , the system generates a computation plan that describes how its matching query G can be broken down into partial queries G_i . Next, based on the computation plan it finds candidates for the partial queries $\{G_i^{(k)}\}_{k=1}^L$. For instance, candidates for G_1 and G_2 can be collected simultaneously for a conjunctive question

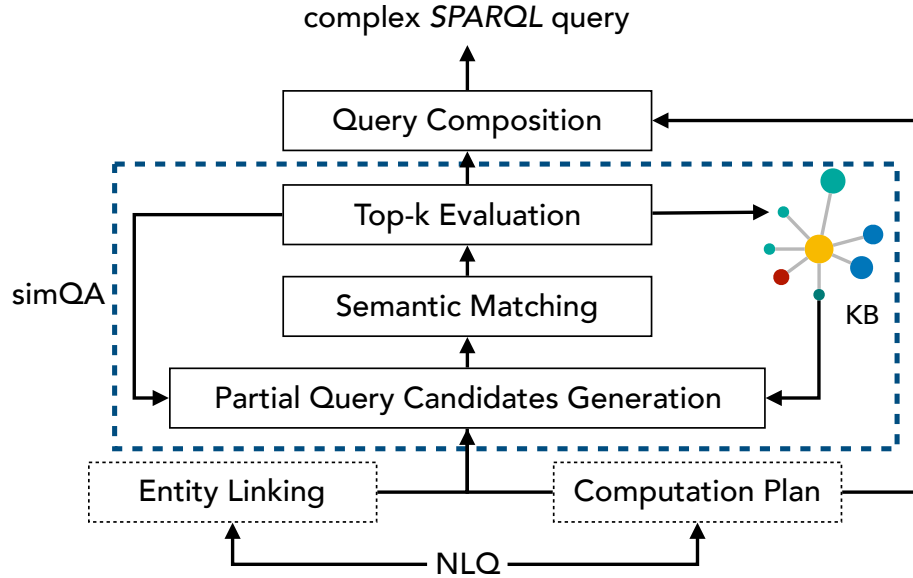


Figure 5.3: System Architecture of TEXTRAY

with a computation plan a) in Fig. 5.2. On the other hand, the search for G_2 can benefit from the answers of G_1 for the computation plan b). Given the candidates for partial queries, it computes their semantic similarity $S_{sem}(Q, G_i^{(k)})$ to the question. The best M candidates for each partial query are executed, their answers help find the candidates for the subsequent query and so on. In this manner, multiple full-query derivations are generated from simple query candidates. The derivation with the highest overall score is finally executed to find answers to the complex question.

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

Embodying these ideas, we design a system TEXTRAY that adopts a *decompose-*

execute-join approach to answer complex questions. Our query composition approach and our semantic matching model trained with weak, implicit supervision are the major contributions of this chapter. Formally, these tasks can be defined as follows:

Query Composition. Given a complex question Q and its computation plan C , find a sequence of partial queries (G_1, G_2, \dots, G_o) and construct the complex query pattern G such that executing G over \mathcal{K} provides answers to Q . To find a correct partial query G_i , we have to collect candidates $\{G_i^{(k)}\}_{k=1}^L$, score them and reserve the best candidates for finding G_{i+1} . Given K -best full-query derivations $G^{(k)} = (G_1^{(k)}, G_2^{(k)}, \dots, G_o^{(k)})$, we have to find the derivation G^* that best captures the meaning of Q . (Section 5.4)

Semantic Matching. Given a question Q and a partial query candidate $G_i^{(k)}$, a semantic matching model provides a semantic similarity score $S_{sem}(Q, G_i^{(k)})$. We have to learn the model offline using distant supervision from a collection of question answer-set pairs $\{Q^i, A^i\}_{i=1}^N$, where Q_i is a complex question and A_i is the set of entities from \mathcal{K} that are answers to the complex question. (Section 5.5)

5.4 Query Composition

We assume that a complex question can be answered using a sequence of simple, partial queries, each focusing on one main relation in the knowledge base \mathcal{K} . Predicting one main relation at a time offers several benefits. First, it ensures query composition is tractable. Consider a computation plan that describes how to construct a complex query pattern by joining one component (i.e. an entity node or a relation edge) at a time. Obtaining the plan reliably will be tedious and error-prone. In contrast, it is easier to obtain a computation plan that describes how to join partial queries. Also, independently matching each component will lose useful information for collectively resolving components in the partial queries. Lastly, executing more specific partial query patterns will be more efficient than query patterns for individual

components. For instance, a query pattern q : Portuguese spoken_in ?o will find fewer matches to join with than q : Portuguese ?r ?o .

Identifying entities. In order to find candidates for the partial queries, we begin by identifying entities from \mathcal{K} that are mentioned in the question. In our example question 3 from Fig. 5.1, mention ‘Portuguese’ refers to the language Portuguese or dialect Brazilian_Portuguese in \mathcal{K} , and ‘Brazil’ refers to the South American country in \mathcal{K} . We delay disambiguation to later steps in order to tolerate errors in entity linking. We use an entity linking system [103, 13] that returns a set of possibly overlapping pairs $E = \{(mention, entity)\}$ with attached confidence scores. We consider 10 best pairs based on their confidence scores.

Constructing Computation Plan. Even though a computation plan can include aggregations and value comparisons, we assume it includes two operations, *simQA* and *join* for simplicity. The *simQA* operator denotes search for a partial query, and *join* describes the join condition of two partial queries. Post-order traversal of the plan yields a sequence in which the partial queries are executed: $z = z_1, z_2, \dots, z_n$, where $z_i \in \{simQA, join\}$. Note that we do not chunk the original question into sub-questions but simply encode the number of partial queries required to construct the query G . We obtain the computation plan using augmented pointer networks [85] trained to predict its likelihood as $P(C|Q) = \prod_{i=1}^j P(z_i|Q, z_{1:i-1})$. In practice, however, the number of main relations is small (2-3). A computation plan can also be approximated using simple syntactic cues such as the number of verb phrases in the question. The dependencies can be estimated from the type of clause connectors in the question i.e. coordinating (example 1 in Fig. 5.2) vs subordinating clause (example 2 in Fig. 5.2).

5.4.1 Partial Query Candidate Generation

We next have to construct the complex query pattern using the sequence described in the computation plan. We generate candidates for a partial query by staged generation method, measure their semantic similarities to the question and find the best candidate (*simQA* operation). Compared to previous methods [106, 10], we tailor the staged generation strategy to handle compositionality. The candidate generation process is described by a set of states S and actions A . We introduce a new state S_t and action A_t (Fig. 5.4).

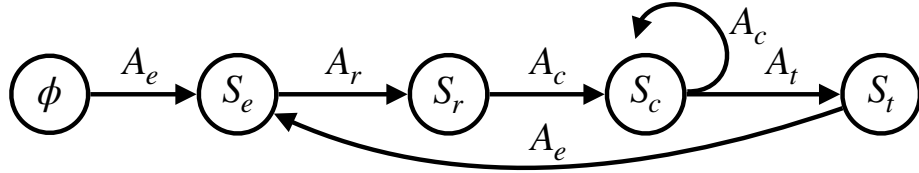


Figure 5.4: Action space showing how to generate candidates for partial query.

States $S = \{\phi, S_e, S_r, S_c, S_t\}$ indicate an empty query (ϕ), single entity (S_e), a main relation path (S_r), a constraint (S_c) in a partial query candidate. Action $A = \{A_e, A_r, A_c, A_t\}$ grow a candidate by adding one query component at a time. As shown in Fig. 5.5, action A_e finds candidates for the seed entity, A_r adds main relation paths to the seed entity candidates and action A_c adds any constraints. Action A_t denotes termination of the partial query G_i and the transition to a state S_t . In the state S_t , the candidate generation refers to the computation plan and determines how to proceed to find candidates for subsequent partial query G_{i+1} . Concretely, if the next operation is *simQA*, G_{i+1} depends on the answers of G_i . Candidates for G_i , therefore, must be scored and the best candidate be executed. The answers of G_i would become the seed for the collecting candidates of G_{i+1} . If the next operation is *join*, such as in our running example, candidate generation for G_{i+1} can resume independently (see Fig. 5.6).

Identify seed entity (A_e). An entity is a seed for a candidate. It could be an

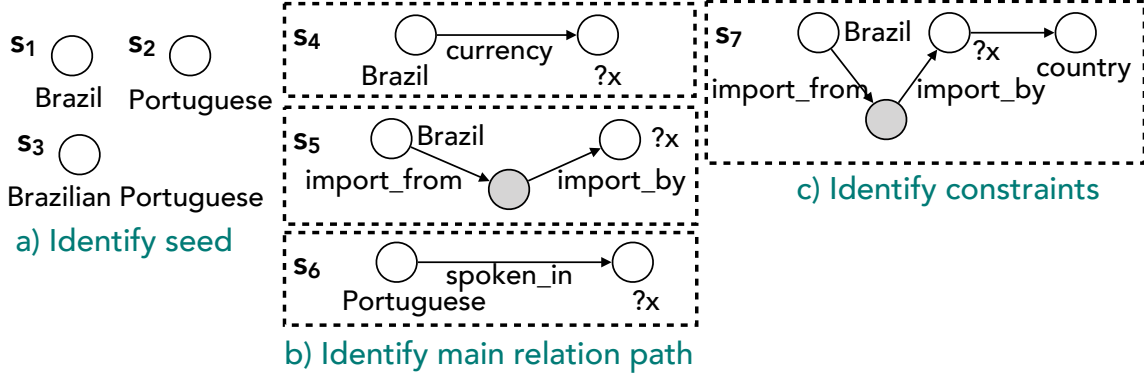


Figure 5.5: Staged candidate generation for a partial query in TEXTRAY.

entity from $E = \{(mention, entity)\}$ pairs identified by the entity linking system or an answer of a previous partial query. For instance, Brazil, Portuguese or Brazilian Portuguese are seed entities. We only allow entities that are not consumed by a previous partial query candidate. For instance, Brazil cannot be the seed for the subsequent partial query in Fig. 5.6.

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

Identify constraints (A_c). We try to attach any constraint entities to the main relation and variables. We consider a set of constraints $E_c = \bigcup\{E, E_t, E_o\}$, where E is the set entity links, E_t are entities connected to the answer via specific relations¹, and E_o are named entities of type *date*, *time*. We ignore entities in E_t that do not appear in the question. We next collect 1-hop paths connecting constraint entities to the query. For instance, COUNTRY is a type constraint on the answer in Fig. 5.5. We consider all subsets of the constraints to accommodate multiple constraints in the query.

Terminate partial query (A_t). After collecting candidates of a partial query

¹common.topic.notable_types, common.topic.notable_for

G_i , the search refers to the computation plan to determine the dependencies for the subsequent partial query G_{i+1} . When the next operation is a join operation, it indicates G_{i+1} does not share any entities from the question. In that case, the search does not have to wait to generate candidates for G_{i+1} . A non-overlapping entity in E becomes the seed for G_{i+1} (as shown in Fig. 5.6). When the next operation in the computation plan is *simQA*, it indicates the G_{i+1} relies on the answers to G_i i.e. the seed entities for G_{i+1} are the answer entities of G_i . In such a case, we score the partial query candidates $G_i^{(k)}$ based on their semantic similarity to question Q , and find the K -best query candidates. We translate these candidates to SPARQL queries and execute them against \mathcal{K} .

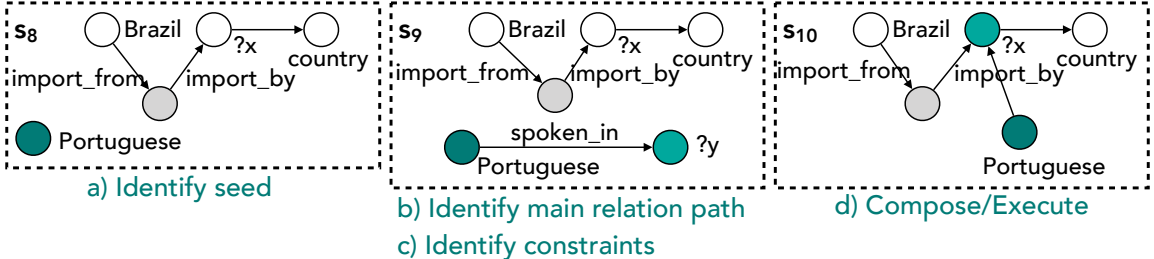


Figure 5.6: Staged generation for a subsequent partial query in TEXTRAY

5.4.2 Query Composition and Execution

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

Note that we only compute semantic similarity of partial queries and not derivations. We still need to judge which derivation among the k best derivations is the correct query G^* for the input question. One approach is to simply aggregate the

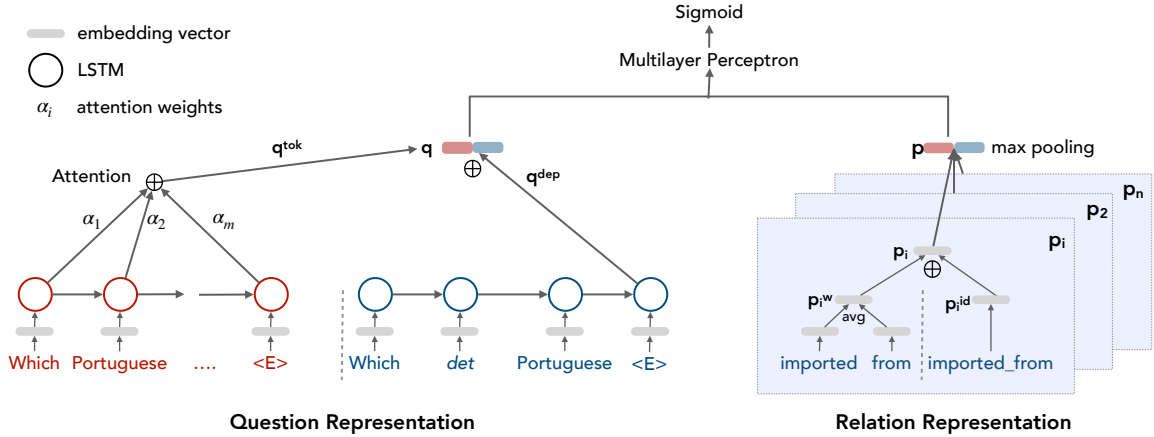


Figure 5.7: Semantic Matching Model for comparing question representation to partial query representation.

semantic similarities of the partial queries. While such a strategy might work for aggressively pruning the large search space of derivations, we need a more reliable scoring function. We train a log-linear model using a set of (question-answer) pairs for scoring derivations based on the scores of the partial queries and other features. Some features we use are confidence scores of seed entities in the queries, number of constraint entities, number of variables, number of partial queries, number of relation edges, and number of answer entities. Once the features are extracted, the model next has to be trained for ranking and scoring the derivations. Due to the lack of correct queries, one effective strategy to learn the model is to rely on the F_1 score of the predicted answer set of a query to determine its correctness. Given the F_1 scores of the queries, we use a pointwise ranking method to determine the best derivation. We translate the best derivation to a SPARQL query, execute the query and return the set of answers entities as answers to the question.

5.5 Semantic Matching Model

5.5.1 Model Architecture

We use a neural network based model for semantic matching. The architecture of the proposed model is shown in Fig. 5.7. It encodes both the question and query as semantic vectors to find their semantic similarity. We use two Long Short-Term Memory (LSTM) networks and an attention mechanism to learn the latent representation of the question Q . We only consider the main relation path and constraint relations to encode the partial query G_i . This is because the entities in the query are already grounded by the entity linking system, and we only need to infer different relations in the query. We feed the latent vector q for the question and p for the partial query as input to a multi-layer perceptron (MLP). The MLP outputs a scalar that is used as the semantic matching score $S_{sem}(Q, G_i)$. We first describe how we encode a question and a partial query. We then describe in detail other elements of the architecture before discussing the learning objective of the model.

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

Encoding main relation path and constraint relations. We encode the partial query using the various relations in the query. For each relation P_i in the partial query, we consider the sequence of both relation ids and relation names [63]. For example, the id sequence of the main relation in Fig. 5.7 is `{import.by}`, while the word sequence is `{‘import’, ‘by’}`. We convert the word sequence to a sequence of embedding vectors and represent P_i as the average embedding p^w of the embedding vectors. At the id level, we translate the relation directly using another embedding matrix E_p of relations. The semantic vector of P_i then is $p = [p^w; p^{id}]$. We apply max pooling over the hidden vectors of the relations and obtain a compositional semantic representation of the partial query.

Attention mechanism. Our goal is to estimate the semantic similarity of a question vector q and a partial query vector p . Complex questions, however, are long and have expressions for matching multiple partial queries in the KB. Irrelevant information in the question can thus distract the matching. We use an attention mechanism to improve the quality of question representation. The idea behind attention is to learn to emphasize parts of the question that are informative to the given attention vector. Following [64, 46], we use the partial query vector p as the attention vector to highlight relevant parts of the question word vector q^w . Specifically, given all hidden vectors h_t at time step $t \in \{1, 2, \dots, n\}$, the context vector c is the weighted sum of all the hidden states:

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

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

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

where W, W_p, W_q are learnable network parameters. The attention weights can be

interpreted as the extent to which the model focuses on a specific word in the question given a partial query.

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

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

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

5.5.2 Implicit Supervision

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

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

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

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

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

Example 10. Consider two candidate partial queries:

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

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

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

$D_2^{(1)} = ?a \text{ event.champion } ?c . \quad ?c \text{ sports_event } 1946_World_Series .$

$?a \text{ arena_stadium } ?x .$

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

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

$?a \text{ place_of_birth } ?x .$

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

The F_1 scores and values then are:

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

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

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

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

Addressing Spurious Matches. Implicit supervision strategy described above is susceptible to spurious partial queries which happen to find correct answers but do not capture the semantic meaning of the question. Identifying such spurious partial candidates as positive examples can greatly affect the training data quality and the performance of the semantic matching model. We propose to alleviate this problem by incorporating some domain knowledge as priors for scoring. By qualitatively examining the positive examples, we found that correct candidates tend to use the same words as the natural language question. For instance, a question “*Which governmental jurisdiction held the LIX Legislature of the Mexican congress in 2011 ?*” has a partial query with main relation `governmental_jurisdiction.governing_officials` sharing

words **governmental** and **jurisdiction**. Thus, surface-form similarity of the main relation and question can be used to promote true positive and false negative examples. We define $lexical_score(Q, G_i^{(k)})$ that computes the ratio of number of words in the main relation of $G_i^{(k)}$ that are mentioned in the question Q .

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

5.6 Experiments

We present extensive experiments which demonstrate TEXTRAY significantly outperforms existing KB-QA systems on answering complex questions. We found our approach to construct the complex query from partial queries is superior to traditional approaches that map a question directly to a complex query.

5.6.1 Datasets and Baseline Systems

We use two KB-QA benchmark datasets.

- **ComplexWebQuestions (CompQWeb)**[85]: This is a very recent benchmark designed specifically for highly compositional questions. The dataset contains 34,689 examples, each containing a complex question, answers (including alias) and a SPARQL query against the Freebase. We only use the question-answers in the dataset for training and evaluation. We use the SPARQL queries only for qual-

itative analysis of our system. The dataset is divided into 27,734 train, 3,480 dev and 3,475 test cases.

- **WebQuestionSP (WebQSP)**[107]: This dataset is an enhanced version of the de facto benchmark dataset WebQuestions (WebQ) [16]. It consists of 4,737 questions and their answers, split into 3,098 train and 1,639 test cases. Additionally, this dataset provides correct SPARQL query for each question, which we reserved for qualitative analysis. Unlike the CompQWeb dataset, most of the questions in WebQSP are simple. Only 4% questions in the test set have multiple entity constraints [108]. We evaluate on this dataset to demonstrate our proposed approach is effective across questions of varying complexity. Note that we chose WebQSP over WebQ because the SPARQL queries provide ground truth for qualitative analysis of the queries generated by various systems.

Knowledge Bases. We use the entire Freebase² dump on 2015-08-02 as the background knowledge source for comparison to baseline systems. We host it with Virtuoso engine³. It contains about 46 million entities and 2.9 billion facts. Our approach, however, can be easily adopted to any other structured knowledge base.

Evaluation Metric. We use averaged F_1 score of the predicted answers to measure the effectiveness of a KB-QA system. We also compute precision@1 as the fraction of questions that were answered with the exact gold answer [13].

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

- **CompQA**[63]: It generates full query candidates for complex questions and uses a neural network model for semantic matching. They handle complex query structures by encoding the constraints explicitly in their semantic matching model. We used the code provided by the authors for experiments.
- **Parasempre**[16]: It parses questions to logical forms that are executed against

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

³<https://virtuoso.openlinksw.com>

Freebase. It generates query candidates by recursively generating logical forms. We used the publicly-available implementation of their system and their pre-trained model.

Many other KB-QA systems [109, 31, 1, 10] powered by Freebase are designed to handle simple questions with a few additional constraints. We directly report numbers from their papers. The only systems that can handle highly compositional questions [80, 85] do not use Freebase as their knowledge source.

5.6.2 Experimental Setup

Entity Linking. We used the entity links released by [103] for the WebQSP dataset. Since CompQWeb is a new dataset with no entity links, we ran AQQU [13] to annotate questions with entity links.

Pre-trained embeddings. We initialize the embeddings for word-level representations for questions and relations with Glove vectors of dimension 300 [71]. There were 7,371 (906) words in the questions vocabulary, 2,853 (1,982) words in the relations vocabulary (excluding the stop words) and 6,904 (3,818) unique relations in the ComplexWeb (WebQSP) dataset. We randomly initialize the embeddings for relation ids. The embeddings are not fixed during training and are learned along with other parameters.

Training Semantic Matching Model. We used NVIDIA GeForce GTX 1080 Ti GPU for training the semantic matching model. To encode the question word sequence and question dependency structure, we use two bi-directional LSTMs. We set the size of the LSTM hidden layer to 300. For the multi-layer perceptron, we use 1024 as the size of hidden layer and *sigmoid* as the activation function for hidden layer. Based on our training data generation for (question, partial query) pairs, we could find 104k (5,026) positive examples and 2.5M (370k) negative examples for CompQWeb (WebQSP) dataset when using a threshold of 0.5 for the partial query

candidate quality score ($V(G^{(k)})$). We upsample the positive examples such that there are 2 negative examples for each positive example. We found similar performances for training data with 4:1 to 1:1 negative examples to positive examples ratio. We chose 2:1 balance ratio in our experiments. We trained the model with a batch size of 64 and used *Adam*[54] optimizer for adaptive learning rate optimization.

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

5.6.3 Results and Discussion

Effectiveness on Complex Questions. As reported in Table 5.1, our proposed system `TEXTRAY` achieves the best performance (with a large margin of 26.82% absolute gain in F_1) among these methods, confirming the potential and effectiveness of `TEXTRAY`. While `CompQA` uses a similar approach of enumerating query candidates and scoring them using a semantic matching model, it assumes queries have a fixed number (one) of main relations. As a result, it does not generate candidates for multiple main relations to construct complex queries. In contrast, leveraging the computation plan helps `TEXTRAY` find candidates for multiple partial queries that can construct better complex query candidates.

We also achieve significantly higher F_1 than `Parasempre` that generates query candidates by recursively generating logical forms. By relying on a mapping of natural language expression to relations and a small set of composition rules, it can generate highly compositional logical forms. However, the logical forms use relations from the large vocabulary in the KB instead of the neighborhood of the seed [104]. Also, `Parasempre` relies heavily on keeping a large beam of good derivations that lead

Method	Average F_1	Precision@1
TEXTRAY	33.87	40.83
CompQA [63]	4.83	4.83
Parasempre [16]	7.05	12.37
SplitQA (web) [85]	-	27.50
MHQA (web) [80]	-	30.10

Table 5.1: Average F_1 scores and Precision@1 on CompQWeb.

to the correct answer. This becomes the bottleneck when understanding complex questions. In contrast, our staged generation design ensures that only areas of the KB that could potentially lead to a successful query are explored. We include results of SplitQA [85] and MHQA [80] that handle complex questions by decomposing them into simple questions, but rely on noisy textual data sources. Evidently, we could answer complex questions more reliably and effectively from structured knowledge sources which naturally support compositionality.

Effectiveness on Simple Questions. We provide a complimentary evaluation on simple questions to demonstrate TEXTRAY can adapt to different complexities of questions. For these experiments, we assume the computation plan simply has a single *simQA* operation i.e. the question has one main relation. We found TEXTRAY achieved comparable if not higher F_1 scores on the WebQSP dataset as other KB-QA systems that adopt a candidate enumeration-scoring strategy (see Table 5.2). STAGG [106], a popular KB-QA system uses a similar approach but improves the results using feature engineering and by augmenting entity linking with external knowledge.

Our proposed system can be integrated with better entity linking and more sophisticated model architectures [109] for semantic matching. The semantic matching model, like in TEXTRAY, however, should be able to accommodate for long, complex question sequences (e.g. via attention). Lastly, we found TEXTRAY adapts well to

Method	Average F_1	Precision@1
TEXTRAY	60.3	72.16
CompQA [63]	59.5	61.64
Parasempre [16]	46.9	51.5
STAGG [106]	66.8	67.3
MulCQA [10]	52.4*	-
AQQU [13]	49.4*	-

Table 5.2: Average F_1 scores and Precision@1 on WebQSP dataset. * are results on the WebQ dataset.

Method	CompQWeb F_1^{**}	WebQSP F_1^*
TEXTRAY	50.83	84.57
CompQA [63]	31.30	75.03
Parasempre [16]	19.15	60.95

Table 5.3: Upper bound F_1 scores for candidate generation.

varying complexities of questions compared to other systems tailored to a specific class of questions [10].

Effectiveness of Candidate Generation. It is possible to pre-train a system’s semantic matching model on different datasets, which could yield different overall F_1 scores when evaluating a target dataset. Thus, we measure the upper bound F_1^* scores of the candidate queries each system generates. Comparing these upper bounds provides insights into the expressivity of their query composition process, agnostic to the quality of the semantic matching model. We found TEXTRAY could consistently find better candidate queries than other systems, as reflected in the best F_1^* scores of their candidate queries (see Table 5.3).

Top-k results. Table 5.4 shows the top-k results on the two datasets. For a large majority of the questions, the query with the highest F_1 score was among the top-10 predictions. Despite using an end-to-end semantic matching model, TEXTRAY

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

Table 5.4: Percentage of questions with the highest F_1 score in the top-k candidate derivations, and the average best F_1 .

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

Table 5.5: Component-wise ablation results (Average F_1) of TEXTRAY.

generates SPARQL queries that can be executed over the KB. It can be treated as a natural language end-point to the KB. The top-k queries can be provided as alternative interpretations to the user, who can then select a query to execute. This process would be less tedious and error-prone for a user than writing a complex SPARQL query.

5.6.4 Ablation Study

We also investigate the contributions of various components in complex question answering. Table 5.5 summarizes these results.

Encoding Constraints. Complex questions tend to have additional constraints on the main path (such as ordinal, answer type, time, etc). Ignoring these constraints can hurt the F_1 score of a query. In the CompQWeb dataset, we observed that 35% of the queries had at least one constraint. Most of the queries (85%) in WebQSP

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

Table 5.6: Average F_1 for different ranking variants in TEXTRAY.

were simple with no constraints. To demonstrate the effectiveness of including additional constraints in query candidates and in semantic matching, we construct simple baseline: candidate generation does not add additional constraints on the main path sequence, and semantic matching model does not encode additional constraints. As can be seen, overall F_1 is higher when constraints are included. The performance gains were smaller on the WebQSP dataset that comprises mostly of simple questions.

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

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

Ranking. As shown in Table 5.4, the upper bound for average F_1 score for top-10 query candidates is much higher than the F_1 score for the best query predicted by

the model. This indicates that a good re-ranking method can significantly improve the overall performance of the system on the KB-QA task. Our ranking model uses features including topic entity linking scores, the query structure and the semantic similarity score for partial queries. We conducted experiments with two different pointwise-ranking methods: linear regression and logistic regression. We compare with a simple baseline where score of a complex query derivation is simply the sum of scores of the partial queries. For training the regression based model, we used F_1 scores as labels for the queries. For training the classifier we used a threshold to classify an example as positive and negative. We found linear regression outperformed other ranking methods because it’s learning objective is to best predict the F_1 scores of the queries (see Table 5.6).

5.6.5 Qualitative Analysis

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

Query Analysis. We compared the complex queries generated by TEXTRAY for complex questions (CompQWeb) with the ground truth queries from the dataset to

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

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

5.6.6 Error Analysis

We analyze randomly sampled 50 queries which had F_1 scores less than 0.1. We broadly classified the errors. Table 5.7 shows some of the failed questions. About 36% of the errors were made because incorrect entities were identified by the entity linking system. 88% of these were made when finding the first partial query. Errors in entity linking become much severe as they propagate when composing a complex query from a sequence of partial queries.

Not surprisingly, a large fraction of the errors (45%) was made because a wrong or ambiguous relation was scored higher than the correct relation by the semantic matching model. It confirms that relation matching is indeed the most crucial component of a KB-QA system. Interestingly, in 50% of the cases with relation matching errors,

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

Table 5.7: Example failed queries from CompQWeb

TEXTRAY found a relation that had the same domain as the correct path (e.g. `education.education.student` vs. `people.person.education` – `education.education.institution`). The two relation paths were very close in meaning, and are hard to distinguish with a semantic matching model. Also, often noisy signals (such as in example 3 in Table 5.7) made it difficult to correctly predict one of the two main relations in the question. Lastly, we found that 19% of the errors were made when constraints or value nodes were missed in a subsequent partial query.

5.7 Conclusion

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

CHAPTER VI

Querying Curated and Extracted Knowledge Bases

Knowledge-based question answering (KB-QA) can benefit from a combination of high-coverage facts from an extracted KB and high-quality facts from a curated KB. In this chapter, we look at answering complex questions using a combination of different types of knowledge sources. We make two key modifications to our KB-QA system, `TEXTRAY`, which answered complex questions over a curated KB. First, we construct complex query patterns using a sequence of simple queries each targeted at a specific KB. Second, to enable collective reasoning over different forms of relations in the KBs, we propose a novel neural-network based model trained using relation alignment and implicit supervision. We call our new KB-QA system for combining information from multiple KBs, `MULTIQUE`.

6.1 Introduction

A single knowledge source, curated or extracted, is often sufficient to answer simple questions which have a single main relation. For instance, example question 1 in Fig. 6.1 can be answered with a query `(Rihanna, place_of_birth, ?)` over a curated KB or `(Rihanna, 'was born in', ?)` over an extracted KB.

Often when questions become complex with multiple relations and entities, no single KB would provide both high coverage and ontological knowledge representation

-
1. Where was Rihanna **born**? *simple*
2. What college did the **author of** 'The Hobbit' **attend**? *nesting*
3. Which Portuguese **speaking** countries **import** fish from Brazil? *conjunction*

Figure 6.1: Simple vs Complex questions.

to answer the questions. For instance, to answer complex question 2 in Fig. 6.1, we need to infer relations corresponding to expressions ‘author of’ and ‘attend’, both of which may not be present in a single KB. We aim to integrate inference over curated and extracted KBs to exploit the information across multiple knowledge sources to answer complex questions. Combining information from multiple sources offers two benefits: evidence scattered across multiple KBs can be aggregated, and evidence from different KBs can be used to complement each other. For instance, inference over ontological relation `book_author` can benefit from textual relation ‘is written by’. On the other hand, evidence matching ‘attend’ may exclusively be in the curated KB.

Example 11. What college did the author of ‘The Hobbit’ attend?

Simple Queries:

G_1 : The_Hobbit ‘is wrtten by’ ?a.

G_2 : ?b person.education ?c . ?c institution ?x.

Join: $G = G_1 \text{ join}_{?a=?b} G_2$

Evaluate: ans = University of Oxford

Leveraging multiple KBs is an attractive approach to answer complex questions but is seldom studied. Existing methods [43] assume a simple abstraction over the KBs and have limited ability to aggregate information across KBs. In this chapter, we extend our framework TEXTRAY, and design a hybrid KB-QA system MULTIQUE for combining information from a curated KB and an extracted KB. We follow the

enumerate-encode-compare approach in `TEXTRAY` and construct complex query patterns using simple queries. However, in order to leverage multiple knowledge sources, we allow each simple query to target a specific KB. This enables `MULTIQUE` construct complex query patterns where the constituent simple queries access potentially different knowledge sources.

The hybrid KB-QA setting poses new opportunities and technical challenges. First, the rich and ambiguous nature of multiple knowledge sources allows a fact to be expressed using different relation forms. The semantic matching model trained to infer over one type of relations (e.g. ontological) may not generalize to other relation type (e.g. textual). Second, redundancy in facts across multiple knowledge sources allows interleaved relation inference. For instance, aligning textual relations ‘is written by’ and ‘is author of’ to ontological relation `book_author` can improve the semantic vector representation of `book_author` for relation inference. Third, the knowledge source being queried must be taken into account when aggregating scores of simple query candidates to find the *best* complex query derivation. For instance, a simple query answerable with a curated KB must be preferred over a simple query targeted at an extracted KB.

In order to address the aforementioned challenges, we propose a neural network based approach that aligns different relation forms and learns unified semantic representations of textual and ontological relations. This enables the model infer over relations of different forms and provides flexibility to interleave inference over different relation forms. Due to the lack of availability of fully-annotated hybrid queries to train the model, we learn to find simple queries with implicit supervision signals in the form of labeled answers for complex questions. To predict the *best* complex query derivation, we train a log-linear model with features over semantic similarity of constituent partial queries, knowledge source (such as extracted fact confidence, entity linking scores of the extracted fact) and query structure (such as number of targeted

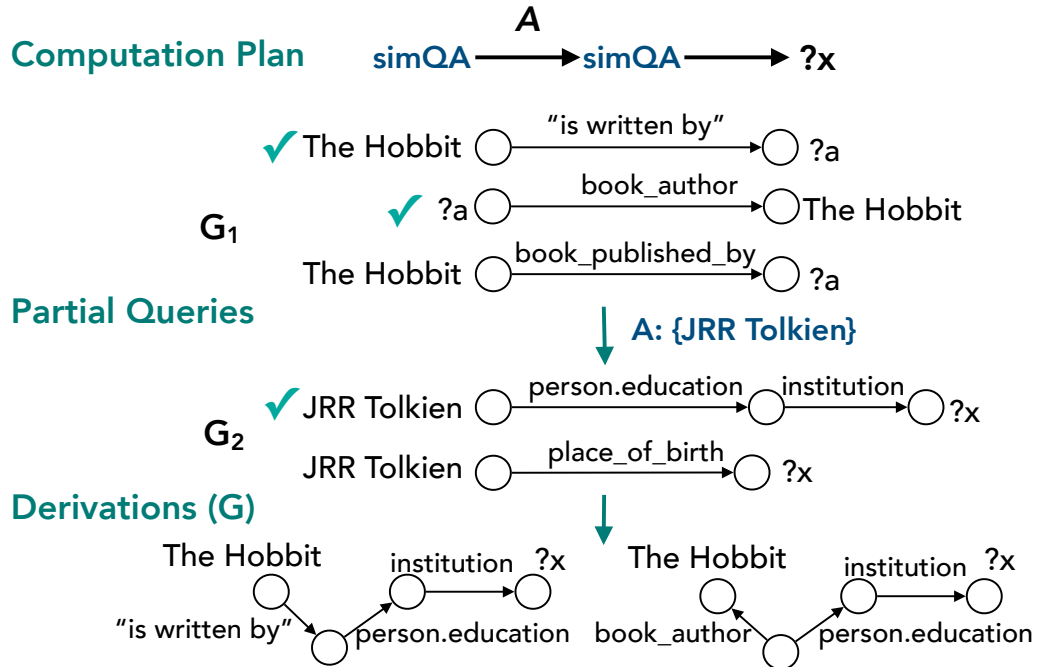


Figure 6.2: Partial queries and derivations.

knowledge sources). The contributions of this chapter are summarized below.

- We propose a novel KB-QA system, **MULTIQUE**, that combines information from curated and extracted knowledge bases to answer complex questions. To the best of our knowledge, this is the first attempt to answer complex questions from multiple knowledge sources.
- To leverage information from multiple KBs, we construct query patterns for complex questions using simple queries each targeting a specific KB. (Section 6.3 and 6.5).
- We propose a neural-network based model that aligns diverse relation forms from multiple KBs for collective inference. It learns unified semantic representations of textual and ontological relations, providing flexibility to interleave inference over different relation forms. The model learns to score simple queries using implicit supervision from answers to complex questions (Section 6.4).
- To determine the *best* complex query derivation among potentially multiple correct derivations, we train a log-linear model with features over semantic similarity,

knowledge source and query structure.

- We provide extensive evaluation on benchmarks demonstrating the effectiveness of proposed techniques on questions of varying complexity and KBs of different completeness (Section 6.6).

6.2 Task and Overview

Our goal is to map a complex question Q to a query G , which can be executed against a combination of curated KB \mathcal{K}_c and extracted KB \mathcal{K}_o .

Knowledge Bases. The background knowledge source $\mathcal{K}=\bigcup\{\mathcal{K}_c, \mathcal{K}_o\}$ is denoted as $\mathcal{K}=(\mathcal{V}, \mathcal{E}, \mathcal{R})$, where \mathcal{V} is the set of entities and \mathcal{E} is a set of triples (s, r, o) . A triple denotes a relation $r \in \mathcal{R}$ between subject $s \in \mathcal{V}$ and object $o \in \mathcal{V}$. The relation set \mathcal{R} is a collection of ontological relations \mathcal{R}^o from \mathcal{K}_c and textual relations \mathcal{R}^t from \mathcal{K}_o . A higher order relation is expressed using multiple triples connected using a special CVT node.

Complex Question, Q corresponds to a query G which has more than one relation and a single query focus $?x$. G is a sequence of partial queries $G = (G_1, G_2, \dots, G_o)$ connected via different join conditions. A partial query has four basic elements: a *seed* entity s^r is the root of the query, a *variable* node o^v corresponds to an answer to the query, a *main relation path* (s^r, p, o^v) is the path that links s^r to o^v by one or two edges from either R^o or R^t , and *constraints* take the form of an entity linked to the main relation by a relation c . By definition, each partial query targets a specific KB.

A computation plan C describes how the query G is constructed and evaluated given the partial queries. It includes two functions, *simQA* and *join*. *simQA* is the model for finding simple queries. It enumerates candidates for a simple query, encodes and compares them with the question representation, and evaluates the best candidate. *join* describes how to join two partial queries i.e. whether they share

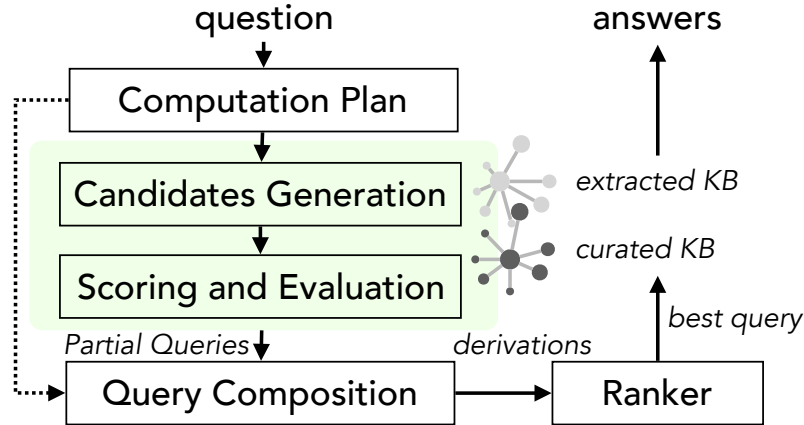


Figure 6.3: System Architecture of MULTIQUE

the query focus or another variable node. Fig. 6.2 shows the partial queries and computation plan for the running example 11.

Overview. Given a complex input question, the task is to first compute a computation plan that describes how to break down the inference into simple partial queries. We then have to gather candidates for each partial query from both curated and extracted KBs. For each candidate, we have to measure its semantic similarity to the question using a neural-network based model that should be capable of inference over different forms of relations. We then have to join the different partial queries to find the complex query for the question. Since there can be multiple ways to answer a complex question, we derive several full query derivations. We rank them based on the semantic similarity scores of their partial queries, query structure and entity linking scores. We execute the *best* derivation over the multiple KBs. Fig. 6.3 shows the architecture of our proposed system, MULTIQUE.

6.3 Partial Query Candidate Generation

We first describe how we find candidates for partial queries given an input question. We use a staged generation method with *staged* states and actions. Compared to previous methods [106, 63] which assume a question has one main relation, our

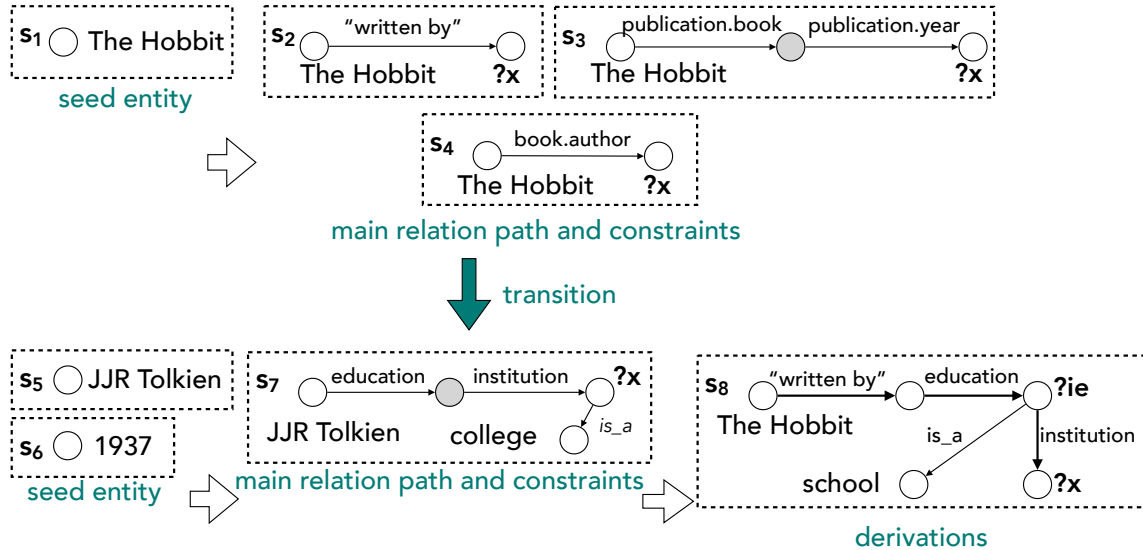


Figure 6.4: Example Candidate Generation for the running example 1.

strategy can handle complex questions which have multiple main relations (and hence partial queries). We include a new action A_t that denotes the end of the search for a partial query and transition to a state S_t . State S_t refers back to the computation plan to determine the join condition between the current partial query and the next query. If they share an answer node, candidate generation for the subsequent query can resume independently. Otherwise, it waits for the answers to the current query.

We generate (entity, mention) pairs for a question using entity linking [13] and then find elements for query candidates. Fig. 6.4 depicts our staged generation process.

Identify seed entity. The seed s^r for a partial query is a linked entity in the question or an answer of a previously evaluated partial query.

Identify main relation path. Given a seed entity, we consider all 1-hop and 2-hop paths p . These include both ontological and textual relations. The other end of the path is the variable node o^v .

Identify constraints. We next find entity and type constraints. We consider entities that can be connected using constraint relations `is_a` relations¹ to the variable node

¹`common.topic.notable_types,common.topic.notable_for`

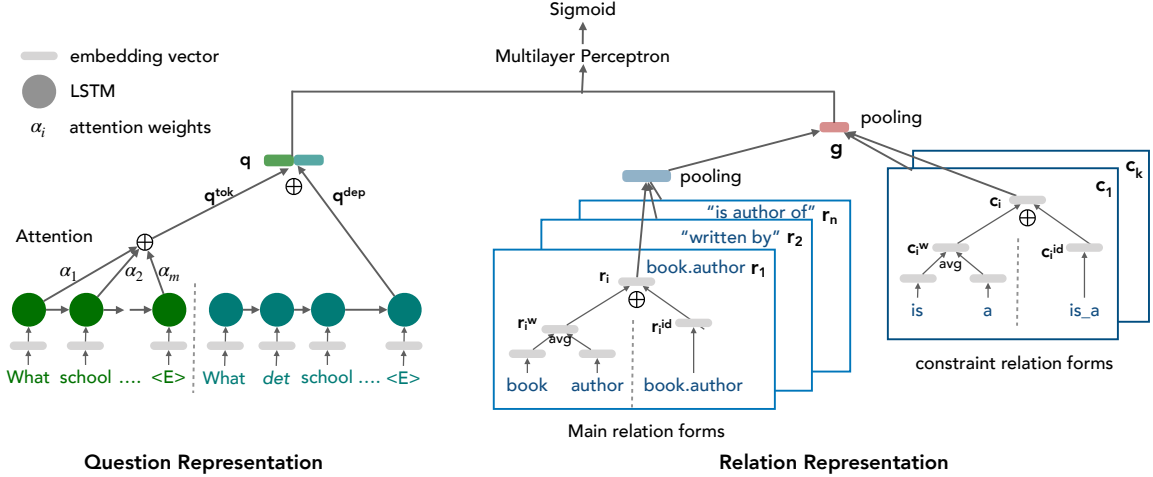


Figure 6.5: Semantic Matching Model for collective reasoning over diverse relation forms.

o^v . We also consider entities connected to the variables on the relation path via a single relation. We consider all subsets of constraints to enable queries with multiple constraints.

Transition to next partial query. Once candidates of a partial query G_i are collected, we refer to the computation plan to determine the start state of the next partial query G_{i+1} . If the next operation is *simQA*, we compute the semantic similarity of the candidates of G_i using our semantic matching model and evaluate K -best candidates. The answers form the seed for collecting candidates for G_{i+1} . Otherwise, candidate generation resumes with non-overlapping entity links in G_i .

6.4 Semantic Matching

We now describe our neural-network based model which infers over different relation forms and computes the semantic similarity of a partial query candidate to the question.

6.4.1 Model Architecture

Fig. 6.5 shows the architecture of our model. To encode the question, we replace all seed (constraint) entity mentions used in the query by dummy tokens w_E (w_C). To encode the partial query, we consider its query elements, namely the main relation path and constraint relations. Given the vector representations q for the question Q and g for the partial query G_i , we concatenate them and feed a multi-layer perceptron (MLP). The MLP outputs a scalar which we use as the semantic similarity $S_{sem}(Q, G_i)$. We describe in detail the encoding methods for the question and different relation forms in the main relation path. We also describe other design elements and the learning objective.

Encoding question. We encode a question Q using its token sequence and dependency structure. Since a complex question tends to be long, encoding its dependency tree captures any long-range dependencies. Let $\langle w_1, w_2, \dots, w_n \rangle$ be the tokens in Q , where seed (constraint) entity mentions have been replaced with w_E (w_C). We map the tokens to vectors $\langle q_1^w, q_2^w, \dots, q_n^w \rangle$ using an embedding matrix E_w and use an LSTM to encode the sequence to a latent vector q^w . Similarly, we encode the dependency tree into a latent vector q^{dep} .

Encoding main relation path. The main relation path can have different forms, a textual relation from \mathcal{K}_o or an ontological relation from \mathcal{K}_c . In order to collectively infer over them in the same space, we first align the textual relations to ontological relations. For instance, we find textual relations ‘is author of’, ‘written by’ can be aligned to ontological relation `book.author`. We describe how we derive the relation alignments in Sec. 6.4.2. Given a relation alignment, we encode each relation form i in the alignment to a latent vector r_i . We apply a max pooling over the latent vectors of different relations in the alignment to obtain a unified semantic representation over the different relation forms. Doing so enables the model to learn better representations of an ontological relation which has complementary textual relations.

To encode each relation form into vector r_i , we consider both sequence of tokens and ids [63]. For instance, the id sequence of the relation in Fig. 6.5 is `{book.author}`, while its token sequence is `{‘book’, ‘author’}`. We embed the tokens into vectors using an embedding matrix and use average embedding r^w as the token-level representation. We translate the relation directly using another embedding matrix E_r of relation paths to derive its id-level representation r_i^{id} . The vector representation of a path then is $r_i = [r_i^w; r_i^{id}]$.

Encoding constraints. Similarly, we encode the constraint relations c_i in by combining its token-level representation c_i^w and id-level representation c_i^{id} . Given the unified vector representation of a relation path, and the latent vectors of the constraint relations, we apply max pooling to obtain the compositional semantic representation g of the query.

Attention mechanism. Simple questions contain expressions for matching one main relation path. A complex question, however, has expressions for matching multiple relation paths, which could interfere with each other. For instance, words ‘college’ and ‘attend’ can distract the matching of the phrase ‘author of’ to the relation `book.author`. We mitigate this issue by improving the question representation using an attention mechanism [64]. The idea is to learn to emphasize parts of the question that are relevant to a context derived using the partial query vector g . Formally, given all hidden vectors h_t at time step $t \in \{1, 2, \dots, n\}$ of the token-level representation of the question, we derive a context vector c as the weighted sum of all the hidden states:

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

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

$$\alpha = \text{softmax}(W \tanh(W_q q^w + W_g g))$$

where W, W_g, W_q are network parameters. The attention weights indicate how much the model focuses on each token given a partial query.

Objective function. We concatenate the context vector c , question dependency vector q^{dep} and query vector g and feed to a multi-layer perceptron (MLP). It is a feed-forward neural network with two hidden layers and a scalar output neuron indicating the semantic similarity score $S_{sem}(q, G_i)$. We train the model using cross entropy loss,

$$loss = y \log(S_{sem}) + (1 - y) \log(1 - S_{sem})$$

where $y \in \{0, 1\}$ is a label indicating whether G_i is correct or not. Training the model requires a) an alignment of equivalent relation forms, and b) examples (question, partial query) pairs. We describe how we generate them given QA pairs.

6.4.2 Relation Alignment

An open KB has a huge vocabulary of relations. Aligning the textual relations to ontological relations for collective inference can become challenging if the textual relations are not canonicalized. We, first learn embeddings for the textual relations and cluster them to obtain canonicalized relation clusters [88]. For instance, a cluster can include both 'is author of' and 'authored'. We use the canonicalized textual relations to derive an alignment to the ontological relations. We derive this alignment based on the support entity pairs (s, o) for a pair of ontological relation and canonicalized textual relation. For instance, relations 'is author of' and `book.author` in our example question will share more entities than relations 'is author of' and `education.institution`.

6.4.3 Implicit Supervision

Obtaining questions with fully-annotated queries is expensive, especially when queries are complex. In contrast, obtaining answers is easier. In such a setting, the

quality of a query candidate is often measured indirectly by computing the F_1 score of its answers to the labeled answers [70]. However, for complex questions, answers to the partial queries may have little or no overlap with the labeled answers. We, therefore, adopt an alternative scoring strategy where we estimate the quality of a partial query as the best F_1 score of all its full query derivations. Formally, we compute a score $V(G_i^{(k)})$ for a partial query as:

$$V(G_i^{(k)}) = \max_{i \leq t \leq n-1} F_1(D_{t+1}^{(k)})$$

where D_t denotes the derivation at level t and n denotes the number of partial queries.

Such implicit supervision can be susceptible to spurious derivations which happen to evaluate to the correct answers but do not capture the semantic meaning of a question. We, thus, consider additional priors to promote true positive and false negative examples in the training data. We use $L(Q, G_i^{(k)})$ as the ratio of number of words in the relations of $G_i^{(k)}$ that are mentioned in the question Q . We also use $C(Q, G_i^{(k)})$ as the fraction of relation words that hit a small hand-crafted lexicon of co-occurring relation and question words. We estimate the quality of a candidate as: $V(G_i^{(k)}) + \gamma L(Q, G_i^{(k)}) + \delta C(Q, G_i^{(k)})$. We consider a candidate a positive example if its score is larger than a threshold (0.5) and negative otherwise.

6.5 Query Composition

In this chapter, we focus on constructing complex queries using a sequence of simple partial queries, each with one main relation path. Since the original question does not have to be chunked into simple questions, constructing computation plans for such questions is fairly simple. Heuristically, a computation plan can simply be derived by estimating the number of main relations (verb phrases) in the question and the dependency between them (subordinating or coordinating). We use a more

sophisticated model [85] to derive the computation plan. The post-order traversal of the plan yields the order in which partial queries should be executed.

Given a computation plan, we adopt a beam search and evaluate best k candidates for a partial query at each level. This helps maintain tractability in the large space of possible complex query derivations. The semantic matching model only independently scores the partial queries and not complete derivations. We, thus, need to find the best derivation that captures the meaning of the complex input question. To determine the best derivation, we aggregate the scores over the partial queries and consider additional features such as entities, structure of the query and targeted knowledge source. We train a log-linear model on a set of (question-answer) pairs using features such as semantic similarity scores, entity linking scores, number of constraints in the query, number of variables, number of relations, number of answer entities, number of knowledge sources targeted, average confidence score of the matching facts and average entity linking score of the arguments of the matching facts. Given the best scoring derivation, we translate it to a KB query and evaluate it to return answers to the question.

6.6 Experiments

We present experiments that show MULTIQUE outperforms existing KB-QA systems on complex questions. Our approach to construct queries from simple queries and aggregate multiple KBs is superior to methods which map questions directly to queries and use raw text instead.

6.6.1 Experimental Setup

Datasets. We use two benchmark QA datasets:

- **CompQWeb** [85]: A recent dataset with highly complex questions with compositions, conjunctions, superlatives and comparatives. It contains 34,689 questions,

split into 27,734 train, 3,480 dev and 3,475 test cases. For simplicity of evaluation, we only reserve questions with compositions and conjunctions (90% of the dataset).

- **WebQSP** [107]: It contains 4,737 questions split into 3,098 train and 1,639 test cases. Most of the questions are simple; only 4% questions have multiple constraints [108]. We evaluate on this dataset to demonstrate our proposed methods are effective on questions of varying complexity.

Knowledge Bases. We use the Freebase² dump as the curated KB. We construct an extracted KB using StanfordOpenIE [4] over the snippets released by [85] for CompQWeb and [84] for WebQSP.

Evaluation Metric. We report averaged F_1 scores of the predicted answers. We additionally compute precision@1 as the fraction of questions that were answered with the exact gold answer.

Baseline systems. We compare against two systems that can handle multiple knowledge sources.

- **GraftNet+** [84]: Given a question, it identifies a KB subgraph potentially containing the answer, annotates it with text and performs a binary classification over the nodes in the subgraph to identify the answer node(s). We point that it collects subgraphs using 2-hop paths from a seed entity. Since this cannot scale for complex questions which can have arbitrary length paths, we follow our query composition strategy to generate subgraphs. We annotate the subgraphs with snippets released with the datasets. We call this approach GraftNet+.
- **OQA** [43]: It is the first KB-QA system to combine curated KB and extracted KB. It uses a cascade of operators to paraphrase and parse questions to queries, and to rewrite and execute queries. It does not generate a unified representation of relation forms across the KBs. For comparison, we augment its knowledge source with our extracted KB and evaluate the model released by the authors.

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

Several other KB-QA systems [31, 1, 10] use only Freebase and handle simple questions with a few constraints. SplitQA [85] and MHQA [80] handle complex questions, but use web as the knowledge source.

Implementation Details. We used NVIDIA GeForce GTX 1080 Ti GPU for our experiments. We initialize word embeddings using GloVe [71] word vectors of dimension 300. We use BiLSTMs to encode the question token and dependency sequences. We use 1024 as the size of hidden layer of MLP and *sigmoid* as the activation function. More details can be found in supplementary material.

6.6.2 Results and Discussion

We evaluate several configurations. We consider candidates from curated KB as the only available knowledge source to answer questions and use it as a baseline (*cKB-only*). To demonstrate that inference over curated KB can benefit from open KB, we consider diverse relation forms of curated KB facts from open KB (*cKB+oKB*). Lastly, we downsample the curated KB candidates to 90%, 75% and 50% to simulate incompleteness in KB.

Effectiveness on complex questions. Our proposed system outperforms existing approaches on answering complex questions (Table 6.1). Even though both MULTIQUE and GraftNet+ use the same information sources, our semantic matching model outperforms node classification. Also, using extracted facts instead of raw text enables us to exploit the relations between entities in the text. We also achieve significantly higher F_1 than OQA that uses multiple KB but relies on templates for parsing questions to queries directly and does not deeply integrate information from multiple KBs. In contrast, we can construct complex query patterns from simple queries, and can infer over diverse relation forms in the KB facts. SplitQA [85] and MHQA [80] use a similar approach to answer complex questions using a sequence of simpler questions, but rely solely on noisy web data. Clearly, by combining the knowledge from curated

Method	CompQWeb	WebQSP
MULTIQUE (cKB-only)	31.24/37.61	61.16/69.84
MULTIQUE (cKB+oKB)	34.62/41.23	57.49/67.51
MULTIQUE (90%cKB+oKB)	27.15/30.21	55.47/65.42
MULTIQUE (75%cKB+oKB)	25.54/28.09	50.64/60.17
MULTIQUE (50%cKB+oKB)	18.57/20.51	41.72/50.82
GraftNet+ [84]	31.96/44.78	57.21/68.98
OQA [43]	0.42/42.85	21.78/32.63
SplitQA[85]	-/27.50	-
MHQA [80]	-/30.10	-

Table 6.1: Average F_1 / precision@1 of baseline systems and MULTIQUE in different configurations.

KB, we can answer complex questions more reliably.

Effectiveness on simple questions. An evaluation on simpler questions demonstrates that MULTIQUE can adapt to questions of varying complexity. We achieve the comparable F_1 score on the as other KB-QA systems that adopt an enumerate-encode-compare strategy. STAGG [106], a popular KB-QA system uses a similar approach for candidate generation but improves the results using feature engineering and by augmenting entity linking with external knowledge and only uses curated KB. MULTIQUE uses multiple KBs, and can be integrated with a better entity linking and scoring scheme for derivations.

KB completeness. Our results show that including information from extracted KB helps improve inference over ontological relations and facts for complex questions (as indicated by 3.38 F_1 gain in cKB+oKB). It instead hurts the performance on WebQSP dataset. This can be attributed to the coverage of the accompanying textual data sources of the two datasets. We found that for only 26% of the questions in WebQSP, an extracted fact could be aligned with a curated KB candidate. In

contrast, there were 55% such questions in the CompQWeb. This illustrates that considering irrelevant, noisy facts does not benefit when curated KB is complete. Such issues can be mitigated by using a more robust retrieval mechanism for text snippets or facts from extracted KB.

A KB-QA system must rely on an extracted KB when curated KB is incomplete. This is reflected in the dramatic increase in the percentage of hybrid queries when curated KB candidates were downsampled (e.g., from 17% to 40% at 90% completeness). As expected, the overall F_1 drops because the precise curated KB facts become unavailable. Despite the noise in extracted KBs, we found 5-15% of the hybrid queries found a correct answer. Surprisingly, we find 55% of the queries changed when the KB is downsampled to 90%, but 89% of them did not hurt the average F_1 . This indicates that the system could find alternative queries when KB candidates are dropped.

Ablation Study. Queries for complex questions often have additional constraints on the main relation path. 35% of the queries in CompQWeb had at least one constraint, while most of the queries (85%) in WebQSP are simple. Ignoring constraints in candidate generation and in semantic matching drops the overall F_1 score by 9.8% (8.6%) on CompQWeb (WebQSP) (see Table 6.2). Complex questions also are long and contain expressions for matching different relation paths. Including the attention mechanism helps focus on relevant parts of the question and improves the relation inference. We found F_1 drops significantly on CompQWeb when attention is disabled.

Re-ranking complete query derivations by additionally considering entity linking scores and query structure consistently helps find better queries. We examined the quality of top-k query derivations (see Table 6.3). For a large majority of the questions, query with the highest F_1 score was among the top-10 candidates. A better re-ranking model, thus, could help achieve higher F_1 score. We also observed that incorporating prior domain knowledge in deriving labels for partial queries at training was useful for complex questions.

Setup	CompQWeb	WebQSP
No constraints	31.23/37.87	52.53/60.84
No attention	26.92/31.24	40.29/51.86
No re-ranking	29.39/36.14	55.13/62.78
No prior	30.88/36.68	57.54/64.63

Table 6.2: Ablation results, average F_1 / precision@1, of MULTIQUE (cKB+oKB).

	CompQWeb		WebQSP	
	%	Avg. <i>best</i> F_1	%	Avg. <i>best</i> F_1
Top-1	35.11	34.62	69.12	57.49
Top-2	39.73	37.02	76.21	63.74
Top-5	51.12	42.08	85.05	70.00
Top-10	59.19	46.39	89.63	73.37

Table 6.3: Percentage of questions with the highest F_1 score in the top-k derivations, and the average best F_1 .

Qualitative Analysis. The datasets also provide queries over Freebase. We used them to analyze the quality of our training data and the queries generated by our system. We derive labels for each partial query candidate by comparing it to the labeled query. On an average, 4 candidates per question were labeled correct. We then compare them with the labels derived using implicit supervision. We found on average 3.06 partial queries were true positives and 103.08 were true negatives, with few false positives (1.72) and false negatives (0.78).

We further examined if the queries which achieve a non-zero F_1 were spurious. We compared the query components (entities, relations, filter clauses, ordering constraints) of such queries with labeled queries. We found high precision (81.89%) and recall (76.19%) of query components, indicating the queries were indeed precise.

Error Analysis. We randomly sampled 50 questions which achieve low F_1 score (< 0.1) and analyzed the queries manually. We found 38% errors were made because

of incorrect entities in the query. 92% of the entity linking errors were made at the first partial query. These errors get propagated because we find candidate queries using a staged generation. A better entity linking system can help boost the overall performance. 12% of the queries had an incorrect curated KB relation and 18% of the queries had an incorrect extracted KB relation. In a large fraction of cases (32%) the predicted and true relation paths were ambiguous given the question (e.g., `kingdom.rulers` vs `government` for “*Which queen presides over the location...*”). This indicates that relation inference is difficult for highly similar relation forms.

Since the extracted KB can contain more noisy tuples than the curated KB, we investigate how the overall performance is affected by the quality of the tuples being queried. We examined the randomly sampled questions which had achieved low F_1 score and found 70% of support tuples were indeed correct, indicating the errors were made at the inference. We found 10% of the errors were because incorrect tuples were extracted automatically and 20% of the errors were because the entities in the extracted tuples were incorrectly linked to the entities in the curated KB. These errors can be alleviated with an improved extraction and entity linking systems or with a human-in-the-loop data integration strategies. These techniques are beyond the scope of this dissertation.

6.7 Conclusion

We have presented a new KB-QA system that uses both curated and extracted KBs to answer complex questions. It composes complex queries using simpler queries each targeting a KB. It integrates an enumerate-encode-compare approach and a novel neural-network based semantic matching model to find partial queries. Our system outperforms existing state-of-the-art systems on highly compositional questions, while achieving comparable performance on simple questions.

CHAPTER VII

Concluding Remarks and Future Work

Much information is available to us as natural language text. This information becomes much easier to query and analyze if it is stored in a database. Therefore, many have studied how to build a knowledge base (KB), or a large-scale repository of facts about real-world entities, and relations between them, from natural language text. Traditionally, domain-specific ontologies have been used to determine how the knowledge in the text could be encoded as concepts and relationships between the concepts. Information Extraction (IE) systems only learn to extract relations that are already defined in the ontology.

The advent of Open Information Extraction (Open-IE) paradigm relaxed the need for a pre-specified relation vocabulary, focusing on shallow representation of the natural language text in the form of verbal and noun phrases (as relations) and their arguments (as entities). Open-IE provides a suitable structure for generic natural language texts because it requires neither a schema nor annotated corpora. This dissertation studies different issues involved in constructing and querying large-scale knowledge bases extracted automatically with Open-IE, specifically to support queries of varying complexity.

Chapter III describes a novel open information extractor that addresses the problem of loss of information and granularity at extraction. Instead of having a fixed

structure for the extraction problem that cannot accommodate the rich variations in natural language, we propose to have a single ‘fact’ partitioned into a set of tuples, which can be joined together to obtain the complete fact. We describe bootstrapping and extraction pattern learning techniques which can extract facts in an expressive, nest-tuple format given a natural language statement. Even if the representation for a tuple is simple, the information in nest-tuples is enriched with connections between tuples.

In Chapter IV, we describe a KB-QA system that uses an extracted knowledge base constructed using an Open-IE method such as ours, to answer complex questions with multiple constraints, such as those expressed via prepositional, conditional and adverbial modifiers. Since the background knowledge in extracted knowledge base could have diverse representations (binary, n-ary and nested), we present a schemaless querying mechanism that is agnostic to the underlying KB representation and can bridge lexical/syntactic gap between the query specification and relevant facts in the extracted KB.

We next study how the precise, but incomplete knowledge in curated KBs can be combined with the broad-coverage and noisier facts in extracted KBs. In Chapter V, we identify the challenges in answering compositional questions over curated KBs namely, constructing complex query patterns and learning from implicit supervision. We describe a KB-QA system that constructs complex query patterns from sequence of simple queries over a curated KB and uses a neural-network based semantic matching model to find semantically similar simple queries given a question. In Chapter VI, we describe novel extensions of the system to accommodate multiple KBs. We describe how hybrid query patterns can be constructed by targeting a specific KB for its constituent simple queries. We also present a neural-network based semantic matching model equipped for collective inference over different forms of relations in the two types of KBs. It leverages the alignment between ontological relations in a curated

KB and textual relations for collective inference.

This dissertation addresses some of the key challenges in extracting key facts from natural language text and supporting complex queries efficiently. However, there is a lot more that can be done to further improve the quality of extracted facts and performance of KB-QA systems powered by automatically extracted and curated KBs. First, open information extraction must be able to exploit the vast available curated knowledge sources. The core challenge in designing such ‘knowledge-aware’ Open-IE system is a knowledge module that can transfer the background knowledge as the extractor processes the unstructured text. The background knowledge is often limited and affects the generalization ability of such an approach. To cope with the sparsity of concepts in the background knowledge, facts must be encoded using continuous representations. Given an argument/relation phrase being processed by the extractor, the system should be able to retrieve the embeddings of related phrases in the background knowledge and integrate them to extract knowledge-aware assertions.

Second, future KB-QA systems should model whether a simple query is answerable from a given a KB or not. It should query the reliable, extracted KBs only when the curated KB lacks sufficient evidence. This could help improve overall precision. Furthermore, while resolving multiple query components simultaneous is beneficial, the inference could be improved if the question representation reflected all prior inferences. Lastly, more operations such as value comparisons and aggregations could be included in the computation plan and incorporated in the semantic matching model to support database-like analytical queries over extracted KBs.

Alternatively, the techniques presented in this dissertation can be used to design a human-in-the-loop system for knowledge acquisition. The system can provide answers to questions posed by users, who in turn can provide labels for the derived answers. The labels can be used as a feedback to improve both the automatic extraction of facts and knowledge-based question answering.

BIBLIOGRAPHY

BIBLIOGRAPHY

- [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. International World Wide Web Conferences Steering Committee, 2017.
- [2] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *Proc. Fifth ACM Conference on Digital Libraries '2000*, pages 85–94, 2000.
- [3] A. Akbik and A. Löser. Kraken: N-ary facts in open information extraction. In *Proc. AKBC '12*, pages 52–56, 2012.
- [4] G. Angeli, M. J. Premkumar, and C. D. Manning. Leveraging linguistic structure for open domain information extraction. In *Proc. ACL '15*, pages 26–31, 2015.
- [5] Y. Artzi, K. Lee, and L. Zettlemoyer. Broad-coverage ccg semantic parsing with amr. In *Proc. EMNLP '15*, pages 1699–1710, 2015.
- [6] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. *Proc. ISWC '07*, pages 722–735, 2007.
- [7] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [8] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction for the web. In *Proc. IJCAI '07*, volume 7, pages 2670–2676, 2007.
- [9] J. Bao, N. Duan, M. Zhou, and T. Zhao. Knowledge-based question answering as machine translation. In *Proc. ACL '14*, pages 967–976, 2014.
- [10] J.-W. Bao, N. Duan, Z. Yan, M. Zhou, and T. Zhao. Constraint-based question answering with knowledge graph. In *Proc. COLING '16*, pages 2503–2514, 2016.
- [11] H. Bast and E. Haussmann. Open information extraction via contextual sentence decomposition. In *Proc. IEEE-ICSC '13*, pages 154–159, 2013.

- [12] H. Bast and E. Haussmann. More informative open information extraction via simple inference. In *Proc. ECIR '14*, pages 585–590. Springer, 2014.
- [13] H. Bast and E. Haussmann. More accurate question answering on freebase. In *Proc. CIKM '15*, pages 1431–1440. ACM, 2015.
- [14] P. Baudis and J. Sedivý. Modeling of the question answering task in the yodaqa system. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction - 6th International Conference of the CLEF Association, CLEF 2015, Toulouse, France, September 8-11, 2015, Proceedings*, pages 222–228, 2015.
- [15] J. Berant and P. Liang. Semantic parsing via paraphrasing. In *Proc. ACL '14*, pages 1415–1425, 2014.
- [16] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *Proc. EMNLP '13*, volume 2, page 6, 2013.
- [17] N. Bhutani, H. Jagadish, and D. R. Radev. Nested propositions in open information extraction. In *Proc. EMNLP '16*, pages 55–64, 2016.
- [18] N. Bhutani, Y. Suhara, W. Tan, A. Y. Halevy, and H. V. Jagadish. Open information extraction from question-answer pairs. In *Proc. NAACL '19*, pages 2294–2305, 2019.
- [19] S. Bird, Y. Chen, S. B. Davidson, H. Lee, and Y. Zheng. Designing and evaluating an xpath dialect for linguistic queries. In *Proc. ICDE '06*, pages 52–52. IEEE, 2006.
- [20] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proc. SIGMOD '08*, pages 1247–1250. AcM, 2008.
- [21] A. Bordes, S. Chopra, and J. Weston. Question answering with subgraph embeddings. In *Proc. EMNLP '14*, pages 615–620, 2014.
- [22] A. Bordes, J. Weston, and N. Usunier. Open question answering with weakly supervised embedding models. In *Proc. ECMD-PKDD '14*, pages 165–180. Springer, 2014.
- [23] E. Brill, S. T. Dumais, and M. Banko. An analysis of the askmsr question-answering system. In *Proc. EMNLP '02*, 2002.
- [24] S. Brin. Extracting patterns and relations from the world wide web. In *Proc. WebDB '98*, pages 172–183. Springer, 1998.
- [25] Q. Cai and A. Yates. Large-scale semantic parsing via schema matching and lexicon extension. In *Proc. ACL '13*, pages 423–433, 2013.

- [26] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell. Toward an architecture for never-ending language learning. In *Proc. AAAI '10*, volume 5, page 3, 2010.
- [27] D. Chen, A. Fisch, J. Weston, and A. Bordes. Reading wikipedia to answer open-domain questions. In *Proc. ACL '17*, pages 1870–1879, 2017.
- [28] J. Christensen, S. Soderland, O. Etzioni, et al. Semantic role labeling for open information extraction. In *Proc. NAACL '10*, pages 52–60, 2010.
- [29] P. Chubak and D. Rafiei. Efficient indexing and querying over syntactically annotated trees. *Proc. VLDB '12*, 5(11):1316–1327, 2012.
- [30] L. Cui, F. Wei, and M. Zhou. Neural open information extraction. In *Proc. ACL '18*, pages 407–413, 2018.
- [31] W. Cui, Y. Xiao, H. Wang, Y. Song, S.-w. Hwang, and W. Wang. Kbcqa: learning question answering over qa corpora and knowledge bases. *Proc. VLDB '17*, 10(5):565–576, 2017.
- [32] R. Das, M. Zaheer, S. Reddy, and A. McCallum. Question answering on knowledge bases and text using universal schema and memory networks. In *Proc. ACL '17*, pages 358–365, 2017.
- [33] R. Das, M. Zaheer, S. Reddy, and A. McCallum. Question answering on knowledge bases and text using universal schema and memory networks. *arXiv preprint arXiv:1704.08384*, 2017.
- [34] R. Das, S. Dhuliawala, M. Zaheer, L. Vilnis, I. Durugkar, A. Krishnamurthy, A. Smola, and A. McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *Proc. ICLR '18*, 2018.
- [35] M.-C. De Marneffe, B. MacCartney, C. D. Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proc. LREC '06*, volume 6, pages 449–454, 2006.
- [36] L. Del Corro and R. Gemulla. Clausie: clause-based open information extraction. In *Proc. IW3C2 '13*, pages 355–366, 2013.
- [37] B. Dhingra, K. Mazaitis, and W. W. Cohen. Quasar: Datasets for question answering by search and reading. *arXiv preprint arXiv:1707.03904*, 2017.
- [38] B. Dhingra, D. Pruthi, and D. Rajagopal. Simple and effective semi-supervised question answering. In *Proc. NAACL '18*, pages 582–587, 2018.
- [39] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial intelligence*, 165(1):91–134, 2005.

- [40] O. Etzioni, A. Fader, J. Christensen, and S. Soderland. Open information extraction: The second generation. In *Proc. IJCAI '11*, 2011.
- [41] A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *Proc. EMNLP '11*, pages 1535–1545, 2011.
- [42] A. Fader, L. Zettlemoyer, and O. Etzioni. Paraphrase-driven learning for open question answering. In *Proc. ACL '13*, volume 1, pages 1608–1618, 2013.
- [43] A. Fader, L. Zettlemoyer, and O. Etzioni. Open question answering over curated and extracted knowledge bases. In *Proc. SIGKDD '14*, pages 1156–1165. ACM, 2014.
- [44] Y. Feng, S. Huang, D. Zhao, et al. Hybrid question answering over knowledge base and free text. In *Proc. COLING '16*, pages 2397–2407, 2016.
- [45] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.
- [46] K. M. Hermann, T. Kociský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching machines to read and comprehend. In *NIPS*, 2015.
- [47] D. Hewlett, A. Lacoste, L. Jones, I. Polosukhin, A. Fandrianto, J. Han, M. Kellecey, and D. Berthelot. Wikireading: A novel large-scale language understanding task over wikipedia. In *Proc. ACL '16*, 2016.
- [48] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Proc. Artificial Intelligence '13*, 194:28–61, 2013.
- [49] R. Hoffmann, C. Zhang, X. Ling, L. Zettlemoyer, and D. S. Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proc. ACL '11*, pages 541–550, 2011.
- [50] S. Hu, L. Zou, J. X. Yu, H. Wang, and D. Zhao. Answering natural language questions by subgraph matching over knowledge graphs. *IEEE Trans. Knowl. Data Eng.*, 30(5):824–837, 2018.
- [51] M. Iyyer, W. Yih, and M. Chang. Search-based neural structured learning for sequential question answering. In *Proc. ACL '17*, pages 1821–1831, 2017.
- [52] D. Khashabi, T. K. A. Sabharwal, and D. Roth. Question answering as global reasoning over semantic abstractions. In *Proc. AAAI '18*, 2018.
- [53] T. Khot, A. Sabharwal, and P. Clark. Answering complex questions using open information extraction. *arXiv preprint arXiv:1704.05572*, 2017.

- [54] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proc. ICLR '15*, 2015.
- [55] T. Kwiatkowski, E. Choi, Y. Artzi, and L. Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. In *Proc. EMNLP '13*. Association for Computational Linguistics (ACL), 2013.
- [56] 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.
- [57] D. B. Lenat, R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd. CYC: toward programs with common sense. *Commun. ACM*, 33(8):30–49, 1990.
- [58] F. Li and H. Jagadish. Constructing an interactive natural language interface for relational databases. *Proc. VLDB '14*, 8(1):73–84, 2014.
- [59] F. Li and H. V. Jagadish. Nalir: an interactive natural language interface for querying relational databases. In *Proc. SIGMOD '14*, pages 709–712. ACM, 2014.
- [60] C. Liang, J. Berant, Q. V. Le, K. D. Forbus, and N. Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Proc. ACL '17*, pages 23–33, 2017.
- [61] P. Liang, M. I. Jordan, and D. Klein. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446, 2013.
- [62] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proc. AAAI '15*, volume 15, pages 2181–2187, 2015.
- [63] K. Luo, F. Lin, X. Luo, and K. Q. Zhu. Knowledge base question answering via encoding of complex query graphs. In *Proc. EMNLP '18*, pages 2185–2194, 2018.
- [64] T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In *Proc. EMNLP '15*, pages 1412–1421, 2015.
- [65] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *Proc. ACL '09*, pages 1003–1011. ACL, 2009.
- [66] N. Nakashole and T. M. Mitchell. A knowledge-intensive model for prepositional phrase attachment. In *Proc. ACL '15*, pages 365–375, 2015.
- [67] F. Niu, C. Zhang, C. Re, and J. W. Shavlik. Deepdive: Web-scale knowledge-base construction using statistical learning and inference. *Proc. VLDS '12*, 12: 25–28, 2012.

- [68] H. Pal et al. Donyms and compound relational nouns in nominal open ie. In *Proc. AKBC '16*, pages 35–39, 2016.
- [69] M. A. Paredes-Valverde, M. Á. Rodríguez-García, A. Ruiz-Martínez, R. Valencia-García, and G. Alor-Hernández. Onli: an ontology-based system for querying dbpedia using natural language paradigm. *Proc. Expert Syst. Appl. '15*, 42(12):5163–5176, 2015.
- [70] H. Peng, M. Chang, and W. Yih. Maximum margin reward networks for learning from explicit and implicit supervision. In *Proc. EMNLP '17*, pages 2368–2378, 2017.
- [71] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *Proc. EMNLP '14*, pages 1532–1543, 2014.
- [72] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100, 000+ questions for machine comprehension of text. In *Proc. EMNLP '16*, pages 2383–2392, 2016.
- [73] E. Riloff. Automatically generating extraction patterns from untagged text. In *Proc. AAAI '96*, pages 1044–1049, 1996.
- [74] D. Saha, A. Floratou, K. Sankaranarayanan, U. F. Minhas, A. R. Mittal, and F. Özcan. Athena: An ontology-driven system for natural language querying over relational data stores. *Proc. VLDB '16*, 9(12):1209–1220, 2016.
- [75] M. Schmitz, R. Bart, S. Soderland, O. Etzioni, et al. Open language learning for information extraction. In *Proc. EMNLP '12*, pages 523–534, 2012.
- [76] Y. Shen, P.-S. Huang, J. Gao, and W. Chen. Reasonet: Learning to stop reading in machine comprehension. In *Proc. SIGKDD '17*, pages 1047–1055. ACM, 2017.
- [77] Y. Shinyama and S. Sekine. Preemptive information extraction using unrestricted relation discovery. In *Proc. NAACL-HLT '06*, pages 304–311. Association for Computational Linguistics, 2006.
- [78] A. Singhal. Introducing the knowledge graph: things, not strings. *Official google blog*, 2012.
- [79] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine learning*, 34(1-3):233–272, 1999.
- [80] L. Song, Z. Wang, M. Yu, Y. Zhang, R. Florian, and D. Gildea. Exploring graph-structured passage representation for multi-hop reading comprehension with graph neural networks. *arXiv preprint arXiv:1809.02040*, 2018.
- [81] G. Stanovsky, J. Michael, L. Zettlemoyer, and I. Dagan. Supervised open information extraction. In *Proc. ACL '18*, pages 885–895, 2018.

- [82] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.
- [83] H. Sun, H. Ma, W.-t. Yih, C.-T. Tsai, J. Liu, and M.-W. Chang. Open domain question answering via semantic enrichment. In *Proc. IW3C2 '15*, pages 1045–1055. International World Wide Web Conferences Steering Committee, 2015.
- [84] H. Sun, B. Dhingra, M. Zaheer, K. Mazaitis, R. Salakhutdinov, and W. Cohen. Open domain question answering using early fusion of knowledge bases and text. In *Proc. EMNLP '18*, pages 4231–4242, 2018.
- [85] A. Talmor and J. Berant. The web as a knowledge-base for answering complex questions. In *Proc. NAACL-HLT '18*, pages 641–651, 2018.
- [86] M. Tu, G. Wang, J. Huang, Y. Tang, X. He, and B. Zhou. Multi-hop reading comprehension across multiple documents by reasoning over heterogeneous graphs. *arXiv preprint arXiv:1905.07374*, 2019.
- [87] 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, 2012.
- [88] S. Vashishth, P. Jain, and P. Talukdar. Cesi: Canonicalizing open knowledge bases using embeddings and side information. In *Proc. WWW '18*, pages 1317–1327. WWW, 2018.
- [89] D. Vrandečić. Wikidata: a new platform for collaborative data collection. In *Proc. WWW '12*, pages 1063–1064, 2012.
- [90] 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.
- [91] Y. Wang, J. Berant, P. Liang, et al. Building a semantic parser overnight. In *Proc. ACL '15*, pages 1332–1342, 2015.
- [92] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *Proc. AAAI '14*, volume 14, pages 1112–1119, 2014.
- [93] J. Weston, A. Bordes, O. Yakhnenko, and N. Usunier. Connecting language and knowledge bases with embedding models for relation extraction. *arXiv preprint arXiv:1307.7973*, 2013.
- [94] J. Weston, S. Chopra, and A. Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

- [95] G. Wiese, D. Weissenborn, and M. L. Neves. Neural domain adaptation for biomedical question answering. In *Proc. CoNLL '17*, pages 281–289, 2017.
- [96] F. Wu and D. S. Weld. Open information extraction using wikipedia. In *Proc. ACL '10*, pages 118–127, 2010.
- [97] W. Wu, H. Li, H. Wang, and K. Q. Zhu. Probase: A probabilistic taxonomy for text understanding. In *Proc. SIGMOD '12*, pages 481–492. ACM, 2012.
- [98] W. Xiong, T. Hoang, and W. Y. Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *Proc. EMNLP '17*, pages 564–573, 2017.
- [99] K. Xu, S. Reddy, Y. Feng, S. Huang, and D. Zhao. Question answering on freebase via relation extraction and textual evidence. In *Proc. ACL '16*, 2016.
- [100] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum. Natural language questions for the web of data. In *Proc. EMNLP '12*, pages 379–390. Association for Computational Linguistics, 2012.
- [101] M. Yahya, S. Whang, R. Gupta, and A. Y. Halevy. Renoun: Fact extraction for nominal attributes. In *Proc. EMNLP '14*, pages 325–335, 2014.
- [102] S. Yang, Y. Wu, H. Sun, and X. Yan. Schemaless and structureless graph querying. *Proc. VLDB '14*, 7(7):565–576, 2014.
- [103] Y. Yang and M. Chang. S-MART: novel tree-based structured learning algorithms applied to tweet entity linking. In *Proc. ACL '15*, pages 504–513, 2015.
- [104] X. Yao and B. Van Durme. Information extraction over structured data: Question answering with freebase. In *Proc. ACL '14*, pages 956–966, 2014.
- [105] A. Yates, M. Cafarella, M. Banko, O. Etzioni, M. Broadhead, and S. Soderland. Textrunner: open information extraction on the web. In *Proc. NAACL Demo '07*, pages 25–26, 2007.
- [106] W.-t. Yih, M.-W. Chang, X. He, and J. Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proc. ACL '15*, pages 1321–1331, 2015.
- [107] W.-t. Yih, M. Richardson, C. Meek, M.-W. Chang, and J. Suh. The value of semantic parse labeling for knowledge base question answering. In *Proc. ACL '16*, volume 2, pages 201–206, 2016.
- [108] P. Yin, N. Duan, B. Kao, J. Bao, and M. Zhou. Answering questions with complex semantic constraints on open knowledge bases. In *Proc. CIKM '15*, pages 1301–1310. ACM, 2015.

- [109] M. Yu, W. Yin, K. S. Hasan, C. N. dos Santos, B. Xiang, and B. Zhou. Improved neural relation detection for knowledge base question answering. In *Proc. ACL '17*, pages 571–581, 2017.
- [110] Y. Zhang, S. He, K. Liu, and J. Zhao. A joint model for question answering over multiple knowledge bases. In *Proc. AAAI '16*, pages 3094–3100, 2016.
- [111] 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.
- [112] W. Zheng, J. X. Yu, L. Zou, and H. Cheng. Question answering over knowledge graphs: question understanding via template decomposition. *Proc. VLDB '18*, 11(11):1373–1386, 2018.
- [113] 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 '14*, pages 313–324. ACM, 2014.