Question Answering and Query Processing for Extended Knowledge Graphs

Mohamed Yahya

Dissertation
zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

Saarbrücken
2016

Abstract

Knowledge graphs have seen wide adoption, in large part owing to their schema-less nature that enables them to grow seamlessly, allowing for new relationships and entities as needed. With this rapid growth, several issues arise: (i) how to allow users to query knowledge graphs in an expressive and user-friendly manner, which shields them from all the underlying complexity, (ii) how, given a structured query, can we return satisfactory answers to the user despite possible mismatches between the query vocabulary and structure and the knowledge graph, and (iii) how to automatically acquire new knowledge, which can be fed into a knowledge graph. In this dissertation, we make the following contributions to address the above issues:

- We present DEANNA, a framework for question answering over knowledge graphs, allowing users to easily express complex information needs using natural language and obtain tuples of entities as answers thereby taking advantage of the structure in the knowledge graph.
- We introduce TriniT, a framework that compensates for unsatisfactory results of structured queries over knowledge graphs, either due to mismatches with the knowledge graph or the knowledge graph's inevitable incompleteness. TriniT tackles the two issues by extending the knowledge graph using information extraction over textual corpora, and supporting query relaxation where a user's query is rewritten in a manner transparent to the user to compensate for any mismatches with the data.
- We present ReNoun, an open information extraction framework for extracting binary relations mediated by noun phrases and their instances from text. Our scheme extends the state-of-the-art in open information extraction which has thus far focused on relations mediated by verbs.

Our experimental evaluations of each of the above contributions demonstrate the effectiveness of our methods in comparison to state-of-the-art approaches.

Kurzfassung

Der Einsatz von Wissensgraphen erfreut sich großer Beliebtheit, die vor allem der schemafreien Natur der Graphen geschuldet ist. Diese ermöglicht ein reibungsloses Anwachsen des Graphen, so dass neue Relationen und Entitäten je nach Bedarf hinzugefügt werden können. Durch dieses rapide Anwachsen des Graphen treten allerdings auch einige Fragestellungen auf: (i) Wie kann man Nutzern ermöglichen, Wissensgraphen in einer ausdrucksstarken und zugleich nutzerfreundlichen Weise anzufragen, die die Nutzer von der zugrundeliegenden Komplexität abschirmt, (ii) wie können für eine strukturierte Suchanfrage trotz möglicher Diskrepanzen in Vokabular und Struktur zwischen Suchanfrage und Wissensgraph zufriedenstellende Antworten geliefert werden, und (iii) wie kann neues Wissen automatisiert akquiriert werden, um es in einen Wissensgraphen zu integrieren? In der vorliegenden Dissertation werden die folgenden Beiträge entwickelt, um die obigen Problemstellungen zu adressieren:

– Wir präsentieren mit DEANNA ein Frage-Antwort-System für Wissensgraphen, das Nutzern ermöglicht, auf einfache Art und Weise komplexe Informationsbedürfnisse natürlichsprachlich auszudrücken. Die Struktur des Wissensgraphen wird dabei dahingehend ausgenutzt, das die Antworten als Entitätentupel ausgegeben werden.

– Mit TriniT entwickeln wir ein Framework, dass unbefriedigende Ergebnisse für strukturierte Suchanfragen auf Wissensgraphen kompensiert. Dabei werden sowohl Fehltreffer als auch unvermeidbare Lücken im Wissensgraphen berücksichtigt. Beide Probleme werden durch TriniT dadurch angegangen, dass der Wissensgraph mithilfe von Methoden der Informationsextraktion aus Textkorpora erweitert wird, und dass Suchanfragen relaxiert werden. Dafür wird die Suchanfrage eines Nutzers auf transparente und für den Nutzer nachvollziehbare Weise umgeschrieben, um Fehltreffer auszugleichen.

– Außerdem stellen wir ReNoun vor, ein Framework für Open Information Extraction zum Extrahieren von binären Relationen, die durch Nominalphrasen ausgedrückt werden. Unser Schema erweitert dabei den aktuellen

Forschungsstand im Bereich der offenen Informationsextraktion, da bislang der Fokus auf verbalen Relationen lag.

Unsere experimentellen Evaluierungen der oben genannten Methoden und Systeme verdeutlichen die Effektivität unserer Methoden im Vergleich zu State-of-the-Art Ansätzen.

Acknowledgments

This dissertation would not have been possible without the support of many people. I am thankful to Gerhard Weikum for taking me on as his student, for his mentoring, and for the countless opportunities he afforded me over the past five years. Gerhard is a brilliant scientist and leader. The breadth and depth of his knowledge is impressive and he has a keen eye for detail. I hope to be like him one day.

I am also thankful to Klaus Berberich. He provided me with a great deal of help and guidance over the past five years. He was always ready to hear me and clarify any points of confusion I have. I enjoyed our lengthy discussions and the cool topics we explored together. I also thank Shady Elbassiouni for his help at the start of my PhD.

I thank Volker Tresp for the collaboration in the early stages of my doctoral studies and for hosting me for an internship at Siemens Corporate Research and Technology in Munich. I also thank him for agreeing to review my dissertation.

I wish to thank Alon Halevy for taking me on as an intern. I greatly enjoyed my time at Google in Mountain View, and our conversations. I learned a great deal from him. I also thank Steven Whang for his help and patience during my Google internship and afterwards. He was instrumental in the ReNoun paper seeing the light of day. I also thank Rahul Gupta for his help and guidance during my time there.

I thank my colleagues in the Database and Information Systems group at MPI for the nice time. I learned a great deal from many people at D5, either by talking to them or just observing them work. Of particular note are my two officemates Johannes and Niket, our office neighbors Christina and Luciano, and Abdalghani. I thank Jannik for helping me with the German translation of the abstract of this dissertation.

I wish to thank Hinrich Schütze for agreeing to review my dissertation and for his feedback, which was very helpful in improving the dissertation.

Thanks to the close circle of friends who made life in Saarbrücken bearable over the past seven years: Ahmad Barghash and Fidaa Abed for their infinite supply of wisdom, which immensely influenced how I see the world; the Macedonians for their hospitality and great sense of humor; the Romanians for their strong opinions; the Germans and Czechs for teaching me how to be precise and intentional.

My parents, Buthaina and Adnan, have invested a great deal in me and my siblings. I am indebted to them beyond repayment. I thank my siblings, Fatima, Mariam, and Omar, for their encouragement. I apologize to all of them for occasionally dropping off the radar, particularly around deadlines.

Contents

1. Introduction

1.1. Motivation

Traditionally, users interact with search engines by providing them with keywords and getting a list of documents that best match these keywords. While this style of querying is satisfactory in simple settings, it often leaves more to be desired. The two major shortcomings of this paradigm concern answer granularity and query expressiveness. Now, more than ever before, these two issues need close consideration.

Documents are not always the appropriate way to answer a query. Often, a user is interested in more focused answers. Entity search (ES) has been proposed as an extension of traditional corpus-based document retrieval in response to keyword queries. While this has shown rapid progress in the last five years, it is limited in two fundamental and closely tied ways. First, in entity search a query result is a set of individual entities. This means that queries asking for entity pairs that stand in a certain relationship, such as *"former regulatory agency members who works for a lobbying organization"*, cannot be answered by ES systems. The second limitation of ES, which can also be seen in the example we just gave, comes from its reliance on keyword queries that have limited expressiveness as we detail below.

Keyword queries are highly "telegraphic" (Sawant and Chakrabarti, 2013), which means that they have limited expressiveness. With this limitation, users cannot express complex information needs. Even for seemingly simple queries, telegraphic queries cannot capture the subtleties related to semantics. A telegraphic query asking for *"john kennedy predecessor"* issued to a typical ES system will have equal chances of returning `LyndonJohnson`, whom Kennedy preceded, and `DwightEisenhower`, who preceded Kennedy. Traditionally, question answering has been proposed to overcome this problem. Here, queries are natural language questions. Past research has focused on the problem of factoid question answering where a question has one or few answers that can be, crucially, found in a single textual document.

What is needed is systems that can take complex information needs in the form of questions and return focused answers possibly in the form of tuples of entities. For the example above, we would like a system to take the question *"Which former regulatory agency members work for a lobbying organization?"* and return two answers: (`MichaelPowell`, `NCTA`) and (`MeredithBaker`, `CTIA`). The need to support such information needs has been brought on by the new modalities for interacting with search engines. For example, people using their mobile devices to query verticals have very limited space to view answers, and therefore prefer crisp answers in the form of entities. Digital assistants embedded in smartphones and cars are built around voice interaction where users can, with minimal effort, express complex information needs and, again, expect precise and concise answers.

As a first step towards satisfying complex information needs, organizations maintain knowledge graphs: collections of facts expressed in the form of triples composed of a subject and an object, connected by a predicate. Examples are:

`IngridBergmann actedIn Casablanca(movie)`

and

`NCTA head MichaelPowell`.

Knowledge graphs get their power from their ability to express crisp relationships that hold between entities, which allows them to be the source of crisp answers.

Such knowledge graphs have become ubiquitous. They form the backbone of social networks like Facebook, Twitter, and Yelp, specialized databases such as IMDB and Last.fm, and general knowledge such as Freebase, DBpedia, and Yago. Web search engines are capable of returning answers to very simple telegraphic queries from these knowledge graphs (e.g., *"casablanca actors"*, *"Saarbrücken restaurants"*). This is done by translating such simple queries to a formal query language used to query their underlying knowledge graph.

This trend poses research challenges along several dimensions:

- **Query Language:** While telegraphic queries cover a significant portion of queries observed by Web search engines, they are inherently unable to express more complex relationship-centric information needs. Moreover, telegraphic queries are unnatural in certain modalities such as voice-based interaction. For these scenarios, natural language is the most appropriate mode of interaction with search engines. This requires a framework for translating potentially complex user queries posed as questions to a formal representation that can be answered by the knowledge graph.

- **Query Processing:** In an ideal setting, evaluating a formal query corresponding to a user's information need over a knowledge graph will result in the desired answers. In practice, however, this is often not the case. Knowledge graphs have gaps in their knowledge, which means that a perfectly formulated query might not return all the expected results, if any at all. Moreover, formulating the correct formal query, either manually by an expert user, or automatically by a machine, is a non-trivial error-prone task. This requires that the engine evaluating formal queries be able to account for these problems by adjusting the user's query and utilizing external data sources to return the desired answers.

- **Knowledge Acquisition:** Both the translation of a question to a formal representation and subsequent adjustment of the formal query require resources external to the knowledge graph. These include dictionaries that connect natural language utterances to knowledge graph entities and facts, and new sources of facts to remedy any incompleteness in the knowledge graph. While much research has gone into building such resources, there are shortcomings that still need to be addressed. One such shortcoming is in the extraction of relations mediated by noun phrases (e.g., *'country of origin'*) following the open information extraction paradigm.

1.2. Contributions

This dissertation presents three contributions that address the research challenges outlined above:

- **Query language:** In order to ensure accessibility and expressiveness, we allow users to issue queries in the form of natural language questions. We cannot assume that the average user of a search engine has the technical competence to formulate a query in a formal language that expresses her information need. Even in the case of a technically competent user, the sheer size of a typical knowledge graph and the lack of a schema mean that formulating such a formal query can be a daunting task. Natural language allows users to express complex queries with multiple predicates and join conditions in an unambiguous manner for humans (a whole different story for machines).

  To accommodate natural language questions as a query language, we present DEANNA: a system for translating natural language questions to formal queries over a knowledge graph. DEANNA uses the select-project-join subset of SPARQL as its target query language. The translation boils down to solving three forms of ambiguity in query segmentation, mapping segments to knowledge graph primitives, and determining the dependencies between these primitives. As the solution to each of the three forms of ambiguity can inform the solution to the other two, we perform the disambiguation jointly. We formulate the disambiguation problem as an Integer Linear Program (ILP), whose solution gives us the intended formal query. Intuitively, the ILP looks for the most likely interpretation of the question (the ILP's objective function) that makes sense (the ILP's constraints).

- **Query Processing**: Ideally, once we have the formal query corresponding to a question, obtaining the desired answers is a simple matter of evaluating the query over the underlying knowledge graph. Unfortunately, however, this is often not the case in practice. One major issue with knowledge graphs is their inherent incompleteness. Other problems include queries that contain some incorrect knowledge graph primitives (with respect to the user's intention), or queries formulated assuming a schema different than that of the knowledge graph.

  To remedy the problem of semantic queries with unsatisfactory results, we propose TriniT: a system for relaxed query evaluation and answer scoring over extended knowledge graphs. TriniT evaluates queries over an extension of the basic knowledge graph. This extension allows for facts that mix knowledge graph primitives and textual phrases. Such facts are obtained through various information extraction schemes with the goal of boosting recall. TriniT also allows for weighted relaxation rules for reformulating the original user query so that relevant answers not returned by the original query over the original knowledge graph can be returned using the extended knowledge graph. To accommodate relaxed query evaluation, TriniT supports answer scoring and top-k query processing. A language-model based scoring scheme assigns scores to answers, so that users can consume a limited number of top-scoring results based on their needs. The top-k query processing scheme ensures that query processing in our setting terminates as early as possible without the need for exhaustive exploration of the entire query space introduced by the relaxations.

- **Knowledge Acquisition:** Semantic query translation and query relaxation rely on resources external to the knowledge graph. Two important resources are additional facts for knowledge graph extension and lexicons for connecting the semantic items in a knowledge graph to natural language. There has been much research on both problems, with several publicly available datasets that we utilize. However, there is a critical gap that we need to fill: the extraction of facts centered around noun phrases. We introduce ReNoun, an open information extraction framework capable of discovering noun phrases that can serve as relations (e.g., '*country of origin*') and extracting facts centered around them from text. ReNoun also includes a scheme for scoring extracted facts, which reflects the likelihood that a fact is actually expressed by the text. The facts extracted by ReNoun are used for both knowledge graph extension and finding paraphrases for knowledge graph predicates.

The contributions this dissertation makes lie at the intersection of natural language processing, information retrieval, and database systems. This dissertation is based on results presented at the following conferences:

- WSDM 2016 (Yahya et al., 2016)
- EMNLP 2014 (Yahya et al., 2014)
- CIKM 2013 (Yahya et al., 2013)
- EMNLP 2012 (Yahya et al., 2012b)
- WWW 2012 (Yahya et al., 2012a)

1.3. Organization of the Dissertation

This dissertation is organized as follows. Chapter 2 introduces the general framework of knowledge graphs, how they can be queried, and how they can be extended by means of knowledge acquisition. It also presents the state-of-the-art for each of these topics. Chapter 3 presents DEANNA, our framework for question answering over knowledge graphs. In Chapter 4 we present TriniT, a framework that allows for flexible querying of knowledge graphs by remedying any mismatches between a structured query and the knowledge graph. TrinT works by combining knowledge graph extension and query relaxation. Knowledge graph extension results in facts automatically extracted from textual documents being used to make up for any incompleteness in the knowledge graph. Query relaxation provides the means to automatically rewrite a user's query to align it with the underlying extended knowledge graph ensuring better effectiveness through improved recall.

Chapter 5 focuses on the problem of knowledge acquisition whereby knowledge graphs can be extended with new facts. More concretely, it tackles the problem of open information extraction for relations mediated through noun phrases. Finally, in Chapter 6 we conclude this dissertation, recap its main contributions and present possible avenues for future investigation.

2. Background and Prior Work

With the abundance of data publicly available on the Web or privately held by organizations and individuals comes the problem of effectively querying and exploring this data. To quote John Naisbitt, "we are drowning in information but starved for knowledge" (Naisbitt, 1982). This dissertation is focused on giving people access to the knowledge in this data. The dissertation presents three contributions towards this end on the topics of question answering, query evaluation, and knowledge acquisition. This chapter introduces the concepts needed in the rest of this dissertation, motivates each of our contributions, and places these contributions in their context in the relevant research communities.

We start in Section 2.1 by presenting the triple-based data model we adopt for expressing knowledge, that of knowledge graphs, and how data in this model can be queried using triple pattern queries. We also introduce three concrete knowledge graphs that we use in the rest of this dissertation.

We next introduce the concept of knowledge acquisition in Section 2.2, focusing on two concrete tasks: named entity recognition and disambiguation, and fact extraction. This section shows how knowledge graphs, introduced in the previous one, can be constructed.

In Section 2.3, we look at the general problem of querying for knowledge. Here, we consider the various alternatives, focusing on their expressiveness and usability.

2.1. Knowledge Graphs

In our setting, the goal of a knowledge graph is to describe a set of entities E, which are uniquely identifiable things such as `Rome`, `IngridBergaman`, or `Casablanca(movie)`. This is achieved by connecting them to other entities in E or to literals L using a set of predicates P (e.g., `bornIn`, `bornOn`), or by grouping them into classes C (e.g., `actor`, `movie`, `ItalianCity`). Each such connection or grouping is called a fact, which is the basic unit of knowledge in our model.

Definition 2.1 (Fact)**.** Given a set of entities E, classes C, predicates P, and literals L, a fact f is a triple $f \in E \cup C \times P \times E \cup C \cup L$. The components of a triple, in order, are called the subject, predicate, and object.

Entities, classes and predicates in a KG are collectively called semantic items:

Definition 2.2 (Semantic Item)**.** Given a KG, a semantic item s is a member of the set $S = E \cup C \cup P$.

We discuss the various types of semantic items we deal with in this dissertation below. Literals are constant values used to describe semantic items, but themselves are not described by a knowledge graph, hence, they do not appear as subjects in a fact. Typical examples of literals are numbers, strings, and dates. We will interchangeably refer to facts as triples in this dissertation. We call this model for representing knowledge the subject-predicate-object (SPO) model or triples model. This model serves as the basis for the RDF data model. Figure 2.1(a) shows examples of facts.

A collection of distinct facts is called a knowledge graph, as given in Definition 2.3

Definition 2.3 (Knowledge Graph)**.** A knowledge graph (KG) is a set of facts, $KG \subseteq E \cup C \times P \times E \cup C \cup L$.

Knowledge graphs are also called knowledge bases. A fact in a KG can be viewed as a labeled edge (the predicate) connecting two nodes (the subject and object), with a collection of facts forming a graph, hence the name knowledge graph. Figure 2.1(a) shows an example KG, part of which is depicted graphically in Figure 2.1(b).

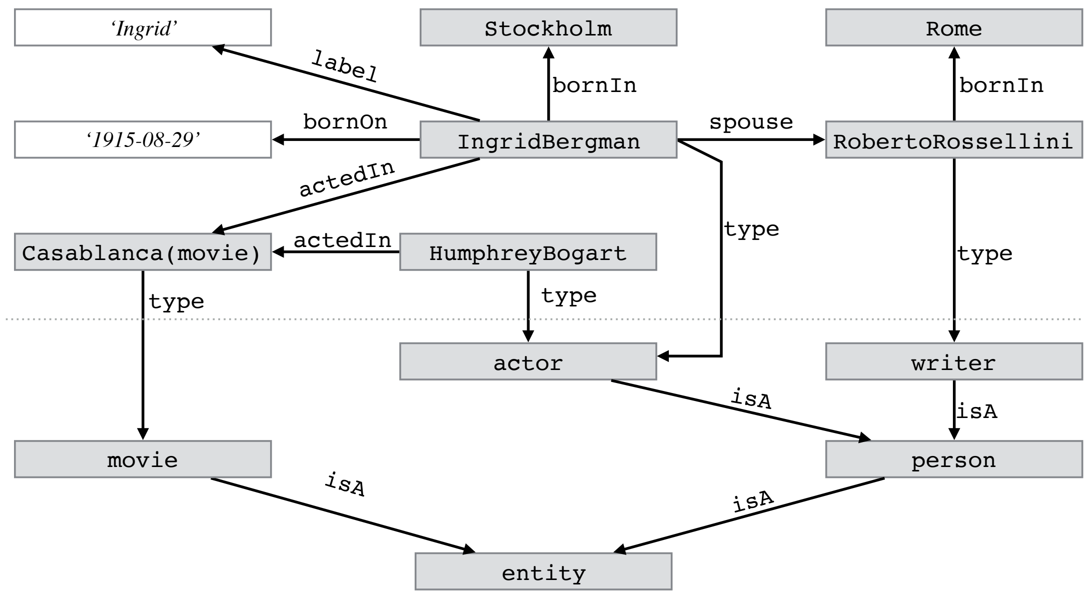
2.1.1. Entities

Entities are the central semantic item of a KG. In fact, the goal of a KG is to describe entities by connecting them to other entities, classes, or literals using predicates, as we stated above. What constitutes an entity can be debated and is often domain and application specific (Sowa, 2000). For this dissertation, the following definition fits our needs: "any abstract or concrete thing that is uniquely identifiable is an entity" (Suchanek, 2009).

For entities, like other semantic items, it is important to point out the distinction between an entity's id and its names. For example, the 1942 movie directed by Michael Curtiz has the unique id `Casablanca(movie)`. This movie can be referred to by multiple names such as '*Casablanca*' and '*Casablanca the movie*'. The name '*Casablanca*' is shared between this entity and the entity `Casablanca`,

#	**Subject**	**Predicate**	**Object**
1	IngridBergman	bornIn	Stockholm
2	IngridBergman	spouse	RobertoRossellini
3	RobertoRossellini	bornIn	Rome
4	IngridBergman	actedIn	Casablanca(movie)
5	IngridBergman	bornOn	‘*1915-08-29*’
6	RobertoRossellini	type	writer
7	IngridBergman	type	actor
8	HumphreyBogart	type	actor
9	Casablanca(movie)	type	movie
10	Casablanca	type	city
11	writer	isA	person
12	actor	isA	person
13	person	isA	entity
14	IngridBergman	label	‘*Ingrid*’
15	IngridBergman	label	‘*Ingrid Bergman*’
16	RobertoRossellini	label	‘*Gastone Rossellini*’

(a)



(b)

Figure 2.1.: An example of a knowledge graph in (a) tabular and (b) graph form (only a fragment is shown).

Morocco's largest city. The name of an entity, which it can share with other entities, is also called a *surface form*, and is used by humans to refer to an entity in natural language utterances.

Entities are interesting in several applications. Entity-seeking queries form a significant part of Web search queries, with no less than 50% of Web queries in one study asking for entities (Pound et al., 2010b). Entity annotations in textual corpora have been used for search (Bast and Buchhold, 2013; Bast et al., 2014; Hoffart et al., 2014b), faceted search (Li et al., 2010), and analytics (Hoffart et al., 2014a), among other applications.

2.1.2. Classes

Classes, also referred to as types or categories, are named sets of entities. In the example KG of Figure 2.1 we have six distinct classes. The class `actor`, for instance, contains two entities: `IngridBergman` and `HumphreyBogart`. Exactly two predicates can be connected to a class in a KG. The first, `type` (often called `instanceOf`), connects entities to classes to which they belong. The second, `isA` (often called `subclassOf`), expresses hyponymy/hypernymy relations, and is used to organize classes of a KG into a directed acyclic graph shaped class hierarchy. The type `person` in our example KG has two subtypes (`actor` and `writer`), a single supertype (`entity`), and transitively contains a total of three entities.

Instances of these two predicates collectively form the ontology within the KG (facts 6-13 in Figure 2.1(a)) which we define as follows:

Definition 2.4 (Ontology)**.** Given a knowledge graph KG, its ontology is the set of facts whose subject or object is a class C. The predicates in such facts are restricted to either `type` or `isA`.

Pound et al. (2010b) reports that 12% of Web queries are "type queries", asking for instances of a specific class. Classes can often express multiple predicates in one shot, which makes them particularly valuable for information-seeking tasks. For example, the Yago KG (Suchanek et al., 2007) contains the classes `ItaliaNobelLaureates` and `RepublicanVicePresidentsOfTheUnitedStates`.

In logics terminology, classes can be seen as unary predicates, so we will occasionally use the notation `c(e)` to denote that entity e is a member of class c (i.e., `(e type c)`).

2.1.3. Predicates

Predicates in our setting are binary relations, whose first and second arguments are called the subject and object, respectively. Predicates are often referred to as relations or attributes. In the KG setting, relation is used to refer to predicates connecting pairs of semantic items. This is in contrast to an attribute, which is used to connect a semantic item to a literal (e.g., `GDP`, `hIndex`).

Predicates have a *type signature*, which gives the most general class of entities that can occupy each of the subject and object arguments of a predicate. We can write facts as triples in line with Definition 2.1 (e.g., `(IngridBergman bornIn Stockholm)`) or in logical form (e.g., `bornIn(IngridBergman,Stockholm)`).

The power of the triples model lies in the flexibility it offers for cases where the data is evolving. This model, compared to the classical relational model, does not require a schema to be defined upfront. Instead, new predicates can be defined as needed and populating them simply means appending new triples to the KG.

2.1.4. Concrete Knowledge Graphs

In this dissertation, we work with three concrete knowledge graphs, which we briefly present here.

Yago (Hoffart et al., 2013; Suchanek et al., 2007) combines two resources: Wikipedia and WordNet (Fellbaum, 1998). Yago taps into Wikipedia's infoboxes for facts linking entities to each other and to literals. The ontology in Yago is obtained from Wikipedia's assignment of entities to categories (for instances of the `type` predicate), and a mapping of these categories to noun synsets in Word-Net (for instances of the `isA` predicate). Yago adopts a rule-based extraction scheme, with each predicate having one or more hand-crafted extractors. Yago is distinguished by its high accuracy extractions and its comprehensive type hierarchy.

DBpedia (Auer et al., 2007), like Yago, is based on Wikipedia. DBpedia's extraction scheme is more liberal than that of Yago, compromising accuracy for recall. Whereas Yago contains about 100 predicates, DBpedia contains thousands. These are generally taken over as-is from the infoboxes in Wikipedia with minimal normalization or sanity checking. DBpedia includes Yago's ontology within it.

Freebase (Bollacker et al., 2008) is a collaboratively edited knowledge graph maintained by Google. Freebase forms an important part of Google's Knowledge Graph, which is actively used for providing concrete answers in response to Web queries, whenever possible. Although not directly extracted from Wikipedia, Freebase entities are linked to their corresponding Wikipedia entries when one is available.

2.1.5. Triple Pattern Queries

Triple pattern queries are used to query knowledge graphs. The triple pattern query language we adopt here is based on the SPARQL standard for querying RDF. In what follows we present this model formally. The basic building block here is a triple pattern. For this, we require variables, which stand for semantic items or literals that need to be returned by the query. Variables are always prefixed with a question mark (e.g., `?x`).

Definition 2.5 (Triple Pattern and its Answers)**.** Given a set of variables V distinct from semantic items and literals, a triple pattern q is a member of the set $V \cup E \cup C \times V \cup P \times V \cup S \cup L$. An answer (a) to a triple pattern over a KG is a total mapping of variables in q to $(S \cup L)$ such that the substitution of the variables with their mappings, denoted $a(q)$, results in a triple t in the KG.

An example of a triple pattern is `(?x spouse RobertoRossellini)`, which asks for the spouse of `RobertoRossellini`. From fact #2 in Figure 2.1(a) we can obtain the answer $a = \{(\texttt{?x}, \texttt{IngridBergman})\}$. We say that fact #2 matches the triple pattern.

Definition 2.6 (Triple Pattern Query)**.** A triple pattern query, or simply a query is a set of triple patterns $Q = \{q_1, ..., q_n\}$ and a projection set $P(Q)$ of variables. We require the join graph of Q, where q_i's are vertices and an edge exists between every pair of vertices sharing a common variable, to be a connected graph (to avoid computing Cartesian products). $P(Q)$ is a (usually proper) subset of the variables in Q, defining the output structure, typically tuples of entities.

We also refer to the triple pattern set Q as a query when the projection set is not relevant for the discussion. An example of a triple pattern query is
`SELECT ?x WHERE { ?x actedIn Casablanca(movie) . ?x spouse ?y }` ,
which is composed of two triple patterns and a single projection variable, $P(Q) = \{\texttt{?x}\}$.

The definition of an answer to a query is a natural extension of an answer to a single triple pattern.

Definition 2.7 (Query Answer)**.** For query $Q = \{q_1, ..., q_n\}$ an answer (a) over KG is a total mapping of the variables in Q to $(S \cup L)$ such that $\forall i \in [1..n] : a(q_i) \in KG$. The restriction of a query answer to variables in $P(Q)$ is called the projected answer, denoted a_P.

For the above query and the KG of Figure 2.1(a), the query has a single answer: $a = \{(\texttt{?x}, \texttt{IngridBergman}), (\texttt{?y}, \texttt{RobertoRossellini})\}$, from which we can obtain the projected answer $a_P = \{(\texttt{?x}, \texttt{IngridBergman})\}$, where a was restricted to the projection variable `?x`.

While we define an answer as a mapping of (projection) variables to semantic items and literals, we can also view it as a tuple of semantic items and literals, in a manner consistent with relational algebra. When presenting the answer as a tuple, rather than a mapping, we take the convention that the elements of the tuple are ordered according to the lexicographical order of the variables they originate from. For the above example, we can present the answer as $a = (\texttt{IngridBergman}, \texttt{RobertoRossellini})$, and the projected answer $a_P = (\texttt{IngridBergman})$.

A query can match multiple subgraphs in the KG, each one producing a distinct answer. The set of answers found for a query in a KG is the query result:

Definition 2.8 (Query Result)**.** The result of a query $Q = \{q_1, ..., q_n\}$ over a KG is a set of answers $A = \{a_1, a_2, ...\}$ such that $\forall a_i \in A : a_i$ is an answer to Q over the KG. A query result is also called an answer set.

An interesting class of queries is relationship queries:

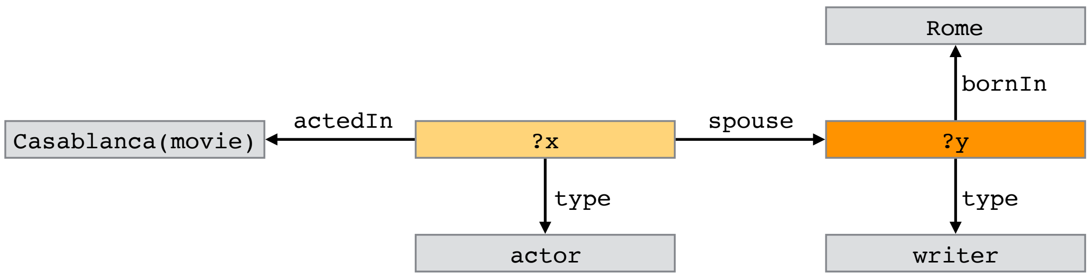
Definition 2.9 (Relationship Query)**.** Let $Vars(Q)$ be the set of variable in Q. A query Q is called a relationship query if $|Vars(Q)| \geq 2$.

Relationship queries allow us to express information needs where an answer contains multiple semantic items (i.e., $|P(Q)| \geq 2$). Even when we want (projected) answers composed of individual semantic items, relationship queries allow us to formulate queries with multiple ungrounded semantic items. Figure 2.2 shows an example of a relationship query which asks for the *"actress who played in Casablanca and was married to a writer born in Rome"*. Relationship queries stand in contrast to traditional entity search queries which are conceptually restricted to a single variable both in the body of the query and its projection set (i.e., $|P(Q)| = 1$). If we were to answer the query of Figure 2.2 using traditional entity search, we would need to first manually find the set of writers born in Rome, and then, for each such writer, ask for actresses who played in Casablanca and were married to the specific writer.

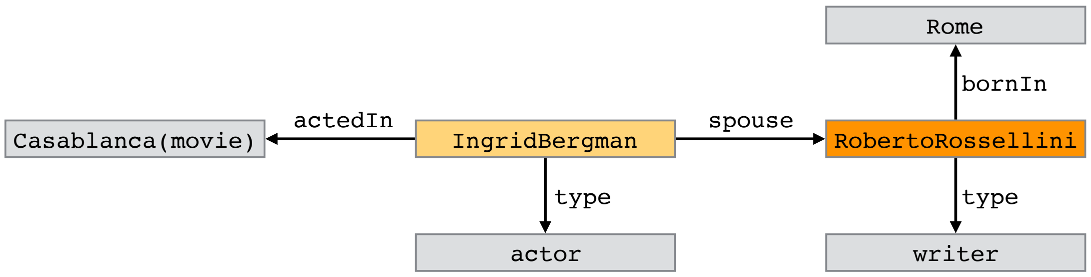
In Section 2.1.3 we discussed the flexibility offered by the RDF model in modeling emerging data. Modeling knowledge using the subject-predicate-object triples model results in the typical triple pattern query being composed of multiple *stars*, each centered around a variable, with the stars connected to each other by *chains.* The query of Figure 2.2 is composed of two stars and a chain connecting them.

```
SELECT ?x WHERE {
 ?x type actor   .  ?x actedIn Casablanca(movie) .
 ?x spouse ?y    .
 ?y type writer .  ?y bornIn Rome
}
```

(a)



(b)



(c)

Figure 2.2.: A triple pattern query corresponding to the question *"Which actress played in Casablanca and was married to a writer born in Rome?"* in (a) and its graphical representation (b) composed of two stars connected by a chain. The query is matched by the subgraph of the KG shown in (c).

2.2. Information Extraction

In the previous section we presented knowledge graphs. An important question is how can we obtain such knowledge graphs in the first place? This is the topic of this section. Most of human knowledge is available in textual form, be it on the Web, in books, newspapers, or manuals, mostly as free text, possibly with some tags to provide lightweight structure (i.e., semistructured data). The goal of information extraction is to extract semantic information from text and put it in a structured machine-readable format – a knowledge graph in our setting (Jurafsky and Martin, 2009).

Information extraction is a very broad area whose ultimate goal is to distill human knowledge into a format that machines can consume and process to facilitate upstream tasks such as language understanding, question answering, and analytics.

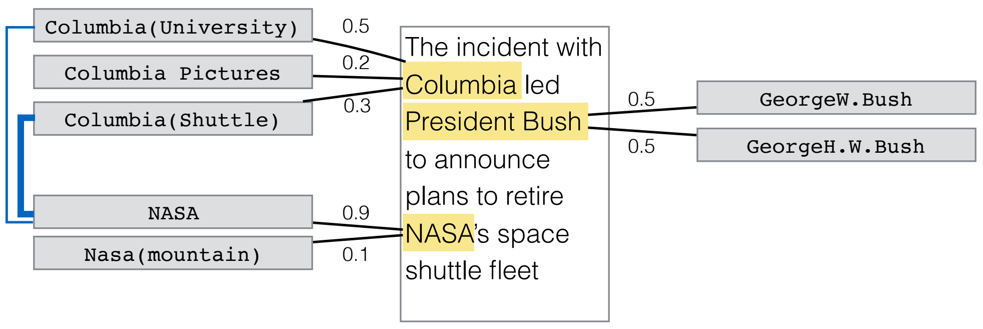
2.2.1. Named Entity Recognition and Disambiguation

The purpose of a KG as introduced in Section 2.1 is to describe entities. Named entity recognition (NER) is one of the basic information extraction tasks that serves as the building block for extracting more elaborate information to populate a KG. The task involves finding mentions of named entities in text and classifying them into one of several types. NER systems usually classify entity mentions into one of a handful of coarse types such as *person*, *organization*, and *location*.

Most general-purpose NER systems are trained on data created for shared tasks. The two most prominent are those from the Message Understanding Conference (MUC) (Grishman and Sundheim, 1996) and the Computational Natural Language Learning Conference (CoNLL) (Sang and Meulder, 2003). The data provided by both MUC and CoNLL is based on newswire texts, which use clear well-formed language. The resulting systems, such as the Stanford NER (Finkel et al., 2005), produce very good results on test data with similar characteristics. This system views the problem as one of sequence labeling, deploying conditional random fields to tackle it. However, models trained from the above data do not generalize well when it comes to texts from different domains, or informal texts such as forum posts, tweets, and Web search queries, leading to more recent work tackling the NER task in such texts (Downey et al., 2007; Guo et al., 2009; Ritter et al., 2011). Nadeau and Sekine (2007) gives a comprehensive overview of the NER task.

Complementary to the NER task is that of named entity disambiguation (NED), also called entity linking. The task here is to map ambiguous named entity mentions in a text to a repository of canonical entities. The mentions to be mapped are usually detected using an NER system.

In Section 2.1.1 we listed several applications where entities play a prominent role. These include entity search (Balog et al., 2011), entity-relationship search (Li et al., 2012), search in entity annotated documents (Bast and Buchhold, 2013; Bast et al.,



Entity	Context
NASA	space, mission, shuttle, astronaut,…
Columbia(shuttle)	space, crash, 2003, mission, nasa,…
Nasa(mountain)	sweden, cottage, silver, mining,…
…	…

Figure 2.3.: Named entity disambiguation example. Each highlighted entity mention maps to a KG entity with a prior. The table at the bottom shows the context signature for each entity. Coherence edges shown in blue for two entity pairs only.

2014; Hoffart et al., 2014b), faceted search (Li et al., 2010), and analytics (Hoffart et al., 2014a). Recognizing the importance of entities, there has been a resurgence in interest in named entity recognition and disambiguation (NERD), with most recent works focusing on Wikipedia or KGs connected to it (e.g., Freebase, Yago) as their reference entity repositories. These works rely on a mixture of priors for a surface form standing for a specific entity, the mutual coherence between candidate entities, and the similarity between a candidate entity's context signature and the disambiguation context (Hoffart et al., 2011). The disambiguation of entities in these works is performed in a joint manner, that is, the decision of which entity each mention in a text maps to is done at once for all entities. Joint disambiguation is crucial as the disambiguation of entity mentions in a text can be mutually informative. Figure 2.3 shows an example disambiguation instance using this framework. Hoffart (2015) gives a more detailed overview of approaches to NED and discusses extensions to the basic NED task such as cross-document coreference resolution and out-of-KG entity detection. Weissenborn et al. (2015) extends joint entity disambiguation to joint disambiguation of named entities and common nouns.

Realizing the value of corpora annotated with entities, multiple organizations have released corpora with annotations of disambiguated entities. Google released Freebase annotations of the ClueWeb'09 and ClueWeb'12 Corpora (FACC1),[1] the TREC KBA stream corpus (FAKBA1), and the TREC Million Query track and Web track queries.[2] Yahoo! released a sample of its query log annotated with Wikipedia entities.[3] We use some of these corpora in our work.

2.2.2. Fact Extraction

With documents annotated with named entities, the next logical step is to extract relations that hold between these entities (i.e., facts) or relations connecting them to literals and classes. This task is called fact extraction, but it is common to refer to this specific task as information extraction as well. The result of this process is a knowledge graph as described in Section 2.1. There are different approaches to tackling fact extraction, based on what sources are available for fact extraction and whether the set of relations is known a priori. Consequently, fact extraction systems are typically classified according to the amount of supervision they require, as we detail below.

Supervised Information Extraction

The earliest systems for fact extraction worked by matching hand-crafted lexico-syntactic patterns over textual corpora. Prominent among these are Hearst patterns (Hearst, 1992) for extracting hyponym/hypernym pairs (i.e., instances of `isA` and `type` relations in a KG setting). Many large-scale knowledge graphs are the result of a similar process that relies on hand-crafted patterns. In Section 2.1.4 we presented Yago, DBpedia, and Freebase, which rely (at least in part) on manually specified rules for extracting facts from Wikipedia, primarily focusing on the semistructured infoboxes. Manually specifying extraction patterns does not scale when the number of relations for which we want to perform extraction increases. To deal with this issue, focus shifted to building extraction systems that need less supervision.

Semisupervised Information Extraction

In bootrapping, supervision is achieved by providing the extraction system with a small number of example instances of the relations for which we desire to extract facts. DIPRE (Brin, 1998) and Snowball (Agichtein and Gravano, 2000) are two prominent examples of such systems. Mintz et al. (2009) extends this approach to distant supervision where bootstrapping is done from a large knowledge graph of facts that serves as the source of supervision. In the process of bootstrapping, distant supervision assumes that a sentence containing a pair of entities that appear in a fact in the reference knowledge graph is expressing that fact. A classifier is trained on these spotted instances

[1]`http://lemurproject.org/clueweb09/FACC1/`,
`http://lemurproject.org/clueweb12/FACC1/`
[2]`http://lemurproject.org/clueweb09/TREC%20Freebase%20Queries,%20v1.1.zip`
[3]`http://webscope.sandbox.yahoo.com/catalog.php?datatype=l&did=66`

to recognize each occurrence of a relation. Using large knowledge graphs and large text copora allows for very expressive features. The classifier also requires negative examples, these are usually generated from entity pairs that do not appear together in the reference knowledge graph.

PROSPERA (Nakashole et al., 2011) is an extension of distant supervision that can automatically generalize extraction patterns and uses constraint-based reasoning for controlling the quality of patterns and facts. Pattern generalization, achieved through frequent itemset mining, allows for higher recall. By using MaxSat reasoning, PROSPERA can control the quality of the resulting extractions by checking for their mutual consistency. PROSPERA builds on SOFIE (Suchanek et al., 2009) but is more scalable and provides higher recall.

Open Information Extraction

In all the above approaches, the set of relations for which we extract facts is defined upfront, along with a set of examples for each relation. Defining what relations are interesting in a specific domain and coming up with examples for each requires effort by humans. Once the relations in a domain need to be extended, or the domain is changed altogether, the existing data will not be very useful, and manual effort has to be invested again. This does not scale well when the data is continuously evolving, and with it the set of interesting relations.

Open Information Extraction (OpenIE) was proposed as a solution to the problems above. The goal of OpenIE is to perform fact extraction without the need for any domain-specific modeling in a scalable manner on large Web-scale corpora (Etzioni et al., 2004). The initial work on OpenIE, KnowItAll, supported open-ended information extraction by providing a framework where extraction rules can be manually defined in a domain-independent manner. These rules would trigger Web search engine queries that allow KnowItAll to find documents to which these rules can apply. The Web search engine is also queried to automatically asses the quality of an extraction by aggregating evidence for that extraction. The successor framework, TextRunner, extracted facts by finding spans of text in a large corpus connecting two noun phrases (corresponding to the arguments of a fact) and using a classifier to score such extractions (Banko et al., 2007; Etzioni et al., 2008; Yates et al., 2007). WOE^{parse} (Wu and Weld, 2010) extends this work by using dependency parsing to connect the argument noun phrases along with a similar scoring scheme.

ReVerb (Fader et al., 2011), one of the most widely used OpenIE systems, relies on a small set of patterns over part-of-speech tags to detect relations, and then looks to the left and right of the relation for noun phrases corresponding to its arguments. Subsequently, extractions are tested against a lexical constraint where relations that don't occur with a sufficient number of distinct arguments are eliminated. The resulting extractions are finally scored using a confidence classifier. OLLIE (Mausam et al., 2012) bootstraps ReVerb by using its extractions as seeds for finding dependency parse

Scheme	**E**	**U**	**NS**
Keywords	✗	✓	✓
Questions	✓	✓	✓
Structured queries	✓	✗	✗

Table 2.1.: Comparison of keywords, natural language questions and structured querying along the dimensions of expressiveness (E), usability (U), and a user's ability to formulate queries without schema knowledge (NS).

patterns expressing facts.

In Chapter 5 we present an approach to OpenIE that focuses on extracting facts centered around noun phrases, which other systems have neglected, and which can extract the arguments of a relation in a semantically meaningful and consistent manner.

2.3. Querying for Knowledge

We now discuss how we can query for knowledge. The starting point for a user is an *information need* for which she would like to obtain answers (Büttcher et al., 2010). The user generates a query (or possibly a sequence of queries) from this information need. For a given information need, the query will differ based on the system to which it will be issued.

In this section we look at three querying schemes relevant to this dissertation: keyword querying, structured querying, and question answering. The schemes differ along three main dimensions: expressiveness, usability in terms of the technical knowledge needed to formulate a query, and how much familiarity is required with the data to pose a query. Table 2.1 gives a comparison of the three query schemes along the three dimensions, we elaborate on this below.

2.3.1. Keyword Querying

Most people are familiar with *keyword querying* through the use of Web search engines. A keyword query is generally considered a bag of *terms*, that is, the order in which they are provided by the user is arbitrary.[4] In the typical setting where keywords are used, a user will issue a keyword query to an engine and expect a ranked list of documents, each possibly corresponding to an entity as we discuss below. Other settings have also been explored where keyword queries are used to retrieve tuples from databases or trees from XML documents.

[4]This assumes that a term refers to an atomic concept, see Metzler and Croft (2005)

Document retrieval has been a major focus of information retrieval systems since the earliest days of the field (Croft et al., 2009). Here, most of the focus has been on ranking textual documents (Web pages, emails, books, scholarly publications) in response to keyword queries. While the roots of the field go as far back as the mid 1960s, the field saw rapid progress since the mid-1990s due to the popularity of Web search engines and driven in large part by the various tracks of the Text REtrieval Conference (TREC), which was started in 1992. Among the successful approaches for ranked document retrieval is language modeling (Ponte and Croft, 1998; Zhai, 2008) and query expansion (Xu and Croft, 1996) which inspire some of the work in Chapters 3 and 4.

Motivated by the familiarity of users with keyword querying, primarily through their interaction with Web search engines, researches have worked on adapting this form of querying for problems other than document retrieval. Most relevant for our setting of knowledge querying is work on entity search and keyword search over knowledge graphs (and databases in general). In *entity search*, the goal is to return an entity or a set of entities in response to a keyword query. State-of-the-art approaches map the problem of entity search to the classical document retrieval setting by constructing per-entity documents and retrieving these documents in response to an entity query. Entity documents are usually fielded, with fields capturing the various facets of an entity. Extensions of this approach allow for fielded queries, minimally with a field for specifying the types to which an entity belongs, and another for a description (with relations or entities) (Balog et al., 2011).

Another class of systems answers keyword queries over databases typically for the purpose exploratory search, with some systems specifically focusing on knowledge graphs. Such systems are a fundamental component for verticals (e.g., Amazon.com), which are predominantly driven by a database containing the topics they offer and their information. Tran et al. (2009) generates the top-k triple pattern queries that best correspond to a keyword query over a specific knowledge graph. Each structured query is derived from a subgraph of the knowledge graph that can connect elements in the knowledge graph mentioned in the keyword query. Elbassuoni and Blanco (2011) adopts an IR-inspired approach, where a set of subgraphs matching a keyword query are retrieved and subsequently ranked based on a statistical language model. In contrast to the first approach, the second one returns answer subgraphs to the user rather than structured queries. Yu et al. (2009) gives a comprehensive overview of the area of keyword search over relational, graph and XML databases.

Keywords inherently have very limited expressiveness, which means that they cannot be used to fully exploit the structured data available to us in knowledge graphs. For example, the information need expressed by the question shown in Figure 2.2: *"Which actress played in Casablanca and was married to a writer born in Rome?"*, is hard to capture in its full semantics in a keyword query such as '*actress played casablanca married writer born in rome*'. Moreover, even if the user were to provide a keyword-querying engine with the full question, she would not benefit in terms of result quality as the engine is not designed to exploit the expressiveness of natural language, making the effort not worthwhile.

An interesting extension of such systems is QUICK (Pound et al., 2010a), which allows *structured keyword queries*. Such queries are meant to be a compromise between keyword queries and structured queries for describing an entity. Using this query language, our example might be formulated as:

"actress, played in(casablanca), married to(writer, born in(rome))"

The various keywords in the structured keyword query are mapped to candidate semantic items in the underlying knowledge graph, and the goal becomes finding the set of top-k scoring subgraphs, each of which correponds to a distinct triple pattern query. A subgraph is scored using a combination of syntactic similarity between its items and keywords in the query, and semantic similarity between these items. The top-k matching subgraphs are found efficiently using an top-k enumeration scheme based on rank-joins (Ilyas et al., 2003). However, even such queries can be hard for the average user to formulate.

2.3.2. Structured Querying

Structured querying over knowledge graphs is on the other extreme from keyword querying. In Section 2.1.5 we presented triple pattern queries for querying knowledge graphs. Here, the information need is expressed unambiguously. Moreover, very complex queries can be expressed, particularly relationship queries which in the classical IR setting would require combining cues from multiple documents to obtain an answer. For instance, our running example can be answered using the following query, which we have seen in Figure 2.2(a):

```
SELECT ?x WHERE {
  ?x type actor  .  ?x actedIn Casablanca(movie) .
  ?x spouse ?y   .
  ?y type writer .  ?y bornIn Rome
}
```

Using structured querying suffers from three major problems. First, formulating structured queries in a formal language such as SPARQL is not a trivial task for the average person, who will have no experience in formal languages. Second, even a user experienced at formulating structured queries might struggle to find the correct knowledge graph terms and structure needed to formulate her query. This is due to the large size of a knowledge graph and its underlying vocabulary, and the lack of a schema. Finally, because knowledge graphs are continuously evolving without a fixed schema, the information need underlying a structured query might have multiple answers in the knowledge graph. However, each answer might require a slightly different query to obtain.

We have already seen how keyword querying was proposed as a way to tackle the issues above and the limitations of keyword querying, particularly when it comes to expressiveness. In this dissertation we tackle the problems above in two ways. In the first, we allow for full fledged natural questions as an alternative for keyword querying. Natural language allows users to express very complex queries without any technical know-how (Chapter 3). In the second, we assume that a structured keyword query is already at hand – either formulated by a user, or automatically generated by a question answering or keyword querying system. We present a triple pattern querying framework that takes a triple pattern query and can automatically adjust it to bridge any gap between the query and the KG. An important component of the second approach is that it allows for processing queries on an extended version of the knowledge graph obtained through information extraction (Chapter 4).

2.3.3. Natural Language Questions

Natural language questions as a query language present an ideal compromise between keyword and structured querying. Questions can be used to express complex information needs that cannot be expressed as keywords without a significant loss in structure and semantics. Natural language questions are the most intuitive way of formulating an information need, and any human can formulate questions to express their information needs. Moreover, with voice seeing wider adoption as a modality for human-computer interaction, support for questions as a query language will become a must.

The idea of machines answering human questions goes back to the earliest days of computer science. Turing's scheme for testing whether machines exhibit intelligence, the Truing test, is based on a machine answering questions in a manner indistinguishable from humans (Turing, 1950). While achieving this goal seems unlikely in the near future, the problem has been tackled from different angles in the past 65 years.

Starting from the mid 1990s, and for a period of about 10 years, question answering saw a revival, primarily in the information retrieval community. This revival was mainly driven by the Web and the availability of large query logs from major search engines at the time (e.g., MSN search and Ask Jeeves). The main driver of QA research during this period was NIST's TREC-QA track, which made available questions donated by Web search engines and corpora in which answers to these questions were to be located.

IBM's Watson system is one of the milestones in the history of natural language question answering (Ferrucci, 2012; Ferrucci et al., 2010). Watson successfully competed against humans at the Jeopardy! television game show. Work on Watson focused on three main topics: question answering (finding answers), strategy (deciding when to answer and how much to bet on a question), and efficiency (answering fast). On the question answering front, Watson relied mainly on textual corpora, with limited use of knowledge graphs for answering question fragments that can be mapped to a KG with high confidence (Chu-Carroll et al., 2012) and type-checking answer candidates for pruning of spurious answers (Kalyanpur et al., 2011).

In recent years, there has been a resurgence of interest in question answering, this time primarily in the natural language processing and database communities with the goal of question answering over databases, with particular interest in knowledge graphs. In contrast to early work described above, the recent approaches operate over much larger data, and rely heavily on statistical methods rather on purely logical ones. Liang and Potts (2015) gives an overview of the field. We also give a more comprehensive overview in Chapter 3.

In terms of usability, questions are a more accessible medium for expressing complex information needs. Most users lack the technical know-how needed to formulate a structured query. Additionally, as discussed in Section 2.1, knowledge graphs are usually large, continually evolving, and lack a schema. This means that finding the right query can be a challenging task, possibly requiring multiple iterations of query reformulation to find the correct query. Natural language questions can relieve the user from this burden by moving it to the machine, which will be responsible for understanding the question with respect to the underlying data. We present our framework for natural language question answering for knowledge graphs in Chapter 3.

3. DEANNA: Natural Language Question Answering over Knowledge Graphs

3.1. Introduction

3.1.1. Motivation

The availability of large knowledge graphs presents a challenge and an opportunity that are, in some sense, duals of each other. The challenge lies in how to make this abundance of knowledge easily accessible to humans. Only a fraction of potential consumers of this knowledge will be versed in formulating triple pattern queries that express their information needs. Even for such people, the sheer size of the data and the lack of a schema means that the task of formulating a triple pattern query can be a challenging task, requiring multiple rounds of tedious query reformulation.

A solution to the above problem is to allow users to query knowledge graphs by posing natural language questions. These questions are subsequently interpreted with respect to the specific KG at hand by mapping them to a triple pattern query, which can be issued to the KG, returning the desired answers to the user.

This new setting of large knowledge graphs presents an opportunity to tackle the question answering problem using new approaches. By working against a knowledge graph, crisp entities can be returned as answers. By exploiting the structure provided by the KG, one can also answer complex questions that require multiple joins, corresponding to paths in the KG. In these cases, a KG can be used to return answers that are proper tuples of entities, rather than singletons. Moreover, users will be able to ask for how an answer was derived by looking at the query that produced it.

3.1.2. Problem Statement

The high-level problem we deal with in this chapter is that of question answering over a knowledge graph. Given a knowledge graph KG and a question u (for utterance), our task is to return a result $A = \{a_1, a_2, ...\}$, where each answer in the result is an n-tuple of entities, $a_i = (e_1, e_2, ..., e_n)$. Figure 3.1 shows an example of a question and

$u =$ *"Who played in Casablanca and was married to a writer born in Rome?"*
$A = \{(\texttt{IngridBergman}, \texttt{RobertoRossellini})\}$

Figure 3.1.: An example question and the corresponding result that contains a single answer composed of an entity pair.

its result containing a single answer composed of a pair of entities. We elaborate on the problem description in what follows.

Our task is called question answering over knowledge graphs. We operate with a very broad definition of a question. For us, a question is a natural language utterance, hence the symbol u, that expresses an information need. These can be questions formed with interrogative words (e.g., *"Who was born in New York City?"*), yes/no questions (e.g., *"Was Obama born in Hawaii?"*) and utterances intended as questions, but do not look as such (e.g., *"Presidents born in Hawaii."* which is simply a shorter form for asking *"Which presidents were born in Hawaii?"*).

An answer in our setting is an n-tuple of entities from the KG. In Figure 3.1 we give an example of such an answer. This is in contrast to the classical IR setting, including the entity-retrieval task (Pound et al., 2010b), where an answer is an individual document (possibly corresponding to a single entity).

While an answer is a tuple of entities from the KG, our definition of the problem does not force us to be able to explain an answer by the facts within the KG. We will assume that the decision about the relevance of an answer in response to a question is done independent of the KG. In this manner, systems working on this task have the freedom to use whatever means and resources necessary to focus on the end task of answering the question at hand, and are not restricted by the unavoidable incompleteness in any KG.

Related to the above point, we focus on the end-to-end task of question answering over a KG. Therefore, our end goal is not to generate the correct triple pattern query in response to a question. In other words, our task is that of question answering rather than semantic parsing. Evaluating the correctness of a query, rather than an answer, is not a well-defined task. A seemingly correct query might miss some or all correct answers due to KG incompleteness. On the other hand, there might be multiple correct queries that capture a question, which produce overlapping or complementary sets of answers. Notwithstanding this discussion, an important component of our QA pipeline is the translation of a question into a triple pattern query.

3.1.3. Contributions & Overview

We present DEANNA, a two-stage end-to-end framework for natural language question answering over knowledge graphs. In the first stage, DEANNA maps a user's question to a triple pattern query. In the second stage, it extends and relaxes this query with textual conditions inspired by traditional IR to compensate for any shortcomings in the first part or incompleteness in the KG. Figure 3.2 shows the full pipeline that takes us from a question to its answers, we elaborate below.

In the first stage, DEANNA generates a triple pattern query from the question. For the example question in Figure 3.1, DEANNA would generate the query:

```
SELECT ?x WHERE {
   ?x type person . ?x actedIn Casablanca(movie) .
   ?x spouse ?y . ?y type writer .
   ?y bornIn Rome   }
```

Generating a triple pattern query from a question is a two-stage process. First, DEANNA constructs a disambiguation graph from the question, which essentially captures all possible interpretations of the question with respect to the KG. To obtain the correct interpretation among these, multiple ambiguities have to be resolved: the segmentation of a question into phrases that can be individually mapped to semantic items in the KG, the actual mapping, and connecting these semantic items together in order to construct triple patterns. As the solution to each of the three ambiguities above can inform the resolution of the other ambiguities, we resort to a joint model of disambiguation. Our model utilizes a judiciously designed Integer Linear Program (ILP) for joint disambiguation. The ILP, composed of an objective function and a set of constraints, captures the intuition that we are looking for the most likely interpretation of the question (the objective) that also makes sense (the constraints). The constraints make use of the semantic knowledge in the knowledge graph, requiring, for instance, that the arguments of a predicate in a triple pattern match its type signature. The result of the first stage is a triple pattern query.

In the second stage, the triple pattern query is extended with text conditions. These conditions serve two purposes. The first is to account for any parts of the question that could not contribute to the triple pattern query, either due to issues with the interpretation framework or the underlying KG. The second purpose of this extension is to allow for the relaxation of the triple pattern query whenever it is unable to return satisfactory results. We deem a result to be satisfactory when it is non-empty. Triple pattern queries extended with text conditions are evaluated over a combination of the KG and a textual corpus of entity descriptions. Query relaxation is interleaved with query evaluation, and continues until a query returns satisfactory answers to the user. For our running example, we can imagine a KG where the `bornIn` predicate is sparsely populated, and where the triple pattern query above results in an empty answer set. We would relax the culprit query condition `?y bornIn Rome` to into a textual condition associated with `?y`'s type constraint: `?y type writer` *'born in rome'*.

DEANNA answers questions over knowledge graphs composed of entities, predicates, and classes. In this chapter we work over a combination of DBpedia and Yago, two knowledge graphs constructed from Wikipedia as described in Section 2.1.4. The connection to Wikipedia is crucial, as it allows us to both create the dictionaries and compute the statistics needed by DEANNA. Yago serves as our ontology, while DBpedia provides instance data.

In the following, we first survey related work in Section 3.2. We then describe disambiguation graph construction in Section 3.3. Next, we present our ILP-based joint disambiguation framework for question interpretation in Section 3.4, and how the result is translated to a triple pattern query in Section 3.5. In Section 3.6 we describe our query extension and relaxation framework, and describe answer scoring, which is required to cope with the potentially large number of results returned from relaxation. Finally, we present an experimental evaluation of DEANNA in Section 3.7.

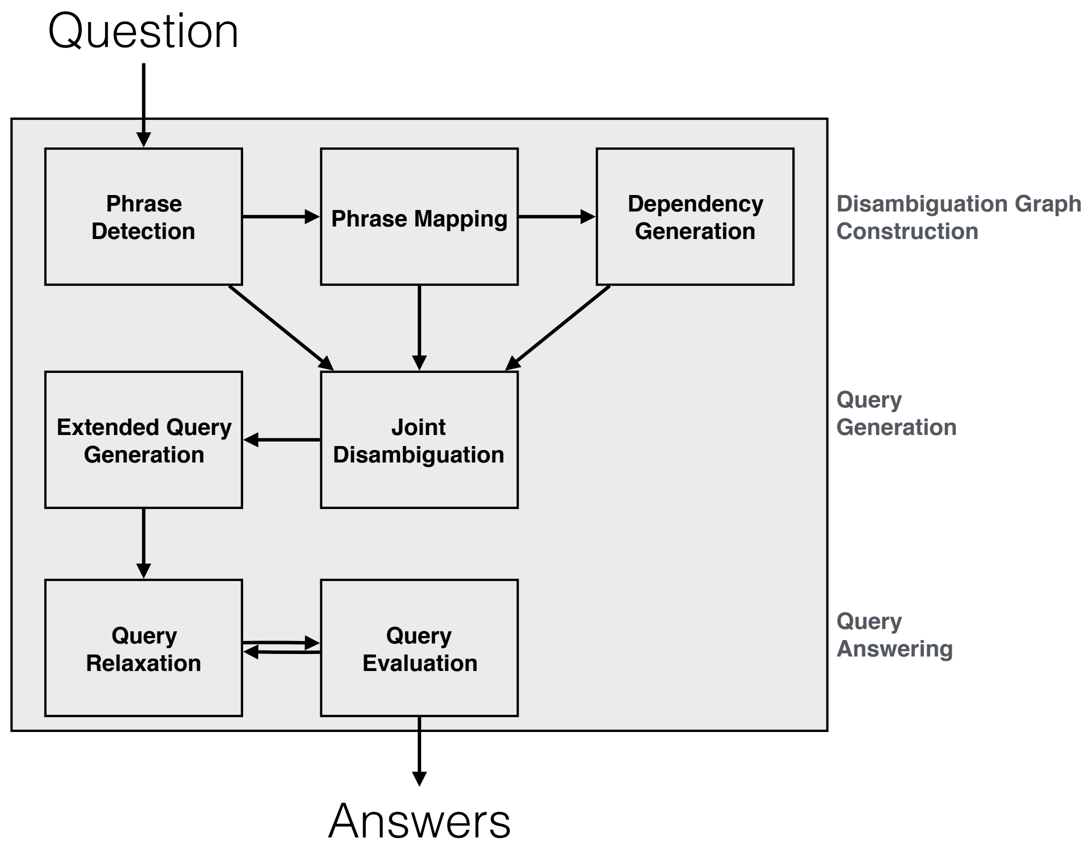


Figure 3.2.: Architecture of DEANNA.

3.2. Related Work

Question answering (QA) has been one of the holy grails of computer science research since its early days. It has been tackled by the natural language processing community, the IR community, and, to varying degrees, by the database and logics communities. In what follows we present an overview of the field.

3.2.1. IR-based Question Answering

The 1990s and early to mid 2000s saw a plethora of work on IR-based approaches to questions answering (IR-QA). Progress in this area was mainly driven by the TREC question answering track, which ran annually between 1999 and 2007 (Dang et al., 2007). Here, the focus was mainly on list, factoid, and definition questions posed over text corpora. Over the years, both newswire and blog corpora were used. An answer in this setting was a pair with an answer string and the id of the document where it was found (or 'NIL' when none can be found). Answer strings were usually constrained to a predefined number of bytes. Crucially, if an answer could be found for a question, then it was expected to be fully contained within a single textual document. DEANNA, by supporting complex join questions, is able to answer questions that conceptually require evidence from multiple documents by working with structured data.

Most IR-QA systems preprocess the document corpus at hand using the typical IR preprocessing pipeline primarily composed of tokenization, normalization and indexing (Manning et al., 2008). Some systems perform further preprocessing and analysis. For example, the START system (Katz, 1997) uses the parse trees of sentences in a corpus to create ternary subject-predicate-object expressions. These expressions are indexed for use during question answering. Other systems, however, perform such sophisticated analysis at question answering time over the subset of the corpus thought to contain the answer, as we detail below.

The majority of IR-QA systems divide the question answering task into the following subtasks: (i) question analysis, (ii) query formulation, (iii) document retrieval, (iv) passage retrieval, and (v) answer generation.

The most important step in question analysis is determining the expected answer type. For this, IR-QA systems maintain a type taxonomy. For example, Webclopedia (Hovy et al., 2002) uses a taxonomy 140 types manually compiled from analyzing a large question repository. Pasca and Harabagiu (2001) uses a type taxonomy with 8707 so-called concepts, which include verbs and adjectives, that are connected to WordNet noun synsets that correspond to answer types. To determine the answer type of a question, a combination of syntactic parse analysis, template matching, and machine learning are deployed (Li and Roth, 2002; Ravichandran and Hovy, 2002). Such type taxonomies are also important in our setting of question answering over knowledge graphs, and are usually part of the knowledge graph. Knowledge graphs such as

DBpedia and Yago, both of which we use in our work, offer hundreds of thousand of types, some of which are taken over from WordNet, while others are very fine grained (e.g., `RepublicanUnitedStatesVicePresidents`) and come from Wikipedia.

Query formulation takes the question and formulates a query that can be issued to the IR system, which serves as the interface to the document corpus. The primary consideration here is to create a query that is likely to find relevant matches in the underlying corpus. This is achieved by a combination of question reformulation and query expansion. In question reformulation, the goal is to take the question, a interrogative sentence, and recast it into a declarative sentence or clause that can better match the underlying corpus. Reformulation generally requires linguistic analysis of the question, which includes, among others, identifying the predicate-argument structure of the question and named entity mentions in it. Query expansion (Xu and Croft, 1996; Voorhees, 1994) is a typical query preprocessing step in IR, where the goal is to bridge any terminological gap between the query and the underlying corpus. Here, morphological, lexical, and semantic expansions are used, often relying on WordNet (Hovy et al., 2000; Pasca and Harabagiu, 2001).

This query is used to retrieve an answer from the underlying corpus in several steps. First, document retrieval is performed, where the goal is to retrieve a relatively small set of documents where the answer is likely be found. By restricting ourselves to a subset of the corpus using document retrieval, we can afford to apply more detailed, and therefore expensive, analysis of these documents. Such analysis includes parsing, phrase chunking and named entity recognition and typing (Kwok et al., 2001; Srihari and Li, 1999). These annotations are used to perform passage retrieval (Salton et al., 1993; Tellex et al., 2003), where passages in the documents that are expected to bear the desired answer are retrieved.

Subsequently, these passages are scored using a host of features, including their overlap with the question and the match of the named entities they contain with the expected answer type. In the final step, answers are extracted from the top-scoring passages. Here, both templates (Hovy et al., 2002; Lin, 2007) and n-gram mining and tiling are used (Brill et al., 2002). In the latter, n-grams from high-scoring passages are collected and scored, in part based on how well they match the expected answer type of the question. Overlapping n-grams are coalesced, with their scores combined, and the highest scoring ones are emitted as answers.

Hirschman and Gaizauskas (2001) and Jurafsky and Martin (2009) give a more detailed overview of the field of IR-QA.

3.2.2. Question Answering over Databases & Semantic Parsing

The earliest work on question answering focused on answering questions from a database. For example, BASEBALL (Green et al., 1961) answered questions about baseball games and their statistics. LUNAR (Woods et al., 1972) answered questions about the Apollo 11 mission to the moon. These systems were rule-based and operated over what would nowadays be considered very small databases. More importantly, each system was designed to deal with a specialized domain, and could not be easily extended to operate on a different domain.

Providing natural language interfaces to databases (NLIDB) has been one driver of research on question answering (Popescu et al., 2003). The early systems discussed above were driven by this goal. This problem has been seeing a revival in the database community, with work on various forms of structured data including relational databases (Li and Jagadish, 2014) and XML (Li et al., 2007). These systems are driven by the desire to make structured data more accessible to those users not well versed in formal languages. The approaches they follow generally map predfined syntactic patterns to specific query patterns. These systems are interactive in that they rely on humans to help resolve ambiguities that arise along the way and allow humans to iteratively refine the resulting query.

The task of question answering over knowledge graphs, or databases in general, is directly related to the task of semantic parsing. In fact, question answering has been one of the main drivers of progress in the area of semantic parsing. The goal of a semantic parser is to translate a natural language utterance to some logical form. This can be lambda calculus, robot commands, or a database query as we do in this work. NLIDB systems like the ones above can be seen as semantic parsers, but many others were developed with an immediate focus on the semantic parsing task, mostly using machine learning and statistical approaches. Such systems vary along several dimensions, most notably what kind of supervision they require and what kind of logical form they produce.

One class of semantic parsers relies on learning from utterance-logical form pairs (Kwiatkowski et al., 2010; Zelle and Mooney, 1996; Zettlemoyer and Collins, 2007, 2009). Owing to the complexity of coming up with utterances annotated with logical forms for training, these early systems operated on fairly small databases. One reason is that such systems learn both how to derive a logical form and the necessary mappings of phrases to logical constants at once. Cai and Yates (2013) presents a system that allows working with larger databases by extending such systems with schema matching and lexicons learned independently.

One way of overcoming the problem above is to exploit utterance-answer pairs rather than utterance-logical form pairs for training, as the former are easier to obtain — in the semantic parsing setting what we call an answer is referred to as a denotation and is obtained by executing the logical form (Liang and Potts, 2015). This scheme is followed by several recent works including Berant et al. (2013), Berant and Liang (2014), Clarke et al. (2010), and Liang et al. (2011). In general, these works rely on a component that can generate logical forms for a given utterance-answer pair, and train a system that can later generate queries, which remain latent from the perspective of the end-user.

DEANNA and several other recent systems works including Adolphs et al. (2011), Cabrio et al. (2012), Unger and Cimiano (2011), Unger et al. (2012a), and Wang et al. (2015) rely on having a domain-independent grammar combined with a domain-specific lexicon for mapping phrases in the questions to entities and predicates in the KG. These systems greatly resemble some of the earliest approaches for QA over databases. However, a crucial difference is that the recent systems heavily harness statistics, backed by very large annotated corpora, to score candidate queries and decide on the one that best captures a question. These systems are valuable for flexibly deploying QA systems across domains, especially when there is a lack of training data as described above to learn a semantic parser.

Recently, approaches were proposed for question answering in what is called an open predicate setting. This is in contrast to the closed predicate setting which includes all the systems described above. Question answering in an open predicate setting is tied closely to the notion of open information extraction, where facts are extracted from text and each word or phrase has its own distinct meaning with the goal of achieving more coverage than closed predicate knowledge graphs. Within this setting, Fader et al. (2013, 2014) propose an approach for question answering over a combined database of curated and automatically extracted knowledge graphs. The power of the method proposed comes from the size and diversity of the underlying knowledge graph and a multi-stage paraphrasing scheme for both questions and structured queries. Krishnamurthy and Mitchell (2015) learns a probabilistic database that defines a probability distribution over answer entities for each textual predicate using a corpus of entity-linked text and probabilistic matrix factorization. At query time, answers are determined by inference where marginals are computed for each potential candidate entity. By essentially embedding entities and facts in a latent space, this approach does not easily lend itself to providing explanations for answers it returns. The questions answered by the above approaches are fairly simple.

In DEANNA we rely on combining structured facts with their textual context to allow for flexible question answering whenever the knowledge graph is incomplete or the triple pattern query fails to fully capture a question. QUETAL (Frank et al., 2007) has also looked at combining structured and unstructured sources for question answering, albeit in a manner different than what we do in this work, with the structured and textual sources serving as complementary sources of answers.

3.2.3. IBM Watson

IBM's Watson question answering system is the most successful example of a deployment of a question answering system. Watson was designed to play the Jeopardy! game show, and in 2011 won the IBM Jeopardy! challenge (Ferrucci, 2012). Watson is an example of a system that combines both IR and KGs to perform question answering.

Analyzing Jeopardy! questions (called clues), reveals that they tend to be structurally very simple in comparison to the questions DEANNA can cope with. The complexity mostly comes from the quiz nature of the questions where the answers are not necessarily entities that would be found in a typical knowledge graph. In fact, analysis reveals that only 2% of Jeopardy questions could be answered by exclusively utilizing structured sources (Ferrucci et al., 2010).

Watson starts out by analyzing a question, which includes deciding whether it is a factoid question or if it falls into on of a number of predefined question classes that can benefit from special handling (e.g., definition or etymology question). The special questions collectively constitute nearly 40% of Jeopardy questions (Lally et al., 2012).

As we do in DEANNA, Watson relies on parsing the question to capture its structure and transform it into a predicate-argument structure similar to our notion of phrase dependencies (McCord et al., 2012).

Watson utilizes unstructured resources for answer retrieval, putting great value on specific textual corpora such as Wikipedia, where page titles contain answers to the majority of Jeopardy! questions (Chu-Carroll et al., 2012).

Watson also uses knowledge graphs and other forms of structured data for answering. It relies on recognition grammars for recognizing occurrences of KG predicates in a question. Since these grammars are manually crafted, they are only created for the most frequent predicates (Chu-Carroll et al., 2012). In addition to obtaining answers directly, using structured data allows Watson to support certain types of reasoning such as temporal and spatial reasoning (Kalyanpur et al., 2012). Watson also extracts its own large-scale lexicalized relation source from text called Prismatic, which is essentially a knowledge graph with textual phrases rather than semantic items, i.e., an open predicate KG (Fan et al., 2012).

Watson relies on so-called type coercion, where a potential answer is scored by a specialized component based on how well it can be coerced into a type compatible with the one the question is asking for (Murdock et al., 2012). This is needed due to the wide variety of question classes in Jeopardy!. Some type coercion components used in Watson rely on knowledge graphs that DEANNA also utilizes, including Yago (backed by WordNet), and DBpedia. These can give a strong (positive or negative) signal for factoid questions where answers are entities.

In addition to the question answering component we outlined above, Watson also has a game strategy component that guides it in its decisions on when to attempt to answer, how much to wager, and which question category to choose and with what monetary value. This is based on a combination of learning from historical game data for human players, Watson's own history, and the analysis of the other players in the game. Interestingly, Watson can produce a score reflecting its confidence in its answers and uses this score to decide whether to attempt a clue (Tesauro et al., 2012). Watson owes its performance in the Jeopardy! challenge to both its question answering and game strategy components in equal measures. Watson is backed by an enormous amount of computational power that allows it to answer questions in interactive times (Epstein et al., 2012).

3.2.4. Keyword Querying over Databases

The topic of keyword search over various forms of structured data such as relational databases and graph data (including knowledge graphs) has been proposed as a means of providing easy access for users to databases (Agrawal et al., 2002; Bhalotia et al., 2002; He et al., 2007; Hristidis and Papakonstantinou, 2002; Yu et al., 2009). To answer such queries, parts of the query are matched with the attributes and contents of relational records. Foreign-key relationships are subsequently used to compute a connected result graph explaining how the various keywords relate to each other.

Pound et al. (2010b) introduces the task of ad-hoc object retrieval over data graphs (in contrast to the well-established ad-hoc document retrieval task over textual document corpora). Here, queries are formulated as a bag of keywords, and the result is a ranked list of entities. Queries are divided into entity (e.g. '*eiffel tower*'), type (e.g., '*tourist attractions*'), attribute (e.g., '*germany gdp*'), and relation queries. This work also shows that such queries collectively constitute more than 55% of the queries in a representative query log of a major search engine.

Joshi et al. (2014) presents an approach for segmenting short telegraphic queries (short ill-formed questions) and mapping segments onto entities, predicates, or types in a KG or considering them a contextual segment. The focus here is on obtaining answer entities to queries, and therefore multiple segmentations and corresponding mappings are considered when scoring an answer entity. Answer scoring is performed using an undirected graphical model where some potentials roughly correspond to DEANNA's notion of prior and coherence. Other potentials rely on statistics collected from an entity-annotated corpus and provide a form of relaxation to make up for incomplete knowledge graphs. This work is an extension of Sawant and Chakrabarti (2013), which mapped segments onto semantic types or considered them as contextual segments.

Pound et al. (2010a) presents an approach for querying knowledge graphs using structured keyword queries, which are an extension of traditional keyword queries. Such queries can be used to recursively describe a single entity using a mixture of entities, relations, and classes, and a combination of these (all expressed through keywords). These queries are automatically translated to SPARQL queries using a disambiguation approach similar in spirit to the one used by DEANNA. To disambiguate such queries, a disambiguation graph is constructed based on the structure of the query. Subgraphs of this graph, which correspond to SPARQL queries, are scored using a combination of syntactic similarity between its items and keywords in the query and semantic similarity between these semantic items. The top-k matching subgraphs are found efficiently using a top-k enumeration scheme based on rank-joins (Ilyas et al., 2003). By requiring that the user provide the structure, such queries can be hard for the average user formulate. The question in our running example is captured by the following structured keyword query:

"actress, played in(casablanca), married to(writer, born in(rome))".

3.3. Disambiguation Graph Construction

DEANNA starts out by constructing a disambiguation graph from the natural language question. A disambiguation graph is a way of compactly encoding all possible ways a question can be interpreted with respect to the underlying knowledge graph. Figure 3.3 shows an example disambiguation graph corresponding to the question *"Who played in Casablanca and was married to a writer born in Rome?"*. In this section we will introduce the notion of a disambiguation graph and show how it is constructed. We first give a formal definition of disambiguation graphs and interpretations obtained from them. We then move to detailing how a disambiguation graph is constructed in this section, and in Section 3.4 we detail how interpretations are obtained from them.

Definition 3.1 (Disambiguation Graph)**.** Given a question u and a knowledge graph KG, let:

- V_p be the set of phrases in u,
- V_s be the set of semantic items in KG with surface forms in V_p,
- V_d be a set of nodes, each corresponding to a dependency between a triple of phrases in V_p,
- $E_{coh} \subseteq V_s \times V_s$ be a set of weighted coherence edges connecting pairs of semantic items,
- $E_{pri} \subseteq V_p \times V_s$ be a set of weighted prior edges connecting a phrase to a semantic item,
- $E_{dep} \subseteq V_d \times V_p$ be a set of dependency edges connecting a dependency node to a phrase and labeled with one of rel, $subj$, or obj,

the disambiguation graph of u over KG, $G(u, KG) = (V, E)$, is a labeled weighted multigraph where $V = V_p \cup V_s \cup V_d$ and $E = E_{coh} \cup E_{pri} \cup E_{dep}$.

In Figure 3.3, phrases in V_p are shown in the center, dependency nodes in V_d are on the left and semantic nodes in V_s are on the right. We omit coherence edges between pairs of semantic items and the labels of other edges for readability.

We will elaborate on the details of the disambiguation graph and how it is constructed in the following sections. Before doing so, we first formalize the notion of an interpretation of a question. We do so at this stage to allow the reader to see the big picture of what we are trying to achieve with disambiguation graphs.

Definition 3.2 (Question Interpretation with Respect to a KG)**.** Given a question u, a KG, and a disambiguation graph $G(u, KG)$ as defined above, an interpretation of the question with respect to the KG is a subgraph of $G(u, KG)$.

An interpretation is essentially a decision about which edges and nodes should be kept in the disambiguation graph and which should be removed. Understanding a question with respect to a knowledge graph boils down to finding the right interpretation. Figure 3.4 on page 49 shows one possible interpretation obtained from the disambiguation graph of Figure 3.3. Moreover, not all interpretations are plausible, some interpretations are nonsensical. For instance an interpretation where a phrase maps to multiple entities, for example '*Rome*' mapping to both `Rome` and `SydneRome`, is not a plausible one. A *plausible interpretation* is one that allows us to generate a meaningful SPARQL query. We will concern ourselves with how to generate such an interpretation from a disambiguation graph and how to map it to a SPARQL query in Sections 3.4 and 3.5, respectively. In the rest of this section, we are concerned with how a disambiguation graph is constructed.

3.3.1. Phrase Detection and Mapping

The first stage in the construction of a disambiguation graph from a question is to find phrases in the question that can potentially refer to semantic items in the final interpretation of the question. In Figure 3.3, phrases are shown in rounded rectangles in the center of the figure. We first formalize this subtask by formally defining what a phrase is and what it means for a phrase to refer to a semantic item.

Definition 3.3 (Phrase)**.** Given a natural language utterance $u = (t_1, t_2, ...)$ and its dependency structure $parse(u)$ in the form of a tree, a phrase p is either (i) a subsequence of u or (ii) a subtree of $parse(u)$.

Phrases come from viewing the question as both a sequence and a labeled tree of tokens. The latter view allows us to construct phrases from non-contiguous spans of tokens. For example, in the question *"Countries in which Nobel Prize winners were born"* this definition allows us to isolate the subtree with the tokens '*were born in*' as a distinct phrase. In principle, defining phrases using the dependency structure should suffice. However, because dependency parses can contain inaccuracies, particularly when it comes to subtrees corresponding to named entities (e.g., '*Saving Private Ryan*'), we also look at subsequences of tokens as phrases as well. In this work we operate with a tree version of the Stanford dependency parse of an utterance (Marneffe et al., 2006).

The KG contains semantic items that are simply unique identifiers. To mention a semantic item in a question, we need a way to refer to that semantic item in natural language. The following definition allows us to do so:

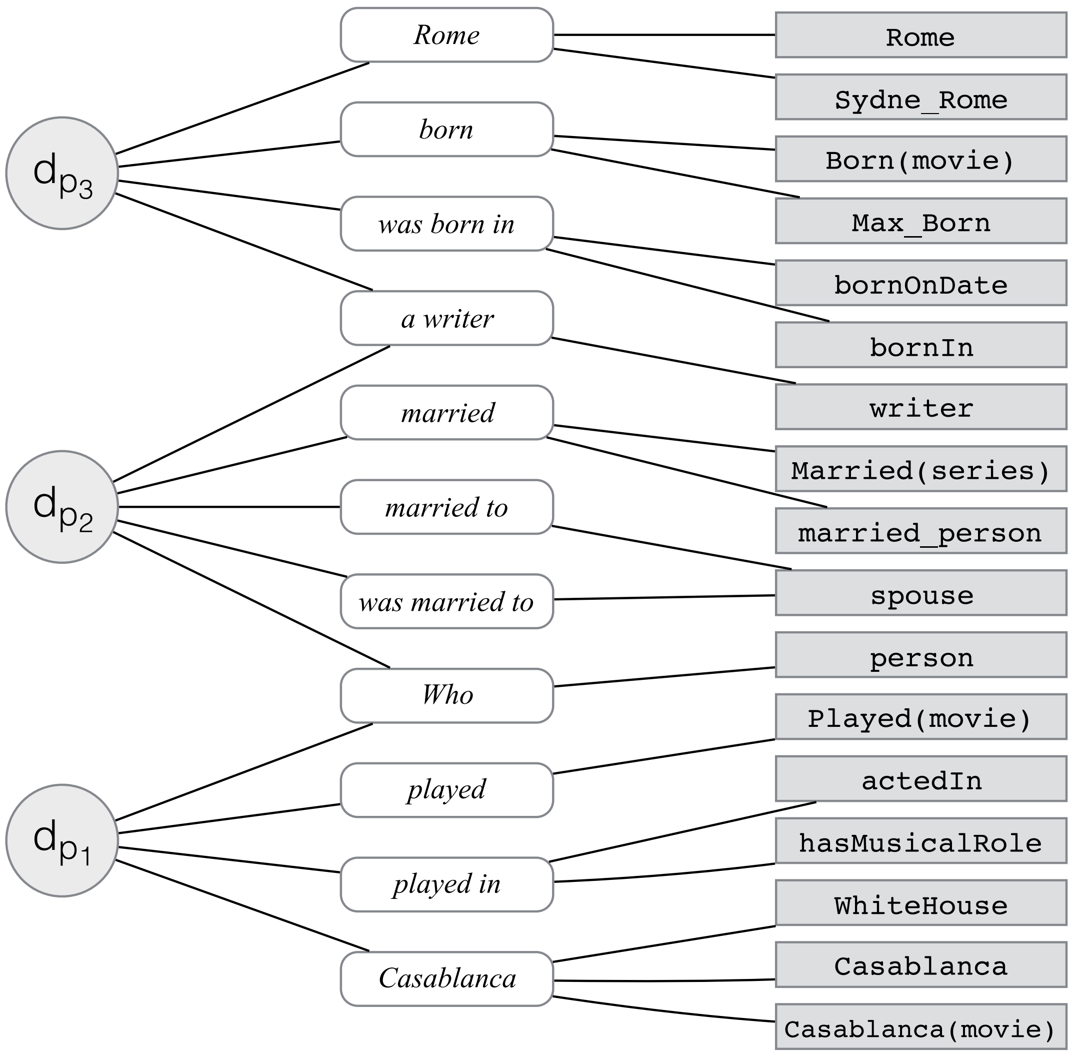


Figure 3.3.: A disambiguation graph corresponding to the question *"Who played in Casablanca and was married to a writer born in Rome?"*. Semantic coherence edges between semantic items and weights of prior edges are not shown.

Definition 3.4 (Surface Form and Mention)**.** Given a semantic item s, a surface form is a phrase p that can be used to refer to the semantic item. We say p is a mention of s in u if the phrase p intended to refer to s.

A surface form of a semantic item is one possible way of referring to it. Surface forms can be ambiguous in that the same surface form can correspond to multiple semantic items in the KG. For example, the surface form '*Casablanca*' might refer to both the Moroccan city ($s =$ `Casablanca`) or the 1942 film ($s =$ `Casablanca(movie)`). This kind of ambiguity is one of the core issues we deal with in DEANNA.

A surface form used in an utterance that leaves no room for ambiguity regarding the intended semantic item is called a mention of that semantic item. Given a surface form without its context, some semantic items are more likely than others. The notion of a *prior* captures this. For instance, given the surface form '*Rome*', the Italian capital (`Rome`) has a higher prior than the city in New York (`Rome(NY)`) and the actress `SydneRome`.

We use a set of *chunkers* for finding potential mentions, driven by the category of semantic item we are after (entity, class, or predicate). The result of phrase detection is a set of phrases $P = \{p_1, p_2, ...\}$. Once a phrase p is found, a *mapper* generates the set of semantic items whose surface form is p, along with the prior weight for this mapping. The result of this process is V_s, V_p and E_{pri} in the disambiguation graph. In what follows we discuss how chunking and mapping is performed for entities, classes, and predicates.

Entities

For entities we derive a comprehensive dictionary of surface forms from Wikipedia following Hoffart et al. (2011). For each entity e, we consider its corresponding Wikipedia page $Wiki(e)$. We take as surface forms of e the text of all links to $Wiki(e)$ in Wikipedia. To assign a prior weight for this specific surface form p mapping to e, we compute the fraction of times that p was linked to $Wiki(e)$ in Wikipedia.

It is important to note that we experimented with using third-party named entity recognizers for this task. However the results were not satisfactory as these recognizers are geared towards a very narrow class of entities (persons, locations, and organizations), and are unable to handle others that are important for our task such as movies, songs, and bands, resulting in very low recall (see Mendes et al. (2012)). In our setting, missing an entity mention can be detrimental to correctly interpreting the question at hand.

We run a dictionary-based chunker over the question using the weighted dictionary described above. Here, we restrict ourselves to phrases that are a contiguous sequence of tokens (see Definition 3.3). For each detected surface form, we produce the corresponding set of entities along with the prior weight for this mapping.

Classes

In the knowledge graph we consider, the type system is that of Yago, where classes are a combination of WordNet noun synsets (Fellbaum, 1998) and Wikipedia categories. The prior weight for this mapping is obtained form a corpus where words are annotated with their WordNet synsets. For classes corresponding to WordNet synsets, we construct a dictionary of surface forms by taking all words in a synset as surface forms for the corresponding class. For Wikipedia categories such as `RepublicanUnitedStates-VicePresidents`, we use the category's name (e.g., '*Republican United States Vice Presidents*') as its surface form. For the latter, the prior weight is set to 1.0.

Similar to entities, we use a dictionary-based chunker with the above dictionary for finding potential mentions of classes in a question.

Predicates

Compared to how entities and classes are detected above, detecting phrases potentially corresponding to predicates is more involved due to the wide variety in which predicates can be expressed in text. Here, we rely on a combination of the part-of-speech tag patterns used by Fader et al. (2011) in ReVerb and our own set of manually defined dependency parse patterns. The latter set of patterns is shown in Table 3.1, where we use the Semgrex syntax (Chambers et al., 2007). Moreover, we rely on a manually-compiled dictionary of common noun phrases mapped to predicates.

To map the phrases detected above to KG predicates, we rely on approximate matching against a dictionary of phrases to predicates compiled using PATTY (Nakashole et al., 2012). To support approximate matching we index the dictionary using Lucene, where we apply lemmatization to dictionary phrases. When a candidate predicate mention is detected, we query the index first with the lemmatized version of the phrase. If no match is found, we additionally perform stop-word removal and repeat the query.

We also define a set of patterns for *latent predicate phrases*, which potentially correspond to predicates that occur without an explicit mention. Formally, a latent phrase is a phrase composed of zero tokens induced by an explicit phrase, called the generating phrase. In our setting, latent predicate phrases are generated for demonyms and adjectives referring to locations. For example, the phrase '*American*' in the question *"American actors"* would induce a latent predicate phrase with '*American*' being its generating phrase. Latent predicate phrases are mapped to a special relation called `ANY` which in the final query appears as a variable, allowing us to map the above question to the query `?x ?r UnitedStates`.

3.3.2. Phrase Dependencies

With phrases identified and the semantic items they could potentially map to enumerated, we move to the problem of finding the dependencies among the phrases. We

Name	**Pattern**
POSS	`({pos:/N.*/}=pred ?>nn {pos:/N.*/}=nnmod ?>amod {pos:/JJ/}=adjmod) >poss {pos:/N.*/}=obj`
NOUN	`{pos:/N.*/}=pred >prep ({pos:/IN|TO/}=prep >pobj {}=obj)`
VERB	`{pos:/V.*/}=pred ?>/aux.*/ {pos:/VB.*/}=aux ?>prep {pos:/IN|TO/}=prep >/nsubj.*|dep|advmod/ {}=subj [>/dobj|dep|advmod/ {}=obj | >/prep/ ({} >/pobj/ {}=obj)]`

Table 3.1.: Dependency patterns for detecting predicate chunking and argument detection.

define a phrase dependency as follows:

Definition 3.5 (Phrase Dependency)**.** Given an utterance u, a phrase dependency is an assignment of a triple of phrases in u to the roles *pred*, *subj* and *obj*, denoted $d_p = p_{pred}(p_{subj}, p_{obj})$.

A phrase dependency is a grouping of phrases into a proposition, more specifically, an interrogative proposition. A phrase dependency within a disambiguation graph allows us to read one or more semantic dependencies:

Definition 3.6 (Semantic Dependency)**.** Given an utterance u and a phrase dependency $d_p = p_{pred}(p_{subj}, p_{obj})$ in the corresponding disambiguation graph G with the mappings $p_{pred} \mapsto s_{pred}$, $p_{subj} \mapsto s_{subj}$, and $p_{obj} \mapsto s_{obj}$, a semantic dependency is an assignment of the three semantic items s_{pred}, s_{subj}, and s_{obj} into the roles *pred*, *subj* and *obj*, denoted $d_s = p_{pred}(p_{subj}, p_{obj})$.

Once the different phrases are mapped to their intended semantic items, phrase dependencies give us the intended semantic dependencies from which we can form triple patterns to query the KG. An example of a phrase dependency in Figure 3.3 is '*was born in*'('*a writer*','*Rome*'), with one corresponding semantic dependency being `bornIn`(`writer`,`Rome`) — this happens to be the intended semantic dependency.

To form phrase dependencies, we start by finding *triploids*. A triploid is a triple of tokens assigned the roles *pred*, *subj*, and *obj*, denoted $t_{pred}(t_{subj}, t_{obj})$.

Since DEANNA is designed to deal with potentially complex questions that translate into multiple triple patterns joined with each other, it is important that we are able to capture long range dependencies in the question. For this reason, we resort to using the dependency parse of a question (Marneffe et al., 2006) for finding triploids. The dependency patterns in Table 3.1, which allow us to detect potential relation phrases, also allow us to capture the core token in these relations using the named token `pred` and the arguments of these relations using the named tokens `subj` and `obj`. These tokens are usually, though not necessarily, the lexical heads of the corresponding phrases in a phrase dependency. We form a triploid from each such token.

We form a phrase dependency by overlaying the tokens of a triploid obtained from the above patterns with the detected phrases (Section 3.3.1). In doing so, we ensure that each component of a phrase dependency pattern maps to one or more phrases found using the appropriate chunker. Namely, phrases in the *pred* component must originate from a predicate chunker, while those in the *subj* and *obj* roles must originate from one of the entity or class chunkers. The phrase dependency above is obtained from the triploid '*born*'('*writer*','*Rome*').

3.3.3. Semantic Item Coherence

The disambiguation graph also includes coherence edges connecting pairs of semantic items that can potentially contribute to the same triple pattern (i.e., connected to a common dependency node). Semantic coherence $Coh_{sem}(s_1, s_2)$ captures to what extent two semantic items s_1 and s_2 occur in the same context.

In our disambiguation framework coherence acts to counter the influence of the prior by allowing the disambiguation context to be taken into consideration. For instance, a prior would favor the Moroccan city `Casablanca` over the movie `Casablanca(movie)` for the phrase '*Casablanca*'. However, a mapping of another phrase in the context to the predicate `actedIn` would be an indicator that the movie is the intended semantic item for the phrase '*Casablanca*'.

To compute the coherence between semantic items we characterize a semantic item by the set of entities that are connected to it. In essence, the larger the intersection of the two such sets for a pair of semantic items, the higher their coherence. We formalize this using the notion of *inlinks*. Since we operate with DBpedia and Yago, whose entities are grounded in Wikipedia, for an entity e with its corresponding Wikipedia page $Wiki(e)$:

$$inlinks(e) = \{e' \mid Wiki(e') \text{ links to } Wiki(e)\}.$$

We extend this to classes and predicates as follows. For a class c,

$$inlinks(c) = \bigcup_{e \mid c(e) \in KG} inlinks(e).$$

For a predicates *pred*, we define:

$$inlinks(pred) = \bigcup_{(e_1,e_2) \mid pred(e_1,e_2) \in KG} inlinks(e_1) \cap inlinks(e_2).$$

The intuition behind the latter is that if the two arguments of a relation occur, then the relation is being expressed in the context of the links.

We define the semantic coherence between two semantic items s_1 and s_2 to be the Jaccard coefficient of their sets of inlinks:

$$Coh_{sem}(s_1, s_2) = \frac{|inlinks(s_1) \cap inlinks(s_2)|}{|inlinks(s_1) \cup inlinks(s_2)|}.$$

3.4. Optimization Model for Joint Question Disambiguation

The disambiguation graph constructed in the previous section encodes the various ways a question can be interpreted with respect to the underlying KG. We call a subgraph of this disambiguation graph an interpretation of the corresponding question. Some interpretations might be more likely than others, while other interpretations might be completely nonsensical. In this section we present our framework for disambiguating the question with respect to the KG.

Disambiguating a question in our setting means finding the correct segmentation of the question into phrases (i.e., selecting the correct phrase nodes in the disambiguation graph), mapping these phrases to the correct semantic items, and grouping the semantic items appropriately to form triple patterns resulting in a query that captures the question. As the solution to each of these problems can inform the solution of the others, we jointly resolve these ambiguities in the spirit of such works as Hoffart et al. (2011) for joint entity disambiguation and Weissenborn et al. (2015) for joint named entity and noun word sense disambiguation. The crucial difference to the latter work is that our setting additionally requires the disambiguation of predicate mentions and the semantic structure of the question. To this end, we devise an Integer Linear Program (ILP).

The intuition behind the ILP can informally be described as: find (i) the most likely interpretation of the question with respect to the knowledge graph that (ii) makes sense. The first part is captured by the objective function of the ILP, while the second is captured by its constraints.

In light of the disambiguation framework presented above, with D_S being the set of semantic dependencies in $G(u, KG)$, the output of the ILP is (i) a selected subset of phrases $V_P^* \subseteq V_P$, (ii) a functional mapping of selected phrases to semantic items $V_P^* \rightarrow V_S$, and (iii) a set of semantic dependencies $D_S^* \subseteq D_S$. Collectively, these three define an interpretation. We start by introducing the notation needed for the ILP.

3.4.1. ILP Notation

Ours is a binary (0/1) linear program, where variables can take on the values 0 or 1. We deal with two types of symbols: variables, whose values are determined by the solution of the ILP, and constants, which are provided to the ILP and serve the role of indicators of membership in a particular set.

We first introduce our variables whose assignment defines an interpretation:

- X_i indicates whether phrase p_i is in an interpretation,
  $X_i = 1 \leftrightarrow p_i$ is selected, i.e., p_i is part of the interpretation of the question.

- Y_{ij} indicates whether phrase p_i maps to the semantic item s_j in an interpretation, $Y_{ij} = 1 \leftrightarrow (p_i \mapsto s_j)$.
- Z_k indicates whether $s_k \in S$ appears in the image of the chosen mapping in an interpretation,
  $Z_k = 1 \leftrightarrow s_k$ appears the image of $V_P^* \rightarrow V_S$.
- $Z_{k,l}$ indicates whether the coherence of s_k and s_l contributes to the objective function, which is the case *iff* both s_k and s_l are chosen in an interpretation (i.e., $Z_k = Z_l = 1$), as detailed in the constraints below.
- Q_{mnd} indicates whether the phrase p_n is chosen as part of the phrase dependency $d_m \in D_p$ for the role $d \in \{predicate, subject, object\}$ in an interpretation.
- T_t indicates for semantic dependency $s_{pred_t}(s_{subj_t}, s_{obj_t})$ whether all three components are chosen in an interpretation, i.e., $Z_{pred_t} = Z_{subj_t} = Z_{obj_t} = 1$, and the corresponding SPARQL triple pattern has a non-empty result in the underlying data.

The result of the ILP is a 0/1 assignment of the X, Y, Z, Q, and T variables, from which a mapping $V_P^* \rightarrow V_S$ and a set of semantic dependencies D_S^* can be induced. The Q and T variables couple the choice of phrases and their mapping to semantic items with the dependencies among phrases and semantic items. This ensures that the output consists of meaningful triple patterns, rather than mapping individual phrases to semantic items independently. Moreover, T variables also encode whether a semantic dependency in the output actually produces answers over the underlying knowledge graph when the corresponding triple pattern (on its own) is executed on the data. The objective function below rewards decisions leading to non-empty answers.

We also need the following 0/1 constants (indicators) for our ILP, whose values are given to the ILP:

- C_j indicates whether the semantic item s_j is a class.
- E_j indicates whether the semantic item s_j is a entity.
- R_j indicates whether the semantic item s_j is a predicate.
- t_{rc} indicates whether the predicate s_r and the concept (an entity or a class) s_c are type compatible.

In the above, $\forall s_j \in S : C_j + E_j + R_j = 1$, i.e., a semantic target is either a class, an entity, or a predicate.

Finally, we define the notation needed for sets of phrases. $\mathcal{P}(t)$ is the set of phrases containing the token (word) t. The set of latent phrases is $\mathcal{P}_{lat}$, and each latent phrase $p_{lat} \in \mathcal{P}_{lat}$ is generated by a phrase $p = gen(p_{lat})$. For example, in the question *"...Australian organization"*, a latent relation phrase p_{lat_1} is generated from the

'*Australian*' as described in Section 3.3.1 with the purpose of capturing the relation between `Australia` and `organization` in the final query. In this case, $gen(p_{lat_1}) =$ '*Australian*'.

3.4.2. Objective Function

The objective of the ILP is to maximize the following function:

$$\alpha \sum_{i,j} s(i,j) Y_{i,j} + \beta \sum_{k,l} r(k,l) Z_{k,l} + \gamma \sum_{m,n,d} Q_{m,n,d} + \delta \sum_{t} T_t$$

The first term of the objective looks for maximizing the weight of the phrase to semantic item mappings. The second term seeks to maximize the coherence between the chosen semantic items. The third term forces the ILP to generate phrase dependencies whenever possible, given the constraints. Finally, the last term gives a way of preferring triple patterns that have non-empty answers.

The intuition of the objective is that it seeks an interpretation of the question that balances the prior mapping scores and the coherence among chosen semantic items while preferring non-empty interpretations whenever possible. The constraints, presented below, make sure that the final solution of the ILP results in a meaningful disambiguation from which a meaningful executable triple pattern query can be generated.

3.4.3. Constraints

Our constraints seek to prevent any nonsensical interpretations. Some of these constraints come from linguistic knowledge, others utilize the semantic knowledge in the underlying KG, while others serve to prefer interpretation with non-empty answers, all else being equal. We introduce the constraints used by DEANNA in what follows using the notation above, and explain the intuition behind each constraint.

1. A phrase maps to one semantic item at most:

   $$\forall i : \sum_{j} Y_{i,j} < 1.$$

   We disallow a phrase to map to more than one semantic item. There is an exception to this constraint, which we overcome by using latent phrases presented in Section 3.3.1 above.

2. If a mapping $p_i \mapsto s_j$ is chosen, then the target node must also be chosen:

   $$\forall j : Y_{i,j} \leq X_i.$$

3. If a mapping $p_i \mapsto s_j$ is chosen, then no phrase that overlaps with p_i can be chosen:
$$\forall t \in u, \sum_{i \in \mathcal{P}(t)} X_i \leq 1$$

4. $Z_{k,l}$ is 1 *iff* both Z_k and Z_l are both 1:
$$\forall Z_{k,l} : Z_{k,l} + 1 = Z_k + Z_l.$$
This constraint is used to decide which $Z_{k,l}$ variables affect the objective function. By having this constraint, we allow the coherence of a pair of semantic items to affect the objective function only if they are chosen in the final interpretation.

5. A phrase p_k and a semantic item s_l are chosen *iff* $Y_{k,l}$ is 1:
$$\forall Y_{k,l} : Y_{k,l} + 1 = X_k + Z_l.$$

6. Each phrase dependency can contribute to each role at most once:
$$\forall m, d : \sum_{Q_{m,n,d}} Q_{m,n,d} \leq 1.$$
This constraint ensures that if a phrase dependency has multiple phrases as candidates for a role (subject, predicate, or object), at most one of these can be chosen. The need for this constraint comes from the manner in which phrase dependencies are generated using dependency parse patterns between individual tokens. These tokens are subsequently aligned with the detected phrase, resulting in multiple phrases being candidates for the same role.

7. For $Q_{m,n,d}$ to be selected, the corresponding p_n must be selected:
$$\forall m, d : Q_{m,n,d} \leq X_n.$$
This constraint ensures that roles are chosen in a meaningful manner. By linking Q and X variables, this constraint also links Q and Y variables through constrains #2.

8. Each chosen phrase dependency (encoded in the Q variable) must include a relation phrase:
$$\forall m, n, r, d = rel : R_r \geq Q_{m,n,d} + X_n + Y_{nr} - 2.$$

9. Each semantic triple should have at least one class:
$$\begin{aligned} &\forall m, n_1, n_2, r, c_1, c_2, d_1 = arg1, d_2 = arg2 : \\ &\qquad C_{c_1} + C_{c_2} \geq Q_{m,n_1,d_1} + X_{n_1} + Y_{n_1 c_1} + \\ &\qquad\qquad Q_{m,n_2,d_2} + X_{n_2} + Y_{n_2 c_2} - 5. \end{aligned}$$

This constraint is applied only to questions that expect a tuple of entities as a result. It is necessary since a semantic dependency in an interpretation needs to generate variables that allow for (i) joins, or (ii) projection of entities. All variables in our setting come from type mentions in the question. In our running example, the semantic dependency `writer bornIn Rome` will generate the triple patterns `?y type writer` and `?y bornIn Rome`. This constraint is not applied to yes/no questions such as *"Was John F. Kennedy assassinated in Dallas?"*.

10. If any two Q variables have a token as part of two different mentions, then at most one of these two Q variables can be chosen:

$$\forall n, n', d, d', m \neq m', t \in p_n, t \in p_{n'}, p_n \neq p_{n'} : Q_{m,n,d} + Q_{m',n',d'} \leq 1$$

This constraint is most relevant when two dependency parse patterns for generating phrase dependencies place a token in two different roles such as subject and predicate, possibly within two different phrases.

11. Each predicate in a chosen semantic dependency (encoded in the values of the Q and Y variables) must have a type signature compatible with the types of its left and right arguments (classes or entities) in the semantic dependency:

$$\begin{aligned} &\forall m, n_1, n_2, n_3, r, c_1, c_2, d_1 = pred, d2 = subj, d3 = obj : \\ &\qquad t_{rc_1} + t_{rc2} \geq Q_{m,n_1,d_1} + X_{n_1} + Y_{n_1 r} + \\ &\qquad\qquad Q_{m,n_2,d_2} + X_{n_2} + Y_{n_2 c_1} + \\ &\qquad\qquad Q_{m,n_3,d_3} + X_{n_3} + Y_{n_3 c_2} - 7. \end{aligned}$$

12. A latent phrase p_l (see Section 3.3.2) can be selected only if the generating phrase p is also selected:

$$\forall p_l \in \mathcal{P}_{lat}, p = gen(p_l) : X_l \leq X_r.$$

13. If a T variable is 1 then all X, Y, Z and Q variables in the semantic dependency it encodes are chosen:
$\forall t, X_i$ s.t. $p_i \mapsto s_j$ and s_j part of $T_t : T_t \leq X_i$
$\forall t, Z_k$ s.t. s_k part of $T_t : T_t \leq Z_k$
$\forall t, Y_{i,j}$ s.t. $p_i \mapsto s_j$ and s_j part of $T_t : T_t \leq Y_{i,j}$
$\forall t, i, d$ s.t. $p_i \mapsto s_{j_d}$ and s_{j_d} part of T_t in role $d : T_t \leq Q_{t,i,d}$

14. T variables corresponding to triple patterns with no matches in the data are set to 0:

$$\forall s_{pred_t}(s_{subj_t}, s_{obj_t}) \notin KG : T_t = 0.$$

Semantic dependencies can be encoded using Q and Y variables, but we resort to the above way of expressing them for readability. This constraint biases the ILP to prefer semantic dependencies with matches in the KG, without forbidding those that do not, by having T variables show in the objective.

3.5. Query Generation

We obtain a subgraph of the disambiguation graph (an interpretation as per Definition 3.2) from the 0/1 assignments of the ILP variables. Each of the X, Y, Z, and Q variables assigned to 1 results in the corresponding node or edge being in the resulting interpretation, while all those assigned 0 are not. Figure 3.4(a) shows the result of a correct disambiguation of the question whose disambiguation graph was shown in Figure 3.3. Opaque nodes and edges are part of the interpretation, while all others are not.

The triple pattern query will naturally fall out from an interpretation generated by our ILP. Each semantic dependency results in one, two, or three triple patterns:

- Each class in a semantic dependency in the resulting interpretation will induce a triple pattern expressing a type constraint on a variable. During this process, we take care that each class originating from a distinct phrase in the question is mapped to a distinct variable. For example, the question *"Who is married to a person born in Honolulu?"* should result in two type-constrained variables, `?x type person` and `?y type person` from the distinct phrases '*Who*' and '*person*'. In this manner, we are able to obtain the correct joins between triple patterns.

- Each semantic dependency in the resulting interpretation will produce a triple pattern where were a non-ontological predicate (i.e., a predicate other than `type` and `isA`) has as its arguments zero, one or two variables. For an existential question with no joins such as *"Is Barack Obama married to Michelle Obama?"* both arguments of the relation are entities. For the example question *"Who is married to a person born in Honolulu?"*, two triple patterns with non-ontological predicates are generated: `?x spouse ?y` and `?y bornIn Honolulu`.

Figure 3.4(b) shows the triple patterns corresponding to the interpretation in Figure 3.4(a).

What remains afterwards is to select the correct projection variables if the question is not an existential one. In our concrete implementation, we work with questions which have a single target (i.e., a single projection variable). Additionally, we allow users to view the bindings of all variables in the generated query whenever they ask for an explanation of an answer (i.e., an answer tuple).

We use a set of heuristics to find the head word of the phrase corresponding to the answer type, and thereby the semantic answer type:

1. if a question word like '*who*', '*where*', or '*which*' with a modifier is present, it determines the answer type e (i.e., person, location, etc., or the type of the modifier),
2. without such words, the head noun of the question's subject determines the answer type unless this is determined by the disambiguation model to be part of an entity phrase, in which case,
3. the first common noun occurring in the sentence that maps to a class determines the answer type.

In our running example, this is the variable `?x`, which was induced by the phrase '*Who*', resulting in the following query:

```
SELECT ?x WHERE {
  ?x type person .
  ?x actedIn Casablanca(movie) .
  ?x spouse ?y .
  ?y type writer .
  ?y bornIn Rome }
```

We also allow users interested in explanations of answers to view the complete variable bindings by projecting all variables that appear in the body of the query.

3.6. Query Extension and Relaxation

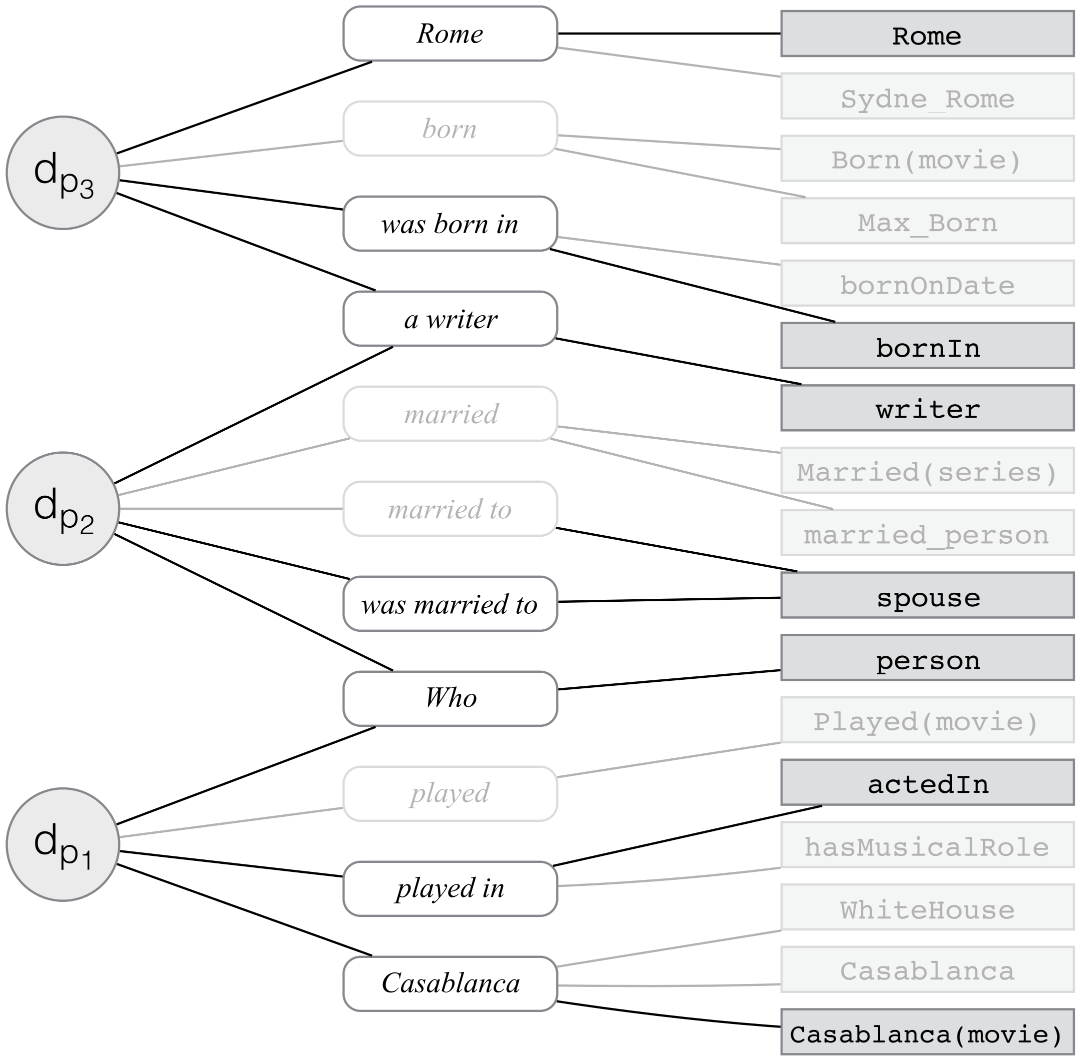
In the previous section we saw how we can obtain a structured triple pattern query from a given question. This query, however, might fail to fully capture the question, and even if it does, it might not return any answers either due to incompleteness in the KG, or due to mistranslation of one or more parts of the question.

To overcome these issues we resort to extending triples in our knowledge graph and triple patterns in the query with a textual (X) component following Elbassuoni et al. (2009), resulting in SPOX queries. We start by presenting the SPOX query model, and then discuss how we utilize it in our setting.

3.6.1. SPOX Query Model

The SPOX model builds on the triples model introduced in Chapter 2, which we adopt for knowledge graphs. Here, each fact in the KG is extended with a textual context. Formally:

Definition 3.7 (SPOX Knowledge Graph or Keyword-Augmented Knowledge Graph)**.** A SPOX knowledge graph (SPOX–KG) is a knowledge graph where each triple is



(a)

```
1. ?x type person .
2. ?x actedIn Casablanca(movie) .
3. ?x spouse ?y .
4.?y type writer .
5. ?y bornIn Rome
```

(b)

Figure 3.4.: The result of applying disambiguation to the disambiguation graph in Figure 3.3 in (a) graphical, and (b) triple pattern query forms. Semantic coherence edges between semantic items and weights of prior edges are not shown.

Subject	**Predicate**	**Object**	**X**
AmyWinehouse	type	singer	*'troubled, deep vocals, alcohol poisoning,...'*
AmyWinehouse	won	GrammyAward	*'guiness record, ...'*
AmyWinehouse	bornIn	London	*'chase farm hospital, jewish parents,...'*

Figure 3.5.: An example SPOX–KG.

augmented with a textual document $\mathbf{w}$, resulting in a set of quads of the form $t = (s, p, o, \mathbf{w})$.

The textual extension of a triple is intended to represent the context in which this fact was mentioned. Fact spotting in text is not an easy problem (Tylenda et al., 2014), so we resort to constructing this document based on keywords associated with the subject and object entities of a fact as we detail below. In this work we view $\mathbf{w}$ as a bag of words and reflect this in our scoring functions. Figure 3.5 shows an example SPOX–KG, with each fact having a textual extension from the context it was found in.

A SPOX–KG can be queried by means of a SPOX query:

Definition 3.8 (SPOX Query (Keyword-Augmented Query))**.** A SPOX Query $Q = \{q_1, ..., q_n\}$ is a set of keyword-augmented triple patterns of the form $q_i = (s_{q_i}, p_{q_i}, o_{q_i}, \mathbf{w}_{q_i})$ where $\mathbf{w}_{q_i}$ is a bag of words.

For the SPOX–KG of figure 3.5, the SPOX triple pattern `?x won GrammyAward` {*'guiness record'*} can be used to ask for people who won a Grammy award resulting in them entering the Guinness Book of Records.

Definition 3.9 (SPOX Query answer)**.** Given a SPOX knowledge graph and a SPOX query $Q = \{q_1, ..., q_n\}$, an answer of the query over the SPOX knowledge graph θ is and answer of the corresponding triple pattern query over the corresponding knowledge graph as per Definition 4.2.

The above definition of an answer operates on the structured parts of the query and the SPOX–KG. Importantly, it does not take into consideration the keywords augmented to both the query and the KG. The role of the keywords is to rank the answers produced. The text augmenting a triple pattern query can be seen as a keyword query issued over the text of the corresponding triple in the SPOX–KG.

We use a query likelihood approach to rank the answers matching a query, and follow an approach inspired by Elbassuoni et al. (2009, 2011). We also factor in the salience of the entities in the answer as well as textual relevance for a general model. We define the probability of generating the query Q from an answer θ as:

$$P(Q|\theta) = \prod_{i=1}^{n} p(q_i|\theta(q_i)),$$

thus assuming that triple patterns are generated independently. The probability of generating the triple pattern $q_i = (s_{q_i}, p_{q_i}, o_{q_i})$ from the corresponding triple $t_i = (s_{t_i}, p_{t_i}, o_{t_i})$ in the answer is defined as

$$P(q_i|\theta(q_i)) = P(q_i|t_i) = P(s_{q_i}, p_{q_i}, o_{q_i}|s_{t_i}, p_{t_i}, o_{t_i}) \times P(\mathbf{w}_{q_i}|\mathbf{w}_{t_i}).$$

Here we assume, for tractability, that the structured and textual components are generated independently.

For the generation of the structured part, we define

$$P(s_{q_i}, p_{q_i}, o_{q_i}|s_{t_i}, p_{t_i}, o_{t_i}) = (1-\beta)P(s_{t_i}) + \beta P(o_{t_i}).$$

The probabilities $P(s_{t_i})$ and $P(o_{t_i})$ reflect the salience of the subject and object. The parameter $\beta \in [0,1]$ is set according to whether s_{q_i} and/or o_{q_i} are variables in the triple pattern.

We use a unigram bag-of-words language model for the generation of the textual part and define

$$P(\mathbf{w}_{q_i}|\mathbf{w}_{t_i}) = \prod_{w \in w_{q_i}} P(w|\mathbf{w}_{t_i})$$

as the probability of generating the bag of keywords associated with the SPOX triple pattern q_i from its counterpart in the keyword-augmented triple t_i.

In our implementation we estimate $P(s_{t_i})$ and $P(o_{t_i})$ using Wikipedia's link structure, based on the number of incoming links to s_{t_i} and o_{t_i}. The bag of keywords $\mathbf{w}_{t_i}$ in the keyword-augmented triple t_i is a concatenation of the documents associated with its subject and object entities. We associate with each KG entity a textual document that consists of its Wikipedia page (including infoboxes and categories) as well as the text of all links pointing to its Wikipedia page. All probabilities are smoothed by considering the global dataset statistics.

The final answer shown consists of a ranked list of bindings of projection variables with duplicates filtered out (we only report the highest ranked binding in case of duplicates). Intuitively, for a (relaxed) query with keyword-augmented triple patterns, our model returns results that match the (relaxed) structured part of the query ranked in a manner that favors results with relevant keywords and salient entities.

We next show how the SPOX model is utilized in DEANNA for the purpose of query extension and relaxation.

3.6.2. Text Extension

The first use of the SPOX model in DEANNA is to account for tokens in the question that could not be captured by the structured query. Such words can be meaningful towards finding the correct answer. These might not have been placed in the correct structure either due to a shortcoming in the ILP framework, or because the KG does not contain the information necessary to formulate that part of the question in an SPO triple pattern. It is also possible in our framework to generate triple patterns that are not connected to the output variable(s) (the `SELECT` clause) of the final query via a join. To avoid computing Cartesian products and hard-to-interpret results, such triple patterns are discarded leading to “left over” phrases.

In the cases above, the words are attached as keywords to the triple pattern expressing a type constraint on a variable. The triple pattern to which they are attached is determined from the dependency parse of the question, where we attach a keyword to the type closest to the phrase that induced a type constraint. For example, given the question *“Which troubled singers won a Grammy?”*, the notion of ‘*troubled*’ cannot be captured by the KG , hence, the following SPOX query would be generated:

```
?x type singer {'troubled'} .
    ?x won GrammyAward
```

This results in someone like `AmyWinehouse` being ranked above `JohnLegend`.

3.6.3. Empty Result Relaxation

Text extension, as presented above, entails creating SPOX queries before there is an attempt to evaluate the query based on the “left over” words. Empty-result relaxation, on the other hand, is interleaved with query evaluation and its goal is to remedy the issue of queries with an empty answer set.

An SPO condition with no matches in the knowledge graph can indicate either a disambiguation error or an overly specific query that lacks coverage in the underlying knowledge graph due to poorly populated classes or relations. While our ILP gives preference to generating triple patterns with non-empty answers in the underlying KG, triple patterns with empty answers are still possible. It is also possible that every single SPO condition produces answers, which is what the ILP considers, but their conjunction yields and empty result.

We compute this form of relaxation is two steps:

1. Initially, each triple pattern is evaluated independently against the knowledge graph, and we check whether the set of matching triples is empty. If this is the case, we map this triple pattern back to its surface form in the question and append the resulting keywords to the triple pattern(s) expressing the type

constraint of the variables in the unmatched triple pattern (since all of our types are populated, the type-restriction on its own is always matched in the KG).

2. In the second step, we perform relaxation iteratively. We start with the triple pattern encoding the answer type: `?x type <class>`. We then iteratively add triple patterns that have a join variable (initially `?x`) in common with the previously added triple patterns. We check if the resulting query has an empty result. If this is the case, the last triple pattern added is removed. This procedure continues until all triple patterns have been exhausted. Triple patterns removed in this procedure are mapped back to their surface form in the question and are added as keywords to the type constraints on the variables they contain, or the type constraint of the projection variable if these have been removed by relaxation.

For example, for a question asking for *"Publishers of books software written in Clojure"* might be translated to the query:

```
?x type person  . ?y author ?x .
?y type OpenSourceProgram . ?y programmingLanguage Clojure
```

If, however, we do not have information about software written in `Clojure` in our KG, the SPOX query

```
?x type person  . ?y author ?x .
?y type OpenSourceProgram 'Clojure'
```

can return relevant answers.

Extreme Relaxation: The final technique we consider is to cast the entire question into a text condition with an additional type filter on the result. The latter is derived from the answer-type heuristics (Section 3.5). The iterative procedure described above for empty-result relaxation degrades into extreme relaxation if none of the queries considered produces answers. Extreme relaxation is also used when the ILP does not produce any SPO triple patterns at all. Extremely relaxed queries can still be highly beneficial, most notably when the query expresses a complex class in the knowledge graph. For example, a question asking for *"American rock bands that..."* can be answered by

```
?x type AmericanRockMusicBands {'...'}.
```

3.7. Experimental Evaluation

3.7.1. Benchmark

As our main test case, we adopt the QALD-2 benchmark (QALD-2, 2012), which consists of 200 natural language questions of two flavors: factoid and list questions, with one half designated as training and the other half designated as test questions. We discard questions that require counting or ask for literals rather than entities as answers, resulting in 18 factoid questions that are supposed to return exactly one correct answer and 30 list questions that are supposed to return proper sets of answers. Examples are *"What is the capital of Canada"* for the former, and *"people who were born in Vienna and died in Berlin"* for the latter. The QALD-2 questions we use in our experiments are listed in Table A.1. QALD-2 comes with manually compiled golden standard queries. In the case of factoid questions we take these as our ground truth. In the case of list questions, this is what we refer to as the *QALD ground truth*. Some of the methods returned additional answers that are correct, but not included in the QALD-2 ground truth. We manually assessed the correctness of these extra answers, establishing an *extended ground truth*.

In addition, we experimented with the 48 telegraphic queries used by Pound et al. (2012). 19 of these are not real questions, but are merely entity lookups and were therefore discarded. They essentially give an entity surface form and ask for the right entity. The queries we use from the benchmark of Pound et al. (2012) are all listed in Table A.2.

3.7.2. Data

The QALD-2 benchmark was designed for DBpedia 3.7, which we combine with Yago2 (Hoffart et al., 2013) to form our knowledge graph. DBpedia provides relations between entities, while Yago provides the type system (an ontology). Yago and DBpedia entities both come from Wikipedia, a fact that we exploit to merge the two KGs into one. As explained in Section 3.6, we use the Wikipedia pages of entities to construct the textual components of triples. We use PostgreSQL for storing KG triples and Lucene for indexing text, with proper linkage across the two. DBpedia contributes more than 400 million facts to our combined KG, with Yago adding another 300K from its ontology. The combined knowledge graph contains a total of 2.6M entities.

3.7.3. Performance Measures

All the systems in our primary effectiveness evaluation return a ranked list of answers $A = (a_1, ..., a_n)$, where a_1 is the highest-ranked answer. Let $rel(a_i, u)$ be the relevance indicator function which returns 1 if a_i is relevant to question u and 0 otherwise. For *factoid questions* with a single correct answer we use the established notion of mean reciprocal rank (MRR) as our main measure of quality (Büttcher et al., 2010), where the reciprocal rank (RR) is defined as follows:

$$RR(u, A) = \sum_{i=1}^{|A|} \frac{rel(a_i, u)}{i}.$$

Additionally, we report on precision at a rank cut-off of 10, where precision at a specific cut-off rank k is computed as follows:

$$Precision(u, A, k) = \frac{1}{k} \sum_{i=1}^{min(|A|,k)} rel(a_i, u).$$

For list questions we mainly report on the normalized discounted cumulative gain (NDCG) as a measure combining both precision and recall with geometrically decreasing weights of ranks (Järvelin and Kekäläinen, 2002). For a question u, its corresponding answer list A, and a rank cut-off of k, NDCG is computed as follows:

$$NDCG(u, A, k) = Z_k \sum_{i=1}^{k} \frac{2^{rel(a_i,u)} - 1}{\log_2(i+1)},$$

where Z_k is a normalization factor corresponding to the ideal answer list A^* (up to the k ranked answer) such that $NDCG(u, A^*, k) = 1$. Additionally, we give numbers for precision at different cut-off ranks.

QALD adopts set-based measures, whereas our system performs ranked retrieval to compensate for relaxation. To allow for a comparison with other systems that participated in QALD-2, we compare the results of these systems with ours at various cut-off thresholds k. We computed the recall with respect to the size of the ground truth result set, regardless of k. This generally results in penalizing our system as, for example, at most 10 relevant results can be returned for $k = 10$, regardless of the total number of relevant results out there. We define recall at a specific cut-off rank k as follows:

$$Recall(u, A, k) = \frac{1}{|golden(u)|} \sum_{i=1}^{min(|A|,k)} rel(a_i, u),$$

where $golden(u)$ is the set of ground truth answers for u.

We use the F1 score to combine the precision and recall into a single score, their harmonic mean:

$$F1(u, A, k) = 2 \times \frac{Precision(u, A, k) \times Recall(u, A, k)}{Precision(u, A, k) + Recall(u, A, k)}.$$

3.7.4. Methods under Comparison

We compare the performance of three configurations of our system in addition to a natural baseline with a strong IR flavor.

- **SPOX+Relax:** This is our full-fledged method with text extension, relaxation, and ranking based on statistical language modeling, as detailed in Section 3.6. The results obtained here are for the end-to-end task of question answering, where we combine our joint disambiguation and query extension and relaxation frameworks.

- **SPO:** This is our main baseline, which involves the generation and execution of purely structured queries with no extension or relaxation. While keeping in mind that our end-goal is to perform question answering, this baseline serves as a measure of the quality of our ILP disambiguation framework presented in Section 3.4 (the semantic parsing component).
- **SPO+Relax:** In the second baseline we restrict our framework to generate structured SPO queries, without the textual X component in SPOX+Relax above. In this case, the relaxation techniques can still choose to partially cover the question to avoid being overly specific, but this baseline does not include any keyword conditions. In both SPO and SPO+Relax, our ranking approach will rely on salience as there are no keywords. Including this baseline helps us see the effect of adding keyword-based scoring in SPOX+Relax.
- **Type+KW:** This is the simplest system we consider in this experiment. Here, we cast the question into a keyword search returning entities and combine this with a filter on the semantic type of the answer entity. This corresponds to enforcing extreme relaxation in our framework where a generated query has the form `?x type <class>` {`‘`w_1 w_2 `...’`}. To obtain a stronger baseline, we manually specified the semantic type of the answer entity. For this special case, we rank answers based on Lucene’s tf-idf score of the keyword query, which gave better results in this setting, with very minimal structure.

We tuned the parameters of the ILP and the SPOX scoring scheme using the QALD-2 training set.

3.7.5. Results

Tables 3.2 and 3.3 show the results over the QALD-2 list and factoid questions, respectively. The results show that our basic method, SPO, which reflects the result of the joint disambiguation performed by our ILP-based framework detailed in Section 3.4 considerably outperforms the pure IR baseline KW+type.

However, SPO sometimes misses some search conditions from the input question or generates overly specific queries. In other cases, the disambiguation process will contain errors resulting in empty results. This is the justification for moving towards the extension and relaxation framework of Section 3.6. SPO+Relax does not improve on SPO, demonstrating the importance of considering the keywords corresponding to the unmapped part of the question or the relaxed part of the generated query in the ranking of answers. SPOX+Relax gives the best results in our setting as it evaluates the structured query while still taking into consideration the parts of the question that could not be mapped.

As an example, take a the list question asking for *“Swedish professional skateboarders”*. We initially map it to the SPO query

	NDCG		**Precision**	
	@10	@100	@10	@100
QALD ground truth				
SPOX+Relax	**0.51**	**0.53**	**0.49**	**0.46**
SPO+Relax	0.41	0.43	0.46	0.44
SPO	0.41	0.42	0.46	0.44
KW+Type	0.24	0.29	0.15	0.10
Extended ground truth				
SPOX+Relax	**0.60**	**0.54**	**0.60**	**0.48**
SPO+Relax	0.42	0.42	0.49	0.46
SPO	0.42	0.41	0.49	0.45
KW+Type	0.30	0.41	0.23	0.13

Table 3.2.: Results for QALD-2 list questions.

	MRR	**Precision@10**
SPOX+Relax	**0.72**	**0.55**
SPO+Relax	0.54	0.50
SPO	0.53	0.50
KW+Type	0.15	0.02

Table 3.3.: Results for QALD-2 factoid questions.

```
SELECT ?x WHERE {
  ?x type Skateboarder .
  ?x type professional .
  ?x ?r1 Sweden
}
```

Although the query captures the question properly, it returns no results as the class `professional` is sparsely populated. Relaxation results in the SPOX query

```
SELECT ?x WHERE {
  ?x type Skateboarder {'professional'} .
  ?x ?r1 Sweden
}
```

which returns a satisfactory result.

The second part of Table 3.2 shows the result over the extended ground truth, which was created by pooling the results of all four systems in this experiment. While the numbers for SPO and SPO+Relax, the two methods with no textual component in the query, remain practically the same, the numbers for SPOX+Relax and KW+Type

	Precision	**Recall**	**F1**
List			
SPOX+Relax @ $k = 1$	0.50	0.15	0.23
SPOX+Relax @ $k = 10$	0.49	0.41	0.45
SPOX+Relax @ $k = 100$	0.46	0.48	0.47
SPOX+Relax @ $k = 500$	0.44	0.58	0.50
SemSek	0.28	0.29	0.29
MHE	0.26	0.36	0.30
QAKis	0.15	0.16	0.15
Factoid			
SPOX+Relax @ $k = 1$	0.68	0.68	0.68
SPOX+Relax @ $k = 10$	0.61	0.74	0.67
SPOX+Relax @ $k = 100$	0.58	0.79	0.67
SPOX+Relax @ $k = 500$	0.55	0.79	0.65
SemSek	0.71	0.78	0.74
MHE	0.52	0.57	0.54
QAKis	0.26	0.26	0.26

Table 3.4.: Comparison with other systems participating in QALD-2 based on the QALD-2 ground truth.

invariably increase. This result is interesting in that it shows how the combination of structured and unstructured data yields the best results, mostly for being able to make up for missing knowledge in the KG, and, to a lesser degree, for compensating for erroneous disambiguations of a question.

For factoid questions, the general trends are the same. SPO offers significant improvement over KW+type. SPO+Relax offers no improvement over SPO alone, but the SPOX extension with relaxation shows significant improvement over all other configurations.

We also compare our results against the systems participating in QALD-2: SemSek (Aggarwal, 2012), MHE (Unger et al., 2012b), and QAKis (Cabrio et al., 2012). Among the three systems, SemSek is the one most similar to DEANNA. SemSek uses the dependency parse of a question to induce phrases that can map to semantic items in the KG. To find the dependencies among these phrases, SemSek creates a list of phrases that starts with the phrase determined to correspond to an entity, and expands the list by adding connected phrases in the dependency parse of the question. For mapping phrases to semantic items, SemSek relies on semantic relatedness captured using Explicit Semantic Analysis (ESA) (Gabrilovich and Markovitch, 2007) over Wikipedia (similar to our notion of coherence). MHE first annotates phrases in the question with candidate entities or relations they can map to. From those, MHE constructs possible subgraphs as query interpretations and matches them against the knowledge graph (Lopez et al., 2013). Finally, QAKis is the simplest of all three systems. It is restricted to questions with a single entity connected to the answer via exactly one predicate. QAKis first determines the types of the question and answer entities and then matches the question against typed relation patterns to retrieve the most likely predicate. It is important to note that the official QALD-2 evaluation considers "partially right" answers as good enough.

On list questions, SPOX+Relax clearly outperforms all other systems on all measures. Questions vary between those that have a couple of relevant answers and those that have more than a hundred answers. As more relevant answers are viewed, there is rapid gain in recall for each cut-off threshold with little sacrifice in precision, which speaks for the ranking approach. For factoid questions, our systems is outperformed by SemSek, but with a margin smaller than the gains we make in list questions. The main issue DEANNA faced here is properly disambiguating the answer types. Sometimes, the prior weights for certain types are so high that the coherence cannot bias the mapping towards the correct type. For example, for the question asking *"Who developed Skype?"*, we could not return `SkypeTechnologies` as the prior for '*Who*' mapping to the type `person` was simply too high. Not getting the answer type correctly is detrimental for DEANNA, as even relaxation would not work in this case.

Finally, for the telegraphic query workload of Pound et al. (2012), SPOX+Relax again turned out to be the best method in our experiments. For factoid questions we achieve an MRR of 0.83 and for list questions a precision@10 of 0.73. These numbers are similar to the ones reported by Pound et al. (2012). We note that the results are not directly comparable, as that prior work used an old, smaller version of Yago as the underlying KG and reported the combined numbers of all questions, regardless of their nature (simple entity lookups, factoid questions, and the generally more difficult list questions).

Table 3.5 shows the results we obtained for some questions. The first two questions are from QALD-2, and the third is from the telegraphic query workload of Pound et al. (2012). For the first two, we show the query generated initially, the relaxed query, the number of results in each of the two ground truths we consider, and the number of relevant results in the top 10 answers with respect to each of the two ground truths (both return more than 10 answers). We also show an example of a correct answer that was not part of the original QALD-2 ground truth, but was added with the new ground truth (indicated with a +). Moreover, for each QALD question we give an example of a correct (✓) and incorrect (✗) answer returned by SPOX+Relax. We discussed the first question earlier. The second one results in a SPOX pattern query that includes a keyword component, as the system could not map the verb '*dwelt*' to an appropriate relation. This query returns satisfactory results to the user. For the last query, despite the fact that the query generated fully captures the user's intention, no results are returned by DBpedia, hence the need for relaxation. The precision at rank 10 (again, this query returns more than 10 results) is equal to 1.0, which means that all returned answers are Grammy-awarded guitarists.

3.8. Discussion

We presented DEANNA, a framework for natural language question answering over knowledge graphs. DEANNA aims to make the abundance of data in knowledge graphs available to ordinary users by providing them with an expressive interface to the knowledge graph that shields them from dealing with all its potential complexities.

DEANNA works in two stages. In the first, a user's question is automatically mapped to a triple pattern query. Here, the main problem is ambiguity, both structural and terminological. We formulate the disambiguation problem as an integer linear program where an objective function allows us to look for the most likely interpretation of the question, and a set of constraints make sure that the chosen interpretation is one that makes sense. In the second stage, unsatisfactory queries are dealt using query extension and relaxation where facts in the knowledge graph are extended with textual context, and crisp query conditions can be relaxed to keyword matching against this context.

While the relaxation and extension framework helps take care of shortcomings in the disambiguation performed by our ILP framework, it is important to understand the sources of these shortcomings. We already discussed the issue with the prior weights for some types being too high, preventing coherence from properly playing its role in the disambiguation process. This suggests the need for further investigation of the computation of these weights, and the resources that can be used to this end.

Another important issue is the dictionaries used to both find and map potential mentions to the corresponding semantic items. While these are comprehensive for entities and simple types (e.g., `actor`, `director`, `movie`), they contain gaps when it comes to more sophisticated types such as `ScientistsOfItalianOrigin`, and predicates. For these sophisticated types, which in our setting come from the Wikipedia categories, there is very little prior work on finding their paraphrases, a research topic we are currently pursuing. For relations, we rely on paraphrases extracted from a large entity-annotated textual corpus following PATTY (Nakashole et al., 2012). There are two issues here: (i) the coverage of the specific corpus, and (ii) the discrepancy between how relations are expressed in (declarative) text and (interrogative) questions. As an example, consider the question *"What does Tom Hanks do?"*, and the discrepancy between how the question expresses the `job` relation, and how it would be expressed in text (the source of our predicate paraphrases).

DEANNA is an important first step in our effort towards question answering over knowledge graphs. There are several natural extensions to DEANNA along the fol-

			Ground truth size		**P@10**	
Question	**Generated Query**	**Relaxed**	QALD	Extended	QALD	Ext.

"Swedish professional skateboarders"	`?x type skateboarder`
	`?x ?r1 Sweden.`
	`?x type professional`
		`?x type skateboarder`
		{'*professional*'} .
		`?x ?r1 Sweden.`
			2	3
				+ `PerWelinder`
					0.2	0.3

					✓ `AliBoulala` ✗ `RuneGlifberg`	

"Which Greek goddesses dwelt on Mount Olympus"	`?x type`
	`greekGoddesses`
	{'*dwelt mount olympus*'}
		Relaxation not needed	7	20
				+ `Hestia`
					0.4	0.8

					✓ `Demeter` ✗ `Circe`	

"guitarists awarded a grammy"	`?x type guitarist.`
	`?x award Grammy.`
		`?x type guitarist`
		{'*awarded grammy*'}.
			Pound et al. (2012) telegraphic workload		1.0	

Table 3.5.: Anecdotal examples of query generation and relaxation.

lowing dimensions: (i) support for more expressive queries, and (ii) support for wider variety of natural language input. In terms of expressiveness, we want to support such operations as aggregation, sorting, and functions (particularly temporal and spatial ones). As for language variety, our dependency parse patterns for capturing the structure of the question have assumed well-formed questions, relying on relaxation to take care of issues that could arise from ill-formed questions.

We envision a system that tackles the above problems by automatically mining question-query templates from large question repositories such as community question answering (CQA) sites. By using questions coupled with their corresponding queries (or answer, through which queries can be obtained), we can generate such templates with the expressiveness needed by users. Since these are based on questions issued by normal users, we can expect such templates to account for ill-formed language. There are many issues to tackle towards this goal, making it an interesting research direction.

DEANNA is a domain-general framework. It supports functionality that is not dependent on the particular domain of the underlying knowledge graph as long as it fits within the KG framework presented in Chapter 2. To allow for flexible domain specialization, we need to carefully consider the architecture of DEANNA. The core issue in domain specialization seems to be the handling of domain-specific concepts and the semantic functions they evoke (e.g., "revenue" in the financial domain). Here, it seems natural that a framework will have domain-general and domain-specific components. The details of how each should be designed, how the two interface, and how disambiguation happens in this setting are all directions to explore.

The approach we presented has specific components which we instantiate. These include chunkers for phrase detection, surface form dictionaries, and prior and coherence scoring. Their instantiations in this work mostly make use of the contents and structure of Wikipedia, because we work with knowledge graphs based on Wikipedia. An important question is how these components can be instantiated for other knowledge graphs. The general problem is that we need a resource annotated with semantic items in the KG to allow us to compute certain statistics about these semantic items. This problem is faced by other works that rely on a setting similar to ours. A general solution for this problem is not possible. However, it would be interesting to see how these components could be instantiated in different settings (e.g., through access to query logs and click-through data).

The relaxation framework, while improving DEANNA's effectiveness, sacrifices part of the structure in the question to improve recall. It does this by casting some SPO query conditions into a textual X component in a SPOX query. An important question is how to perform relaxation by falling back onto text without sacrificing this structure. We address this problem in the next chapter.

4. TriniT: Relationship Queries over Extended Knowledge Graphs

4.1. Introduction

4.1.1. Motivation

Knowledge graphs have seen rapid adoption by organizations for storing their data. This is, in large part, due to the ease with which they can be extended, particularly when it comes to predicates, owing to their schema-free nature as discussed earlier in Section 2.1. Additionally, by maintaining relationships between entities in the form of facts, knowledge graphs allow for very sophisticated triple pattern queries about entities and the relationships between them. The expressiveness offered by such queries is beyond what corpus-based IR systems can offer. However, the sheer size of a knowledge graph, the diversity of the vocabulary used, coupled with the lack of a schema means that querying a knowledge graph can be a challenging task. The result is that oftentimes a triple-pattern query issued over a knowledge graph will fail to return satisfactory answers to the user.

In a best-case scenario, a user might be able to retrieve the results she was looking for in the knowledge graph after multiple rounds of tedious query reformulation. The problem here is that the user is initially unfamiliar with the terminology and structure of the knowledge graph. For instance, are players in `LaLiga` (the Spanish football league) directly connected to it in the KG, or are they connected to the teams they play in using a relation like `playsFor` relation, and these teams, in turn, are connected to the league? Depending on how the KG grows, it is even possible that both are true as is the case in Figure 4.1. Here, two complementary triple pattern queries would be needed to articulate the information need asking for `LaLiga` players:

```
SELECT ?x WHERE {?x playsIn LaLiga},
```

and

```
SELECT ?x WHERE {?x playsFor ?y .  ?y league LaLiga}.
```

Formulating multiple queries can be a tedious task. In reality, a user might not be aware of the fact that multiple complementary queries are needed to get the complete set of results she expects.

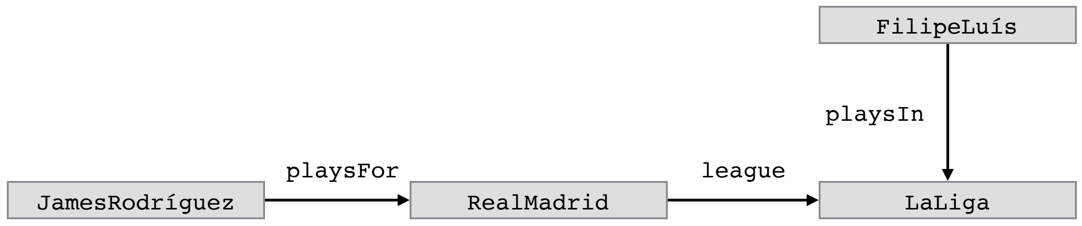


Figure 4.1.: An example of two ways a player could be connected to the league they play in, complicating query formulation.

Another common problem that we touched on in the previous chapter is the incompleteness of knowledge graphs. The knowledge graph's vocabulary might not be sufficient to formulate the desired query. Even when it is, the KG might lack some necessary predicate instances to provide some or, in the extreme case, all desired answers. Despite the large size of a typical knowledge graph, incompleteness is inevitable as the world changes rapidly.

In all cases, the end result is the same: users left with partial or empty results in response to their queries. The two issues above, the vocabulary and structure gap between a KG and a query, and the incompleteness of knowledge graphs, are addressed in this chapter. What is needed is a system that can take a user's query and automatically relax it until relevant answers can be returned. Furthermore, to compensate for incompleteness in the knowledge graph, its contents need to be extended from external sources. This chapter presents TriniT, a framework designed for flexible querying of extended knowledge graph. In contrast to earlier approaches such as SPOX presented in the previous chapter, TriniT overcomes the above problems without sacrificing the structure that is integral to the expressiveness of triple pattern queries.

4.1.2. Problem Statement

The problem we deal with in this chapter is satisfactorily answering triple pattern queries over knowledge graphs, as defined in Section 2.1.5, while accounting for possible mismatches between a user's query and the data in the knowledge graph, and missing data in the knowledge graph. We elaborate on the problem definition in what follows.

A satisfactory query result is one with good precision and recall, i.e., it includes as many of the relevant answers as possible and as few irrelevant ones as possible. Because we deal with potentially incomplete knowledge graphs, a point we elaborate on below, some answers might be relevant while not strictly adhering to the specifications of the query. In judging the relevance of an answer we assume that the query is simply one possible expression of an information need, and that the judgment of relevance of an answer is based on the information need, rather than the query.

A mismatch between a query and the underlying knowledge graph can be terminological or structural. A *terminological mismatch* occurs when a semantic item or literal in a query does not occur at all in the KG, or the intended one is expressed differently from the one in the query. In general, these are mismatches that prevent a single triple pattern in the query from matching a single (potential) triple in the KG. A *structural mismatch* occurs when considering relations between semantic items in the query that do appear in the KG and are connected to each other in it, but the path connecting the two does not match that in the query. We gave an example of this above, with the connection between football players and the leagues they play in possibly going through teams. Put differently, a structural mismatch occurs when a set of triple patterns in the query is intended to match a set of triples in the KG, with the two sets being of different sizes.

In saying that a KG can be incomplete, we assume that it is attempting to model some domain by casting as much as possible of that domain into facts. However, due to various factors such as stringent quality controls or insufficient human and computational resources, the knowledge graph may lag behind. The full knowledge is buried in documents such as Web pages, technical reports, spreadsheets, emails, etc. Such knowledge can be exposed using some form of information extraction, possibly at lower accuracy than the KG. Missing data in the knowledge graph can take various forms. An entity, predicate, or class in the query might not exist in the knowledge graph at all. Alternatively, a fact relevant for a query might be missing. In this case, all individual resources relevant to the query, including potential answers, exist in the KG, but are not connected in a manner that allows a match with the query.

4.1.3. Contributions & Overview

We make two contributions towards a solution to the problems outlined above. First, we extend the traditional KG model (see Definition 2.3) to accommodate general textual triples obtained using various forms of information extraction (see Section 2.2.2). The justification for doing so is that knowledge graphs typically constitute a small fraction of the information maintained by an organization. What information makes its way into a knowledge graph is generally a matter of judgment and is constrained by limitations on manpower and computational power and quality requirements. The complete knowledge an organization has at hand is usually available in much less crisp textual form, such as manuals, Web pages, tables, etc. Such documents are often annotated with links to the knowledge graph. We extend the KG model to allow for facts extracted from such documents. This extension of the data model also requires an extension of the query model of Section 2.1.5 so that the extended KG can be queried.

Our second contribution is a query evaluation framework over the extended KG that can dynamically relax queries by automatically rewriting query conditions. Since query relaxation results in a query returning a large number of answers, we need to assign scores to answers. We take a language modeling approach to scoring answers of a given query in isolation. Relaxation of a query is done by invoking weighted relaxation rules at query processing time. The relaxation weight essentially captures the semantic drift between the query condition being relaxed and its relaxation, and the scores of answers of the resulting query are attenuated by these relaxation weights. We present a scheme for aggregating the scores of answers obtained through different relaxations. Our score aggregation scheme is judiciously designed to allow for efficient top-k-style query processing where a query relaxation is invoked at query processing time only if it can realistically contribute to the top-k scoring answers. Adopting a top-k approach relieves the query processor from the need to explore the entire space of queries that can be induced by the relaxations, which can be prohibitively large.

We start by placing TriniT in the context of earlier work in both the IR and database communities in Section 4.2. In Section 4.3 we formally define extended knowledge graphs and discuss how they can be constructed and queried. Section 4.4 presents our framework for triple pattern query relaxation, and discusses concrete relaxations considered in this work. As relaxations can result in a potentially large query space to explore and large results, we present in Sections 4.5 our scheme for answer scoring. In Section 4.6 we present our query processing scheme and elaborate on how it interacts with our scoring model to allow for efficient query execution using top-k query processing. Finally, in Section 4.7 we present experiments demonstrating the effectiveness of TriniT in comparison to earlier state-of-the-art systems for both entity and entity-relationship search.

4.2. Related Work

4.2.1. Entity Search

Starting with the initial work of Fang and Zhai (2007), Nie et al. (2007), Serdyukov and Hiemstra (2008), and Vallet and Zaragoza (2008), the methods for entity search over large text corpora have been greatly advanced (Balog et al., 2012). In these models, a query is a bag of keywords, and the result of a query is a ranked list of individual entities, each of which is an answer. Some methods use knowledge graphs for feature expansion (Dalton et al., 2014), but stick to the same query-and-answer model. One of the currently best methods is that of Balog et al. (2011), which is based on entity language models and harnesses entity categories (i.e., semantic types) for ranking and for restricting answers to the desired type – so it can, for example, ensure that a query returns only songs, not movies, albums, or singers. However, the model is still limited to computing a single list of single entities. So unlike TriniT, there is no way of returning tuples of entities, such as song-movie pairs, as answers. We included the method of Balog et al. (2011) as a state-of-the-art baseline in our experiments.

4.2.2. Query Relaxation

In IR, the classic case for query relaxation is query expansion for keyword queries (Xu and Croft, 1996) or recommendations for query reformulation (Boldi et al., 2011). Some works along these lines have explored the use of thesauri or knowledge graphs to generate semantically related terms for a given query (e.g., Theobald et al. (2005) and Voorhees (1994)). For completely structured data, generating relaxed queries has been explored by Chaudhuri et al. (2004), Mottin et al. (2013), and Zhou et al. (2007) for relational data and by Elbassuoni et al. (2011) for RDF data. For tree-structured XML data, a number of works have developed structural relaxation techniques, such as rewriting an XPath child condition into a descendant condition (Amer-Yahia et al., 2004, 2005; Cohen et al., 2003). Theobald et al. (2008) integrated semantic-relatedness-based relaxations for content terms in XPath queries. However, none of these is suitable for the combination of graph-structured data and text corpora.

4.2.3. Search on Knowledge Graphs

There is plenty of work on querying RDF databases and Linked-Open-Data with SPARQL (see, e.g., Heath and Bizer (2011), Huang et al. (2011), Neumann and Weikum (2008), and Tummarello et al. (2010)). However, this is exact-match querying on structured data emphasizing efficiency and scale while disregarding ranking or relaxation. Keyword-based graph search has also been extensively studied for relational databases (Yu et al., 2009). These include ranking, but are limited to structured data and do not consider full documents attached to graph nodes.

The most notable works on ranking answers of SPARQL queries are Elbassuoni et al. (2009) and Kasneci et al. (2008), using statistical language models and supporting entity-tuple answers. Our approach is largely inspired by these models. The model of Elbassuoni et al. (2009) is the basis for the SPOX model used for query extension and relaxation in DEANNA as we detailed in Chapter 3. This prior work associates teXtual keywords with each triple in the KG and allows users to add textual conditions to each triple pattern in a query — hence the X in SPOX. The textual keywords associated with a KG triple come from the extraction context of a fact. A query is answered in this setting by first matching its structured SPO components against the KG and subsequently ranking the answers by how well their textual components match those in the query. We use this model as one baseline in our experiments and elaborate on it in Section 4.7. In contrast, our extended KG setting allows for text-based triples, which means that text can be used to express crisp structure and relationships rather than simple bag of words matches.

The same limitation holds for work on graph query languages such as Sagiv (2013), Wood (2012), and Yang et al. (2014), but they do feature path relaxation techniques. Finally, there are methods for searching and ranking over RDF-structured Linked Data and KGs with queries that combine keywords and entity examples (e.g., Bron et al.

(2013)) or interpret telegraphic text queries on the underlying structured data (e.g., Pound et al. (2012)). Again, this does not extend to our more demanding case of an extended knowledge graph.

4.2.4. Querying Entity-Annotated Text

Searching and exploring text corpora that are annotated with entities and/or linked to a KG has been addressed in various projects, most notably the Broccoli system (Bast and Buchhold, 2013; Bast et al., 2014), Facetedpedia (Li et al., 2010), ERQ (Li et al., 2012), and STICS (Hoffart et al., 2014b). Albeit not based on SPARQL, these are very expressive and powerful search engines. However, except for ERQ (see below) they do not provide any non-trivial ranking of results.

The work of Joshi et al. (2014) and Sawant and Chakrabarti (2013) addresses telegraphic text queries over combinations of text and structured data, reminiscent of our notion of extended knowledge graphs. The approach here is to jointly learn the segmentation, the entity, class and predicate interpretation of the input query (in text form), and the ranking of candidate results. There is no notion of structured queries, though, and the more advanced queries that our model allows are beyond the scope of that prior work. Most importantly, queries that need to test for multiple relationships and return tuples of entities are not supported. Another model that addresses semantic similarity measures over entity-annotated text corpora is that of Schuhmacher and Ponzetto (2014). However, that work is not about search, there is no query language.

Closest to our approach is the ERQ system by Li et al. (2012). This work integrated text conditions into a structured query language with typed variables for both entities and entity pairs (i.e., relationships). Albeit primarily addressing richer entity-relationship search over Wikipedia, the ERQ method could be applied to our notion extended knowledge graphs. We therefore include it as a baseline in our experimental studies.

4.3. Data Model and Query Language

4.3.1. Extended Knowledge Graphs

Our starting point is a knowledge graph, which, as given in Definition 2.3, is a collection of facts in the RDF SPO data model. To recap, the elementary components of a knowledge graph are semantic items S and literals L. Semantic items are canonical objects encoded as URIs and can be further divided into entities, classes and predicates. Literals correspond to such things as dates, numbers, and strings. Predicates P are a special type of semantic item which can be thought of as binary relations connecting a subject and an object.

Knowledge graphs are never complete and will not be able to fully cover the domain they are intended to describe. This can be due to limitations in man power and computational power, stringent quality control requirements or the inherent delay in updating the KG to reflect the current state of the world. The result is usually KGs which are missing entities, predicates or facts and therefore users with queries for which they cannot obtain satisfactory answers. The information missing from an organization's KG, however, will exist in some textual source, which we can use to extend the KG with new entities, classes, predicates, and facts. To do this, we combine our KG with textual corpora annotated with entities, predicates, and classes, some of which might already exist in the KG whereas others might be missing. New facts are extracted using Information Extraction (IE) techniques, and become additional knowledge used to bridge the gap between a user's information need and the KG. More specifically, we use OpenIE methods (see Section 2.2.2 and Chapter 5) to extract triples that consist of two noun phrases (S and O) that are connected by a noun phrase or a verb phrase. We additionally employ methods for Named Entity Disambiguation (NED) like that of Hoffart et al. (2011) to map the two noun phrases to entities in the KG whenever possible. This process is error prone, but can be tuned to favor precision over recall.

As an example, Figure 4.2(a) shows the original KG we start with. OpenIE identifies further triples shown in Figure 4.2(b), where some of the SPO components are mere text phrases as they could not be mapped to a canonical entity, class, or predicate. The union of the triples in both tables of Figure 4.2 forms the extended knowledge graph, XKG for short. In an XKG, we no longer refer to semantic items and literals, but instead collectively refer to these simply as tokens T.

Definition 4.1 (Extended Knowledge Graph (XKG))**.** An extended knowledge graph (XKG) is a bag of triples over tokens; that is, triples from $T \times T \times T$.

We note that as opposed to the the KG (Definition 2.3), the XKG is a multi-set, which takes into account that the IE process can produce the same fact multiple times from different documents. We exploit this redundancy later in our ranking model.

4.3.2. Triple Pattern Queries

We now define the triple pattern query language for querying XKGs. The query language is a modification of the triple pattern query language defined in Section 2.1.5. In addition to tokens T used to construct the XKG, we need a set of variables V that are distinct from tokens and are prefixed with a question mark. These variables will serve as placeholders for tokens the user is asking for.

Definition 4.2 (Triple Pattern and its Answers)**.** A triple pattern q is a member of the set $V \cup T \times V \cup T \times V \cup T$. An answer a to a triple pattern over an XKG is a total mapping of variables in q to T such that the substitution of the variables with their mappings, denoted $a(q)$, results in a triple t in the XKG.

S	P	O
`BangBang`	`type`	`song`
`SpaceOddity`	`type`	`song`
`KillBill`	`type`	`movie`
`WalterMitty`	`type`	`movie`
`SpaceOddity`	`usedIn`	`WalterMitty`
`SpaceOddity`	`performedBy`	`DavidBowie`
`KillBill`	`hasSoundtrack`	`KillBillAlbum`
`KillBillAlbum`	`contains`	`BangBang`

(a)

S	P	O
`DavidBowie`	‘*born and lives in*’	`UK`
`DavidBowie`	`won`	‘*best British singer*’
`BangBang`	‘*by*’	`NancySinatra`
‘*Sinatra’s daughter*’	`bornIn`	`USA`
`NancySinatra`	‘*is an*’	‘*American*’
‘*Lonely Shepherd*’	‘*appears in*’	`KillBill`
‘*Lonely Shepherd*’	`performedBy`	‘*Zamfir*’
‘*Zamfir*’	`bornIn`	`Romania`

(b)

Figure 4.2.: Example triples in (a) a KG and (b) additional triples resulting from information extraction. The combination of both tables in the XKG.

```
 SELECT ?s ?m WHERE {
   ?s type song .  ?m type movie .  ?s usedIn ?m .
   ?s performedBy ?x .  ?x bornIn UK
 }
```

(a)

```
 SELECT ?m ?s ?x WHERE {
   ?s type song .  ?m type movie .
   ?m hasSoundtrack ?a .  ?a contains ?s .
   ?s performedBy ?x .  ?x ?p 'American'
}
```

(b)

Figure 4.3.: Example queries over the XKG of Figure 4.2.

Triple patterns are the building blocks of queries. The definition of a query we give here is the same as that of Definition 2.6 except that the nature of triple patterns has changed by introducing the notion of tokens. We restate the definition here for completeness.

Definition 4.3 (Query)**.** A query $Q = \{q_1, ..., q_n\}$ is a set of triple patterns q_i and a set of projection variables $P(Q)$. We require that the join graph whose vertices are q_is and where an edge connects each pair of triple patterns with a common variable to be a connected graph (i.e., no cross products). $P(Q)$ is a (usually proper) subset of the variables in Q, defining the output structure (typically entity tuples).

We also restate the definition of relationship queries first given in Definition 2.9 since they serve as a crucial motivation for our framework distinguishing it from more traditional IR tasks such as document or entity retrieval.

Definition 4.4 (Relationship Query)**.** Let $Vars(Q)$ be the set of variables in Q. A query Q is called a relationship query if $|Vars(Q)| \geq 2$.

Finally, we define query answers in light of the change in our data and query model.

Definition 4.5 (Query Answer)**.** For query Q, an answer (a) is a mapping of the variables in Q to tokens in T. Applying an answer (a) to a triple pattern $q_i \in Q$ results in the triple t_i, we denote this by $a(q_i) = t_i$. The restriction of a query answer to bindings of variables in $P(Q)$ is called a projected answer, denoted a_P .

Examples
Figure 4.3 shows examples of two queries over the XKG of Figure 4.2. The query in

Figure 4.3(a) asks for movies with British songs. This query, issued over the KG in Figure 4.2(a) only (without the extension), would have no results as the last triple pattern cannot be matched, whereas the XKG has the information needed to obtain the desired result tuple `(WalterMitty, SpaceOddity)`.

The second example query in Figure 4.3(b) asks for movies with American songs and their singers. This query requires the elaborate use of long paths in the XKG (i.e., join chains) and directly makes use of text-based triples in the XKG.

Formulating such sophisticated queries is awfully hard for a user who does not know the details of the underlying XKG. A seemingly perfect query, as is the case for the first query, can result in no answers being returned. A small change in the formulation of the second query could have easily resulted in a similar problem. We next show how such sophisticated queries that return answers can be generated automatically by rewriting user queries based on query relaxation rules.

4.4. Query Relaxation

Moving from the KG to the XKG is only one component in our scheme for coping with queries that cannot be answered from the KG in a satisfactory manner. The second component is *query relaxation*, where the goal is to automatically rewrite one or more triple patterns in a query in order to obtain answers that will satisfy the user's information need but could not be returned by the original query. As we discussed earlier, the need for query relaxation can be shown by contrasting the XKG of Figure 4.2 with the query of Figure 4.3. Having motivated the need for query relaxation, we present our framework for query relaxation in what follows.

4.4.1. Relaxation Framework

We first formally define our framework for query relaxation and then discuss the specific choices of relaxation rules.

Definition 4.6 (Relaxation Rule)**.** Given a query $Q = \{q_1, ..., q_n\}$, a relaxation rule is a triple $r = (\mathbf{q}, \mathbf{q}', w)$, where $\mathbf{q} \subseteq Q$ is the non-empty domain of the relaxation rule, $\mathbf{q}'$ is a set of triple patterns called the range, and $w \in [0, 1]$ is a relaxation weight that captures the closeness between $\mathbf{q}$ and $\mathbf{q}'$.

For readability, we also use the notation $r : \mathbf{q} \rightarrow \mathbf{q}'(w)$ for the rule $r = (\mathbf{q}, \mathbf{q}', w)$.

Definition 4.7 (Relaxation Rule Application and Relaxed Query)**.** Given a query $Q = \{q_1, ..., q_n\}$ and a relaxation rule $r = (\mathbf{q} \subseteq Q, \mathbf{q}', w)$, the application of r to Q results in the query $Q' = r(Q) = (Q \setminus \mathbf{q}) \cup \mathbf{q}'$ with $P(Q') = P(Q)$.

Given a query Q and a sequence of relaxation rules $\mathbf{r} = (r_1, ..., r_m)$, a relaxed query is a query $Q' = r_m(...r_1(Q))$.

The restriction that the set of projection variables does not change before and after a relaxation is applied means that no relaxation can result in the complete removal of a projection variable from the body of the query. Additionally, as per Definition 4.3, a relaxation cannot result in a cross product, as queries must have a connected query graph.

We do not allow for recursive application of relaxations. That is, a triple pattern obtained through a relaxation is not subject to subsequent relaxation. This is in line with established practice in query rewriting and expansion in IR systems. Moreover, it ensures efficient query processing and the scoring scheme we present below is unlikely to produce high-scoring answers from recursive relaxation.

Going back to our example XKG in Figure 4.2 and the query for movies with British songs in Figure 4.3(a), the original query will not be able to return all the relevant answers (as a matter of fact it returns no answers at all). One possible way to handle this would be to simply drop the culprit triple pattern talking about birth in the UK. This, however, would result in losing part of the intention of the user's query. A better alternative is to relax the overly restrictive predicate `bornIn` to the textual predicate '*born*' or by a variable that can be matched by any predicate or token. Likewise, it is possible that a relaxation for the entity `UK` is needed, where we could generate such tokens as '*British*', '*English*', or '*Scottish*', or entities (which are also tokens) like `England` and `Scotland`. Our framework generates these variants automatically.

Some relaxations will drift away from the original query more than others. For example, '*from*' is a cruder approximation for the predicate `bornIn` than the token '*born in*'. Similarly, '*Scottish*' can be seen as a cruder approximation of `UK` than '*English*'. Replacing a token by a variable in both cases is an even cruder approximation. In our framework, the cruder an approximation is, the lower the weight assigned to the corresponding relaxation rule.

The order in which relaxations are applied results in different queries. In Sections 4.5 and 4.6 we discuss our query ranking and processing schemes. The two are designed to incrementally explore relaxations only if they can contribute to the top-k-scoring answers. This approach avoids the need for explicit enumeration of all possible relaxed versions of a query, which is usually prohibitively expensive.

4.4.2. Concrete Relaxations in TriniT

TriniT provides a programmatic interface for users or KG administrators to implement their own relaxations. Implementations of this interface can tap into any resources needed such as the KG itself, third party lexicons and dictionaries, and Web services.

In this dissertation, we consider two concrete forms of relaxation: structural relaxations and predicate paraphrasing. Structural relaxations result in replacing a triple pattern with a set of triple patterns that conceptually denote a path. We apply this relaxation to spatial predicates that connect an entity with a location (e.g., `(?x bornIn`

`UK)` to `(?x bornIn ?y . ?y locatedIn UK)`).

Predicate paraphrasing is the most important type of relaxation we consider in this dissertation. We generate paraphrases for XKG predicates using the XKG itself. For each predicate (e.g., `graduatedFrom`), we generate paraphrases (e.g., '*went to*') and inverse paraphrases (e.g., `alumnus`) by considering the overlap between predicate arguments in the XKG. Given two predicates, including textual ones, p_1 and p_2, where $args(p_i) = \{(s,o)|(s,p_i,o) \in XKG\}$ (i.e., subject-object pairs linked in the XKG by p_i), the weight assigned to the relaxation

$$r = (\{?x \;\; p_1 \;\; ?y\}, \{?x \;\; p_2 \;\; ?y\}, w)$$

is:

$$w = \frac{|args(p_1) \cap args(p_2)|}{|args(p_2)|}.$$

Inverse paraphrases are generated and weighed in the same manner as above, by matching the XKG with an inverted version of itself, where the subject and object components of triples are switched. In our experiments, we do not consider paraphrases that are stop words, as these consistently hurt results. Table 4.1 shows examples of paraphrases and inverse paraphrases (the latter indicated by the superscript $^{-1}$) for both KG and textual predicates extracted from our XKG.

Predicate	**Paraphrase**
`graduatedFrom`	'*graduated from*'
`graduatedFrom`	'*went to*'
`graduatedFrom`	'*alumnus*' $^{-1}$
'*performed by*'	'*recorded by*'
'*performed by*'	'*singer*'
'*performed by*'	'*performance of*' $^{-1}$

Table 4.1.: Example predicate paraphrases.

Note that the number of relaxed queries possible is exponential in the number of triple patterns in the query.

4.5. Answer Ranking

With the XKG and relaxation rules defined, we now present our scoring model that allows for flexible answering of queries with support for relaxation. TriniT's scoring model is based on a language model (LM) for individual triple patterns, the basic building block of a query. The scores obtained for the retrieved triples need to be subsequently aggregated to come up with scores for complete answers to a query. Our setting, with multiple variables and joins is very different from the established setting of scoring keyword queries over textual corpora using language models, including existing work on entity search (Balog et al., 2011). Moreover, our work treats triples obtained from free text using OpenIE as first-class citizens, therefore differing from existing language models for ranking and relaxation in knowledge graphs (Elbassuoni et al., 2009, 2011).

4.5.1. Answers for Individual Triple Patterns

In analogy to the traditional IR setting, we can view a triple pattern as a document which generates individual triples. In this generative setting, we define a language model for each such triple pattern using a mixture model as follows:

$$P(t \mid q_i) = \lambda \frac{\#t}{|q_i|} + (1-\lambda)\frac{\#t}{|XKG|}, \tag{4.1}$$

where $\#t$ denotes the number of occurrences of triple t in the XKG, $|q_i|$ is the total number of triples matching q_i in the XKG. Likewise, $|XKG|$ is the total size (i.e., number of triples) of the XKG and λ is a tunable parameter between 0 and 1. The first term is defined to be non-zero only if t matches q_i. The above defines a proper probability distribution: for each q_i, summing up over all triples in the XKG will always give us 1.

There are some subtle differences between this setting and the traditional IR setting. Smoothing with a background model (the XKG in our case) serves two purposes in traditional IR, namely to avoid zero probabilities and to attain an *idf*-like effect (Zhai and Lafferty, 2004). In our setting, since we only consider triples t that match the triple pattern q_i, zero probabilities are not an issue. Further, a relative weighting of triple patterns, corresponding to the *idf*-like effect, is already obtained by considering triple-pattern selectivities $|q_i|$, that is, how many matching triples exist. For our mixture model, the parameter λ thus controls whether the probabilities $P(t|q_i)$ are only based on the number of occurrences of t (for $\lambda = 0$) or also consider triple-pattern selectivities (for $\lambda > 0$), resulting in a relative weighting of triple patterns in the query.

4.5.2. Answers for Entire Queries

With the score for an answer to a single triple pattern defined, we now move on to defining the score for an answer a for a composite query Q:

$$score(a, Q) = \prod_{q_i \in Q} P(a(q_i)|q_i), \tag{4.2}$$

where $a(q_i)$ is the triple resulting from applying the answer a to the triple pattern q_i, as defined earlier.

Multiple answers can produce the same projected answer a_P, which is the final result the user is interested in. The query may contain, for example, variables corresponding to a movie, songs in its soundtrack, and their singers, but the user may only be interested in movie-singer pairs, and hence project on the two corresponding variables, with the song being "projected away". This projection can result in duplicate movie-singer pairs.

Thus, for each binding of the projection variables we need to define how to aggregate the scores of the individual results (with bindings for all three variables in the example above) for the whole group of duplicates. While summing up scores seems a natural choice, it incurs two problems: i) inflating the score of answers with frequently occurring entities (e.g., artists with many songs in many movies), and ii) forcing the query processing to retrieve all duplicates for each projected answer as they contribute to the scoring. Especially the second point would be critical from an efficiency perspective and rule out early pruning when scanning posting lists during query processing. Therefore, we opt to use the maximum score for each group rather than the sum over all duplicates, and define the score of a projected answer of a query as

$$score(a_P, Q) = \max_{a : a_P \subseteq a} score(a, Q). \tag{4.3}$$

4.5.3. Scoring with Relaxation

We finally extend our scoring model to account for query relaxation. Starting with a user-provided query, Q, we relax it in one or more steps by applying a sequence of relaxation operators $r_1, ..., r_n$ to obtain $Q' = r_n(...r_1(Q))$. Each relaxation operator r_l carries a relaxation weight $w_l \in [0, 1]$ which reflects how much a relaxed query drifts away from the previous query.

The score of an answer obtained from a relaxation is defined as:

$$score(a, Q, Q') = \prod_{l=1}^{n} w_l \times score(a, Q'). \tag{4.4}$$

Intuitively, the score of a with respect to Q' is attenuated to reflect the divergence of Q' from the original query Q. For the special case where $n = 0$, i.e. $Q' = Q$, with no relaxation operators invoked, $score(a, Q, Q') = score(a, Q)$. It is important to note that the relaxation weight applies to the entire relaxation, and not to each triple pattern contained in it.

Because it is possible to generate the same answer through multiple distinct relaxations, we define the score of an answer with respect to the original query and the space of all possible relaxations $\mathbf{R}$ as:

$$score(a, Q, \mathbf{R}) = \max_{\substack{Q' = r_n(...r_1(Q)),\\ r_l \in \mathbf{R}}} score(a, Q, Q'). \tag{4.5}$$

Here, an answer is assigned the maximum score it can obtain from any of the possible relaxations of the query (including the original query). The rationale for this design decision is analogous to the above score aggregation over duplicates when variables are projected away and in line with Theobald et al. (2005): we want to avoid unduly inflating the influence of seeing the same answer many times in different contexts and allow for early pruning in query processing.

Finally, the score of a projected answer in a setting with relaxation is defined as the maximum score of the projected answer obtained through any relaxation.

$$score(a_P, Q, \mathbf{R}) = \max_{a: a_P \subseteq a} score(a, Q, \mathbf{R}). \tag{4.6}$$

We next discuss how we can efficiently evaluate queries in this setting, where our main concern will be to find the top-k projected answers for a query while exploring as little as possible of the relaxation space.

4.6. Query Processing

Query processing in TriniT is a natural fit for the top-k paradigm (Fagin et al., 2003; Ilyas et al., 2008) where the goal is to produce the k projected answers with highest scores, without necessarily computing the complete set of answers. This approach has been shown effective in similar IR settings (Anh and Moffat, 2006; Theobald et al., 2005) involving query relaxation and rewriting, where the space of possible queries blows up rather quickly.

Our query processing scheme is an adaptation of that of Ilyas et al. (2003) and Theobald et al. (2005). The first provides the framework for top-k evaluation in a setting with joins, while the second provides the framework for dynamically relaxing a query during query processing using the answer scores and relaxation weights to guide the process and avoid exploring the entire relaxation space. The focus of our work presented in this chapter is on retrieval effectiveness, while allowing for efficient retrieval. In this section we present our implementation the TriniT, but leave comprehensive evaluation of this aspect of TriniT to future work.

Basic Top-k Query Processing

A top-k query processing scheme applies to a setting where the score of an answer (projected query answer in our setting) is computed from aggregating the scores of its constituents (triple pattern answers in our setting). As stated above, the goal is to compute the set of k answers with the highest scores without necessarily computing all possible answers. To achieve this, a top-k query processing scheme maintains an upper bound on the scores of uncomputed answers, and can therefore stop whenever this bound is lower than the scores of the answers already computed. To be able to establish such a bound, two restrictions need to apply to the query processing setting:

1. the function used to aggregate triple pattern answer scores into a single (projected) query answer score must be monotonic, and

2. for a given triple pattern, its answers must be retrieved in descending order of their scores.

Our answer scoring scheme satisfies the first requirement. Equation 4.2 used to aggregate the scores of triple pattern answers into a single score for a query answer is monotonic. Moreover, by choosing *max* to aggregate scores of equivalent answers and projected answers, we relieve ourselves from the need to generate all possible answers of a triple pattern query.

As for the second requirement, by retrieving triple pattern answers in descending order of their scores, we are always able to establish an upper bound on the scores of unseen triple pattern answers, and therefore an upper bound on the aggregation of these scores (as the aggregation function is monotonic).

In Algorithm 4.1, we present the Hash Rank Join algorithm for computing top-k join results. We adapt it for our setting from Ilyas et al. (2003). For simplicity, we describe the algorithm for the case of two triple patterns. The input to the algorithm is the two triple patterns, a left one q_L and a right one q_R, and for each of the two a list of its matching triples in the XKG in descending order of their scores $p(t|q)$, and k, the number of desired answers.

The priority queue of answers PQ allows access to already produced answers, with their priority determined by their score. The two offsets keep track of the next position to read from in the answers lists of the two triple patterns. *top* and *bottom* keep track of the upper and lower bounds of the scores of the observed answers for a specific triple pattern, respectively. *top* is assigned only once, when the first answer of a triple pattern is read (lines 18–19), while *bottom* is assigned each time a new answer for the corresponding triple pattern is read (line 20), and its value is therefore monotonically non-increasing.

The outer loop in the algorithm (lines 7–28) runs until k answers have been obtained or both triple pattern answer lists run out (lines 12–13), in which case $l < k$ answers will be returned. The second inner loop (lines 14–28) is responsible for producing new join results and updating the threshold, which is an upper bound on yet-unseen join results. Each iteration of this loop reads from one of the two inputs L or R, updates the corresponding score bounds, and computes a threshold based on these bounds (line 22).

The intuition behind the threshold computation is as follows. At any point, the threshold should give an upper bound on the score of yet unseen join results (query answers). If we fix one of the inputs, say L, then we know that the maximum score achievable by a yet-unseen join result induced by reading the next triple pattern answer from q_L is $aggr(bottom_L, top_R)$, since this yet-unseen answer can potentially come from joining the next triple pattern from L with the highest scoring triple pattern answer from q_R. If we now decide to fix R, then the argument is analogous, resulting in the threshold computation on line 22.

On line 23, a hash table for the triple pattern under consideration is updated with the triple pattern answer, and this same answer is used to probe the hash table of the other triple pattern, with the join results produced placed in the priority queue for consumption by the first inner loop.

The first inner loop simply queries the priority queue for answers with scores higher than the threshold until the answer list has k answers or the priority queue has no more such answers to offer.

Adding Relaxation

We add relaxation to the basic scheme for obtaining the top-k join results following the approach of Theobald et al. (2005). Relaxation is guided by the scoring scheme, specifically Equations 4.5 and 4.6, where we are interested in the distinct top-k projected answers. By choosing *max* to aggregate the scores of an answer across the entire query relaxation space, it is sufficient to access answers of triple patterns and their relaxations in descending order of their contribution to the score of a query answer. The *incremental merge* approach of Theobald et al. (2005) allows us to do just this.

Conceptually, incremental merge coalesces multiple score-sorted lists of answers into a single score-sorted list. In practice, however, this merging is not done in one shot, but incrementally to allow for more efficient query processing, hence the name. We formally describe the incremental merge operation below when we discuss the incremental merge operator as part of the discussion of the implementation of TriniT.

Implementation

We implement the scheme described above using three database-style operators organized into a tree. Each operator supports three methods: *Open*, which performs any initialization needed, *GetNext* which returns a scored answer each time it is called, and *Close* which performs any necessary cleanup. An operator is specified by its inputs and the specification of the output of it's *GetNext* function. These operators are organized in a tree, called a query plan, where each operator serves as input to its parent operator. Figure 4.4 shows an example query using all three operators we need in our setting: *scan* operators, *incremental merge* operators, and *rank join* operators. We describe each of the three operators starting from the scans which appear as leafs of the operator tree:

Scan Operator: Each triple pattern in the query or a relaxation requires a scan. Successive calls to a scan's *GetNext* return triple pattern answers (see Definition 4.2) in descending order of their score, $P(a(q)|q)$.

Algorithm 4.1: Hash Rank Join Algorithm.

```
input : $q_L, q_R$: two triple patterns,
        $M_L, M_R$: triples matching $q_L$ and $q_R$ respectively
        in descending order of $P(t|q)$,
        $k$ the size of the output list
// Initialization
1  $PQ$ is an empty priority queue;
2  $A$ is an empty answer list;
3  $H_L, H_R$ are empty hash tables;
4  $offset_L, offset_R := 0$;
5  $top_L, top_R := \infty$;
6  $bottom_L, bottom_R := \infty$;
7  $threshold := \infty$;
// Processing
8  while $true$ do
9      while $size(A) < k \wedge PQ$ $not$ $empty \wedge PQ.top.score \geq threshold$ do
10         tuple = $PQ$.top;
11         remove tuple from $PQ$;
12         A.add(tuple);
13     if $size(A) = k \vee (offset_L = |M_L| \wedge offset_R = |M_R|$ ) then
14         break ; // Finished processing
15     while $true$ do
16         $next$ is one of $L$ or $R$;
17         tuple = $M_{next}[offset_{next}]$;
18         $offset_{next} = offset_{next} + 1$;
19         if $offset_{next} = 0$ then
20             $top_{next}$ = tuple.score;
21         $bottom_{next}$ = tuple.score;
           // $agg$ is the score aggregation function of Equation 4.2
22         $threshold = max(agg(top_L, bottom_R), agg(bottom_L, top_R))$;
23         insert tuple in $H_{next}$;
24         probe other hash table with tuple;
25         foreach $valid$ $join$ $combination$ do
26             compute the score of the join result;
27             insert the result in $PQ$;
28         if $PQ$ $not$ $empty$ then
29             break;
30 return $\mathcal{A}$
```

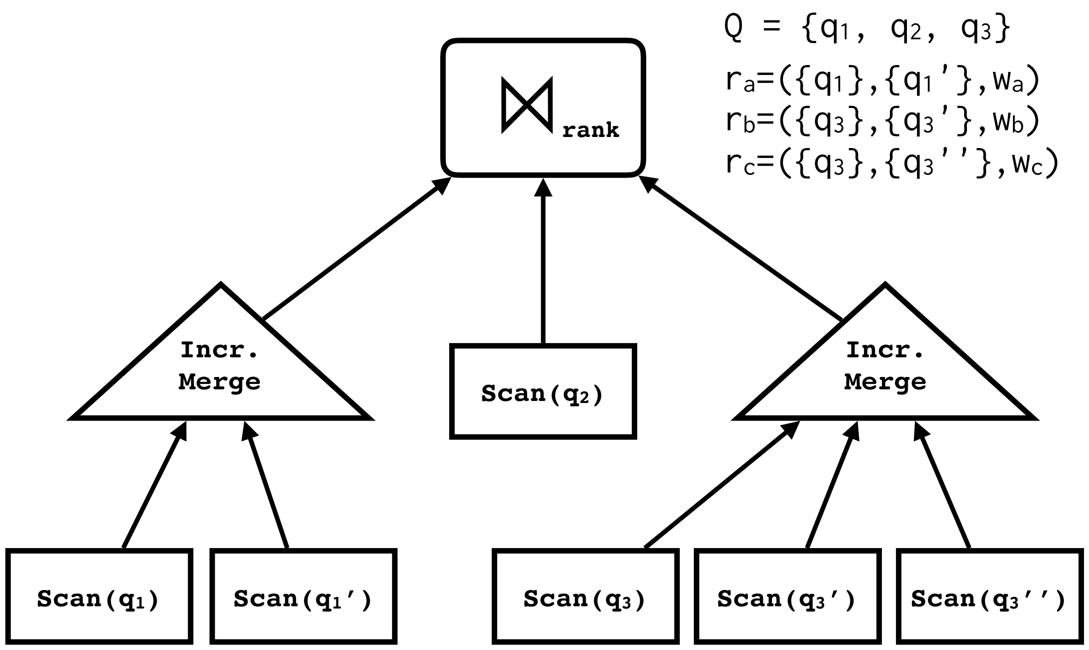


Figure 4.4.: Example query plan with relaxation.

Incremental Merge Operator: Conceptually, the role of an incremental merge operator is to coalesce multiple sorted lists of scored answers, each provided by a different input operator, into a single sorted list of scored answers. Moreover, the operator allows for providing for an attenuation factor $w \in [0, 1]$ for each of its input operators. The score answer retrieved from the corresponding operator is attenuated by a factor of w. This operator is adapted from Theobald et al. (2005), to account for the different nature of answers in our setting (variable bindings to XKG tokens rather than single-document ids).

More concretely, the input to an incremental merge operator is a list of operators $(OP_0, ..., OP_l)$ that each produce answers in descending order of their scores and a list of relaxation weights $(w_0, ..., w_l)$. Conceptually, let M_i be the full list of answers returned by OP_i and $offset_i$ is the current position in that list initialized with 0, the offset of the first answer with the highest score. Each call to an incremental merge operator's $GetNext$ will:

1. return the answer $a = M_i[offset_i]$ where

$$i = \arg\max_{i' \in [0,..,l]} w_{i'} \times score(M_{i'}[offset_{i'}])$$

   assigned a score of $w_i \times score(M_i[offset_i])$, and

2. increment $offset_i$ by 1.

In our setting, we construct an incremental merge operator for each set of triple patterns $\mathbf{q} \subseteq Q$ that occurs in the domain of a relaxation rule $r = (\mathbf{q}, \mathbf{q}', w)$. In the relaxation rules we use in our experiments (see Section 4.4), we only deal with $|\mathbf{q}| = 1$. The list of input operators to the incremental merge operator is constituted of operators corresponding to $\mathbf{q}$ (a scan since we restrict ourselves to $\mathbf{q} = 1$) and one operator for each of its relaxations, with the weights provided accordingly. In our setting, this is a scan if $|\mathbf{q}'| = 1$ and a rank join operator (see below) if $\mathbf{q}' \geq 2$ (i.e., a structural relaxation).

Rank Join Operator: This is where answers from the different triple patterns and their relaxations are joined to form the actual answers to the original query. We follow the provably optimal approach of Ilyas et al. (2003) which guarantees early termination when the aggregation function used to combine the incoming triples is monotonic (as in our case).

We adopt the Hash Rank Join (HRJN) as a physical operator in our implementation. This operator's $GetNext$ function is a natural extension of Algorithm 4.1, where instead of adding a tuple to the answer list to accumulate the top-k answers we return this answer so that the next operator can consume it. In this manner, successive calls to $GetNext$ return answers in descending order of their scores. Rank joins are used to produce the final answer of the query as well as for structural relaxations where the domain of the relaxation has multiple triple patterns.

4.7. Experimental Evaluation

We now present experimental results to demonstrate the effectiveness of query answering in TriniT.

4.7.1. Methods

We compare two different configurations of TriniT with three natural state-of-the-art baselines. Figure 4.5(c) shows an example query for each system. The first TriniT configuration (TriniT-Relax) processes queries without considering relaxations. We contrast this to TriniT+Relax where TriniT has access to relaxations that it can automatically invoke as needed during query processing.

The first baseline we consider (ES) is based on the work by Balog et al. (2011) for entity search. Here, an entity is represented by two fields containing the semantic types it belongs to and a textual description. We use Model 4, which is the most effective one that allows us to enforce type constraints that are crucial for good results. The ES approach cannot return tuples in response to relationship queries, so we formulate queries to ask about a single entity (a single variable).

The second baseline (ERS) is that by Li et al. (2012) for entity-relationship search.

Here, queries are evaluated over an entity-annotated corpus. The score of a match of a query condition (a textual description of a typed variable or textual relation connecting two such variables) depends on the proximity of phrases in the condition and variable bindings.

Finally, we compare TriniT with SPOX(SPO+teXt), an extended version of the approach by Elbassuoni et al. (2009). Each SPO triple pattern in the KG is associated with textual keywords taken from the context in which this fact is expressed. For example, the triple `RusselCrowe actedIn ABeautifulMind` would be associated with the set of keywords {'*john*', '*nash*', '*true*', '*story*', '*princeton*', '*australia*' ...}. On the query side, SPO triple patterns can optionally come with teXtual conditions that specify (soft) constraints which cannot be expressed in structured SPO form over the KG. For example, the SPOX pattern `?x actedIn ?y['true story']` is used to search for actors in a movie based on a true story. During query answering, the SPO parts of the query are first matched against the KG, and the answers are subsequently ranked using language models that consider witness counts of KG triples as well teXtual keyword conditions with their frequencies. The original SPOX approach includes a form of relaxation in which entities or relations in a query are replaced by variables. We improve on this by moving these entities or relations to the X component of a triple instead of completely discarding them. This way they can still influence the final ranking of answers. We report results from this improved approach. We do not consider Elbassuoni et al. (2011) as a baseline since relaxations here are performed by replacing KG entities and relations with other KG relations, which almost always results in semantic drift, as KGs rarely contain redundancy.

4.7.2. Benchmarks

Existing entity-search queries tend not to be relationship-centric. A contribution of this work is a new set of 70 inherently relationship-centric information needs, referred to as COMPLEX queries here (e.g., *"Programming languages invented by people who won the Turing Award."*). We next describe how these queries were generated. A query was constructed starting from a *chain* of entities, for example:

`[ALGOL -- JohnBackus -- TuringAward]`,

where some entities become part of the query and others serve as an answer. These chains are automatically sampled from domains within the XKG, where a domain is the set of entities that fall within a specified set of semantic types. The domains we consider are cinema, music, books, sports, computing, and military conflicts. The cinema domain, for example, includes entities of the types `actor`, `show`, `director`, `award`, and `producer`. Within each domain, we iteratively sample entities starting from a pivot entity to form a chain. The first pivot entity (`ALGOL` in our example) is sampled non-uniformly based on a popularity prior from the domain. Next, we find

`[ALGOL--JohnBackus--TuringAward]`

(a)

"Programming languages invented by a Turing Award winner."

(b)

TriniT:

```
SELECT ?x ?y WHERE {
 ?x type programming_language . ?y type person .
 ?y "invented" ?x . ?y won TuringAward }
```

ES:

```
type:(programming_language)
text:(programming language invented by a turing award winner)
```

ERS:

```
SELECT ?x ?y
FROM programming_language ?x, person ?y
WHERE ?x:["won", "turing award"] AND ?x,?y:["invented"]
```

SPOX:

```
SELECT ?x ?y WHERE {
 ?x type programming_language .   ?y type person .
 ?x ?r ?y ["invented"] . ?y won TuringAward }
```

(c)

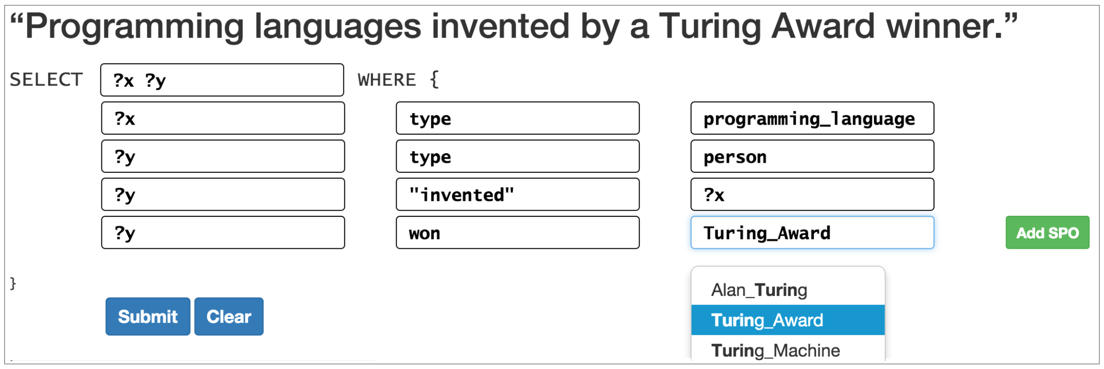
Figure 4.5.: Generation of COMPLEX queries: (a) an example chain used to generate (b) a COMPLEX question and (c) the corresponding queries for the various systems under consideration.

the 20 entities in the domain that have the highest coherence with the current pivot by the Wikipedia-link measure of Milne and Witten (2008). We then sample these entities non-uniformly, based on the number of XKG facts connecting them to the pivot to choose the next pivot. This process is repeated to obtain chains of size 2-4 (determined randomly). A human annotator then constructs a question from the chain asking for the first entity while containing multiple unknowns corresponding to other entities in the chain. An annotator can discard a chain if she thinks no interesting question can be generated from it. Figure 4.5(a) shows an example of a chain, and (b) shows the corresponding question formulated by the human annotator. While the question is constructed by considering a single chain, it can have many answers. In the example of Figure 4.5, `(Pascal,NiklausWirth)` and `(Smalltalk,AlanKay)` are two possible answers among several others. Table B.3 shows all 70 COMPLEX queries.

We additionally ran experiments with two benchmarks from previous work. The first, ESQ, is a set of 485 entity-centric queries compiled by Balog and Neumayer (2013). We remove from this dataset SemSearch ES and INEX LD queries as they do not fit our setting. SemSearch ES contains queries such as *"YMCA Tampa"* and *"nokia e73"*, which refer to a specific entity with no relations at all. INEX LD, is highly keyword-centric (e.g., *"allegedly caused World War I"*) with a very noisy gold standard (e.g., `Aerial_bombing_of_cities` is considered a relevant entity for the above query). This leaves us with 255 queries from which we remove 37 involving aggregation (e.g., *"Books by William Goldman with more than 300 pages"*, *"movies with eight or more Academy Awards"*) as these are beyond the capability of all the systems in this experiment, leaving us with 218 queries. Unlike our COMPLEX queries, ESQ queries ask for individual entities rather than tuples, and are usually expressed in the form of a type (e.g., *"EU countries"*) or a type with a description that contains a single relation (e.g., *"movies directed by Francis Ford Coppola"*). The full list of ESQ queries used in our experiments is shown in Table B.1.

The other benchmark from prior work, which we call ERQ, is constructed by Li et al. (2012) and consists of 28 queries. 22 of the queries in this dataset are similar to the ones in ESQ, with 6 only asking for pairs of entities. The ERQ queries are shown in Table B.2.

System Input Generation: TriniT and each of the three baselines described in Section 4.7.1 expects a specific form of query as shown in Figure 4.5(c). For our experiments, it is important that we generate these queries systematically to facilitate a fair comparison of the approaches. We create these queries in a two-step process. In the first step, an information need in the form of a question (Figure 4.5(b)) is shown to a human annotator who is asked to translate it into a proto-query. In the second step, the proto-query is used to automatically create queries for the four systems using a set of rules described below. The annotator, after being shown four examples of


"Programming languages invented by a Turing Award winner."

```
SELECT ?x ?y WHERE {
?x    type         programming_language
?y    type         person
?y    "invented"   ?x
?y    won          Turing_Award
}
```

Figure 4.6.: Proto-query interface, with input for the question in Figure 4.5(a).

question-to-proto-query translation, is presented with the UI shown in Figure 4.6. Here, each row provides SPO fields for specifying SPO triple patterns. Each field provides auto-completion functionality for KG entities (S and O) and predicates (P). The annotator is asked to express the given question in SPO form, and is instructed to use the auto-completion suggestions when appropriate, or resort to textual tokens if necessary.

For ES, the type field is filled with the type associated with the first variable in the proto-query, and the text of the question is used in the text field. For ERS and SPOX, variable type-constraints are maintained. ERS cannot deal with KG predicates in the query (it only returns entity tuples as results), so they are mapped to their textual form. SPOX can only deal with entities and predicates in the S and P components, respectively, but not textual tokens. To accommodate this, we extend an SPO triple pattern in the proto-query with an X component and move the S/O textual component there – note that this is not the same as SPOX relaxation described above, which is part of query processing. In real life, we envision a system used by professionals such as journalists and researchers willing to invest some learning effort in exchange for the more expressive querying they get in return from the different types of queries, with support from appropriate UIs.

4.7.3. Data

We finally describe the data we use for TriniT and the various baselines we consider. We note that we run our own implementations of the baselines on a scale two orders of magnitude larger than what was previously reported.

We use as our KG Yago2s (`yago-knowledge.org`), whose predicates connect entities from Wikipedia to other entities (e.g., `TomHanks actedIn ForrestGump`), to literals (e.g., `TomHanks birthDate 1956-7-9`), or to semantic types (e.g., `TomHanks type actor`), with a total of 48M triples (44M class assignments, and 4.4M relations and attributes).

As a text corpus, we use the FACC1 dataset, which annotates text spans in the ClueWeb'09 corpus (`lemurproject.org/clueweb09/`) with entities linked to Wikipedia entities (via Freebase) with a precision and recall estimated to be 80-85% and 70-85%, respectively.

For TriniT, we construct the XKG by combining the KG described above with the result of a simple yet effective open information extraction scheme over the annotated ClueWeb'09 corpus as follows. We look for pairs of entity annotations in the same sentence separated by a string of at most 50 characters, and create a triple where the two entities are the subject and object, and the separating string is the relation connecting them. In this way we obtain 392M extractions, resulting in 65M unique triples. By sampling, we estimate the accuracy of the XKG to be around 70-80%. For predicate paraphrases, we run the predicate paraphrasing scheme described in Section

4.4.2 on top of the XKG to generate 172M pairs of scored predicate paraphrases like those in Table 4.1.

Originally, ERS was designed for entity-relationship search over Wikipedia. While the original paper comes with a demo (`idir.uta.edu/erq/`), we could not use it to evaluate our queries as it is restricted to a subset of Wikipedia as a corpus annotated with entities from 10 semantic types. We therefore use the annotated ClueWeb'09 corpus for ERS as we did for TriniT. In addition, ERS takes type associations from our KG, ensuring ERS has data comparable to that used by TriniT. We attempted to evaluate the three benchmarks on the ERS online demo (`idir.uta.edu/erq/`). However, it contained only a subset of the entities and types needed to answer COMPLEX and ES queries, preventing a fair comparison with other systems. For SPOX, we use the annotated ClueWeb'09 corpus to associate keywords with entities and subsequently KG triples. To do this we find all words that occur in the same sentence as an entity and keep those with a positive association (using normalized PMI) with the entity. SPOX uses the same KG as TriniT, Yago2s.

In ES we use an entity's Wikipedia page to populate its textual description, in line with Balog et al. (2011). We tried to extend this by adding to this field sentences from the annotated ClueWeb'09 corpus that mention the entity. However, this resulted in worse retrieval effectiveness. We populate the semantic type field for an entity from the types it is associated with in the KG.

4.7.4. Results and Analysis

Table 4.2 shows our experimental results. Following earlier work, we use precision@5, NDCG@5 (with binary relevance), and recall as quality measures (see Section 3.7.3 for the definitions of these quality measures). For queries with an empty results list, we define all measures to be 0.

For ESQ queries, we use the gold standard provided with the benchmark. For ERQ and COMPLEX no gold standard is given, so we crowdsource relevance judgments and determine the relevance of an answer by a majority vote of three judges. Note that since the XKG contains noise, human annotators were instructed to base their judgments on the real world rather than on the XKG. Inter-annotator agreement was measured using Fleiss' kappa to be 0.837 indicating almost perfect agreement. In all cases we use binary relevance.

ERQ and COMPLEX queries do not easily lend themselves to computing the complete set of relevant results, with queries such as *"NBA teams married to actresses, and the teams they play for"* (COMPLEX) or *"people born in Spain"* (ERQ). To compute NDCG and recall, we use pooling from the various systems to create a golden standard. Since ES returns single entities and not tuples, we project all answers on the same dimension as the one used for the ES query and compute the golden standard over that dimension.

“Spouses of actors who graduated from an Ivy League university.”

Query:

```
SELECT ?x ?y ?z WHERE { ?x type person .
   ?y type actor .  ?z type university .
   ?y graduatedFrom ?z .  ?x marriedTo ?y .
   ?z "member of" IvyLeague }
```

XKG:

```
ChristopherReeve graduatedFrom JuilliardSchool
ChristopherReeve "went to"     CornellUniversity
...
```

Relaxations:

```
 (?w graduatedFrom ?z)→(?w "went to" ?z):  0.066
```

Relevant answers:

-TriniT-Relax: ϕ

-TriniT+Relax:

{(DanaReeve, ChristopherReeve, CornellUniversity)}

Figure 4.7.: Anecdotal example of results - I.

"Lieutenant governors of the province where Ottawa is located."

Query:

```
SELECT ?x ?y WHERE {?x type province .
   ?y type person .  Ottawa locatedIn ?y .
   ?x "lieutenant governor of" ?y }
```

XKG:

```
Ottawa locatedIn NationalCapitalRegion
NationalCapitalRegion locatedIn Ontario
Ontario "lieutenant governor"   DavidOnley
HilaryWeston "lieutenant governor of"   Ontario
...
```

Relaxations:

```
 (?w locatedIn ?z)→(?w locatedIn ?u .  ?u locatedIn ?z):  1.0
 (?w locatedIn ?z) → (?w "part of" ?z):  0.073
 (?w "lieutenant governor of" ?z)→
      (?z "lieutenant governor" ?w):  0.259
```

Relevant answers:

-TriniT-Relax: ϕ

-TriniT+Relax: {`(DavidOnley, Ontario)`, `(HilaryWeston, Ontario)`}

Figure 4.8.: Anecdotal example of results - II.

	ESQ (218)		
	P@5	**NDCG@5**	**R**
ES	0.183	0.211	0.093
ERS	0.182	0.232	0.119
SPOX	**0.249**	**0.336**$^{\triangle}$	**0.188**
TriniT-Relax	0.158	0.192	0.093
TriniT+Relax	0.218	0.287	0.156

	ERQ (28)		
	P@5	**NDCG@5**	**R**
ES	0.492	0.489	0.236
ERS	0.408	0.387	0.177
SPOX	0.577	0.580	0.248
TriniT-Relax	0.467	0.502	0.174
TriniT+Relax	**0.637**	**0.692**	**0.267**

	COMPLEX (70)		
	P@5	**NDCG@5**	**R**
ES	0.132	0.172	0.115
ERS	0.249	0.322	0.234
SPOX	0.243	0.250	0.134
TriniT-Relax	0.370	0.419	0.258
TriniT+Relax	**0.603**$^{\triangle}$	**0.775**$^{\triangle}$	**0.613**$^{\triangle}$

Table 4.2.: Experimental results.

Figures 4.7 and 4.8 give illustrative examples from our COMPLEX queries set, with the relaxations invoked to answer them, and the relevant results produced. We discuss the results by benchmark next.

ESQ Queries: Here SPOX outperforms all systems. On these non-relationship-centric queries, SPOX boils down to an improved version of ES. If a query can be formulated in a structured manner, then this tends to be reflected in SPOX query formulations, resulting in SPO only queries, without the X components. When this is not possible, most of the query conditions end up in the X component of a type-constraint triple pattern either by formulation or through the improved SPOX relaxation scheme we described above.

TriniT is penalized against SPOX on queries that can be satisfactorily answered

$^{\triangle}$ Significant improvement (two-tailed paired t-test, $p < 0.01$).

with keywords without the need to establish crisp relationships, as TriniT requires. For example, the query *"Nordic authors known for children's literature"* is reduced in the SPOX model to `?x type author[`*`'nordic children's literature'`*`]`. Here, looking at the co-occurrences of an author and the keywords suffices to return good answers. On the other hand, the SPO query formulation used for TriniT, with P set to '*known for*', could not find matches in the XKG, even with relaxation. This result demonstrates that keyword based querying is an effective paradigm for a certain class of queries. Once we move to more relationship-centric queries below, we start observing the advantage of TriniT's approach. TriniT could in principle define a special predicate `hasKeywords` that allows the association of entities with keywords when a crisp relation cannot be formulated (e.g., `?x hasKeyword` *`'children's literature'`*). In this work we are interested in truly relationship-centric queries, so we leave this for future work.

We can already observe the advantages of relaxation at this stage. For example, on a simple query asking for *"Italian Nobel winners"*, the TriniT query uses the KG predicate `won`. While this looks reasonable, the KG only lists winners of specific Nobel prizes (e.g., `NobelPrizeInLiterature`). Only by relaxing `won` to the inverse textual predicate, '*winner*'$^{-1}$, is the TriniT query able to return the correct answers from triples like `(NobelPrize` *`'winner'`* `EnricoFermi)`.

It is also interesting to observe the relatively close performance of ES and ERS. For most ESQ queries, ERS will reduce to ES, with a type constraint and a set of keywords describing the target entity, but with different scoring schemes.

ERQ Queries vary in how relationship-centric they are. The majority of ERQ queries (22/28) ask about a single entity, not a tuple, requiring no joins. The rest, while they ask for a tuple, can all conceivably be answered from an individual document describing a relevant entity. ERQ queries like *"films starring Robert de Niro, and their directors"* and *"Novels and their Academy Award winning film adaptations"* issued to the ES system in search for movies/novels, respectively, return satisfactory results. TriniT, when answering such questions is able to return tuples with an explanation of the precise relations that hold between the various entities in a tuple (either those part of the user's query, or matched through relaxation).

Seemingly simple ERQ queries which ask for a list of individual entities, not tuples, can be easily mishandled due to a lack of relation awareness. Results from ES and ERS for the query *"football players who were FIFA Player of the Year"* include the entities `DavidBeckham` and `ThierryHenry`, both of whom were runners-up for the award, but never actually received it. TriniT is able to handle this query correctly.

For this class of queries SPOX starts to suffer when the desired relationships connecting the two variables are either not sufficiently populated or unavailable in the KG.

COMPLEX Queries are the most interesting: they require combining factual knowledge from multiple sources and establishing the existence of the relation specified in the query. TriniT with relaxations significantly outperforms all other systems on this dataset. Figure 4.7 helps understand why. It would be rare to find documents where keyword matching would correctly answer this query. Even then, we cannot expect to obtain a complete list of results. This query is inherently relationship-centric, a perfect fit for TriniT. The original TriniT query contains one textual XKG predicate '*member of*', as no KG relation covers it. However, the KG relation `graduatedFrom` lacks sufficient coverage, as the fact `(ChristopherReeve graduatedFrom CornellUniversity)` is missing. The XKG compensates for this through the textual relation '*went to*' (see Table 4.1), which is exploited by an automatically-invoked relaxation to facilitate the return of the relevant answer shown. For ES, this query, like most others in the benchmark, is too challenging: the evidence needed to answer is multiple hops away from a relevant entity. As in earlier examples, ERS can be brittle when establishing the existence of a relation in the query is critical for answering it. ERS scores entities in an answer based on their proximity to each other and to query terms expressing relations that must hold. This can be detrimental due to the complexities of natural language: e.g., *"private and public universities including Ivy League members, MIT, VanderbiltUniversity, SwarthmoreCollege , CalBerkeley..."* is incorrectly taken as evidence of `SwarthmoreCollege`'s membership in the `IvyLeague`. Figure 4.8 shows an example of a COMPLEX query where multiple relaxations are triggered, including a structural relaxation of the spatial `locatedIn` predicate, a predicate paraphrase and inverse paraphrase. On this query, SPOX fails as it contains no relation connecting a pair of entities that can serve as correct bindings of `?x` and `?y`, both of which have to be projected to the user. In this case, SPOX's relaxation scheme is not helpful. Here, ERS turns out to be more effective than SPOX as it relies on a less rigid scheme for answering queries.

4.8. Discussion

We have presented TriniT, a framework for the evaluation relationship queries over knowledge graphs in a flexible manner that automatically compensates for mismatches between a query and the underlying KG, possibly due to incompleteness in the KG. TriniT extends the basic KG model to allow for facts extracted from texts, with textual tokens possible in any of their SPO components. It automatically takes care of mismatches between the query and the data, including the extended knowledge graph, by means of automatically mined weighted relaxation rules which are automatically invoked at query processing time. TriniT's scheme for scoring answers ensures that answers to the original query are preferred over those generated from relaxations unless there is strong evidence for the latter. The scoring scheme also allows for efficient top-k query processing, thereby avoiding the need to fully explore the potentially huge space of query relaxations.

In contrast to earlier work on query relaxation in the knowledge graph setting, TriniT does not sacrifice the structure of the query when performing relaxation. This makes sure that the expressiveness of the query is not lost in the process of relaxation.

We presented an experimental evaluation to study the effectiveness of TriniT in comparison with several natural baselines using established benchmarks. Moreover, we created our own benchmark of relationship-centric to demonstrate the effectiveness of TriniT on this challenging and interesting class of queries. On relationship-centric queries, TriniT+Relax outperforms the baselines by a clear margin. This most pronounced for the COMPLEX benchmark with relationship-centric queries – the very point that our research aims to address. Here the gains are high in all metrics.

It is also important to understand the limitations of TriniT and why it fails on some queries. Simpler entity queries can often be answered satisfactorily using traditional keyword-based approaches, as we have seen for SPOX over ESQ. Here, TriniT's relationship-centric approach is crucial only when keyword matches are misleading and crisp relations must be established. For TriniT+Relax, we observe that losses are mostly due to incorrect XKG facts and semantic drift in the relaxations. Such incorrect facts arise from incorrect entity annotations, or shortcomings in the extraction scheme. Generally, incorrect facts matching a triple pattern are less frequent than correct ones, which also means that the answers obtained through these have smaller weights. This is usually a problem for queries with a small number of expected correct answers, where the correct answers rank highest, and incorrect ones are at the bottom of the list.

The second source of incorrect answers is relaxations that drift from the original query intention. Again, this is mostly an issue in queries where few correct answers are expected compared to the number of answers actually returned, and can often be pruned away by observing a sudden drop in answer scores. This is why the scoring scheme presented in Section 4.5 uses *max* rather than *sum* for score aggregation.

The focus of this work has been on effectiveness, while having a framework that allows for efficient performance. A comprehensive efficiency evaluation as well as a more in-depth investigation of the relevant systems issues is necessary. In ongoing work we are looking at obtaining good performance for TriniT to allow for interactive querying times. This extension lies at the intersection of IR and database systems. Here, we are developing models for predicting when relaxations are unlikely to be invoked (i.e., the original query can return satisfactory results). These models would drive speculative query plans that resort top-k style processing only when necessary, thereby reducing the overhead of such operators in the case where they are not needed.

We presented two types of relaxation rules in this work. The choice of rules was motivated by the specific benchmarks we are working with. The problem of generating such rules and weighing them also needs further investigation.

5. ReNoun: Fact Extraction for Noun-mediated Relations

5.1. Introduction

5.1.1. Motivation

An important development in the field of information extraction is Open Information Extraction (OpenIE) (Etzioni et al., 2008), where the goal is to extract facts without any specific knowledge about the domain of the documents on which the extraction process is running. We have already seen a very simple scheme for doing this in the previous chapter, where the end goal was to extend a knowledge graph and find paraphrases for its predicates.

OpenIE schemes function by first having a small set of general-purpose patterns over some linguistic representation (e.g., part-of-speech tags or dependency parses) that capture how relations are expressed. When invoked, these patterns generate large numbers of facts, which, in turn, could be used to learn additional patterns. Existing OpenIE schemes make the assumption that nouns represent concepts (entities or unary predicates), and verbs represent relations (binary predicates). This notion can be seen in several natural language processing systems, and is so well established that even the official specification of the RDF turtle syntax grammar calls predicates verbs (Beckett et al., 2014).

In practice, as we show in Section 5.3, this is not the case. Nouns and noun phrases can express very interesting and diverse relations which are rarely found in verb (or verb phrase) form. For example, the noun phrase '*legal affairs correspondent*' is a relation connecting a news outlet with a specific correspondent. What the state-of-the-art in OpenIE is missing is a scheme for extracting instances of such relations from text.

5.1.2. Problem Statement

Given a large textual corpus, our goal is to extract instances of binary relations mediated through noun phrases from the long tail with high recall, following the principles of OpenIE. Each instance has the form *(S,R,O)*, where R is the relation, and S and O

are its subject and object arguments, respectively. For our purposes we define a noun phrase to be a phrase headed by a noun and does not start with an auxiliary verb.

Our focus here is exclusively on extracting triples where the relation connecting the subject and object is a noun phrase. Furthermore, we restrict ourselves to relations, whose arguments are concrete entities (see Section 2.1.1) such as '*wife*', '*protege*', and '*chief economist*'. We do not consider attributes, whose arguments are literals (e.g., '*GDP*', '*population*') that are better extracted from (Web) tables (Cafarella et al., 2008) and vague attributes (e.g., '*culture*', '*economy*') whose value is a narrative.

By extracting triples, the facts we produce are in line with the triple-based knowledge graph representation introduced in Chapter 2. However, the problem we deal with here does not involve linking noun phrase relations to existing predicates in a reference KG.

Fat head	daughter, headquarters president, spokesperson,
Long tail	chief economist, defender, philanthropic arm, protege

Table 5.1.: Examples of fat head and long tail relations.

By high recall we mean extracting facts for as many noun phrase mediated relations as possible, and as many correct instances of these as possible. Following OpenIE principles means that we do not need our relations to be defined upfront. Instead, the system discovers new relations as part of the fact extraction process. While the framwork presented in this chapter can discover relations on its own, we resort to working with a predefined set of noun phrases that can serve as relations. These can be collected using the methods described by Gupta et al. (2014), Lee et al. (2013), and Pasca and Durme (2007). The justification is that this allows us to analyze the recall of our approach. In our experiments, we will use Biperpedia (Gupta et al., 2014), a repository of noun phrase relations automatically extracted from Web text and a query log of a major search engine. Since the relations themselves are the result of an extraction algorithm, the may include false positives (i.e., nonsensical relations).

Aiming for high recall means extracting facts in a fairly liberal manner, which will inevitably introduce a great deal of noise. To offset this, fact extraction is followed by a fact scoring stage where correct extractions are given higher scores than incorrect ones, and the consumer of these facts can make use of these scores for their particular application.

We will make an important distinction between the *fat head* and the *long tail* of relations. Our focus in this work will be on the long tail, as existing knowldege graphs and OpenIE schemes perform poorly on it while performing satisfactorily on the fat head as we show later in our experiments. To define the two sets, we order the relations in Biperpedia in descending order of the number of their occurrences in a large textual corpus (see Section 5.7.1). We define the fat head to be the relations until the point N in the ordering such that the total number of occurrences of relations before N equals the number of total occurrences of the relations after N. In our corpus, the fat head has 218 relations (i.e., $N = 218$) and the long tail has 60K relations. Table 5.1 shows examples from both. This distinction will become important when we compare the recall of ReNoun to existing OpenIE schemes.

By working with large Web-scale corpora, OpenIE can exploit redundancy in how facts are expressed both for learning extraction patterns and subsequently scoring extractions. We exploit rich syntactic and linguistic cues by processing the documents in the corpus with a natural language processing pipeline comprised of dependency parsing, noun phrase chunking, named entity recognition, coreference resolution, and entity resolution to Freebase. The chunker identifies noun phrases in the text that include our relations (but the annotation process is independent of our specific setting). All these annotations are exploited in the various stages of our extraction pipeline.

5.1.3. Contributions & Overview

This chapter presents ReNoun, an OpenIE scheme for finding and extracting instances of binary relaions mediated by noun phrases from text. The novel contributions of ReNoun are a bootstrapping scheme for extracting facts centered around noun phrases, and a scoring scheme for facts that benefits from the redundancy in patterns that extract a certain fact.

For fact extraction, ReNoun starts by defining a small set of high-precision patterns for extracting facts centered around noun phrases. These patterns will extract noun phrase relations and instances of these relations at the same time. With these facts in hand, we proceed to looking for sentences where the same facts occur, but expressed in a less direct way using long-range dependencies. These dependencies are extracted and generalized to patterns, which can then be matched against a large corpus to produce even more facts, most of which will not have been observed in the initial seed fact extraction stage.

The second component of ReNoun is a scheme for scoring facts by considering the set of patterns that produces each fact. The fact extraction scheme we outlined above generally aims for high recall in terms of fact extraction. Each stage of the extraction process can introduce noise. Observing that the same fact is usually extracted multiple times by different dependency parse patterns, and that patterns differ in how prone they are to admitting noisy extractions, we design a scheme that scores a fact based on the set of dependency parse patterns from which it was extracted. Our approach essentially quantifies the trustworthiness of each pattern, and then aggregates these numbers to quantify the confidence in an extracted fact. An application consuming the facts extracted by ReNoun can take the score associated with each fact either as a signal in some complex machinery, or simply set a cutoff threshold based on its specific requirements.

Moreover, we present extensive experimental analysis of existing knowledge graphs to motivate the need for accounting for relations expressed through noun phrases and analyze the extractions of existing OpenIE systems to demonstrate their deficiency for this specific case. The scheme described here is already in active use in a major search engine.

We start by presenting related work in the field of information extraction in Section 5.2. In Section 5.3 we back the motivation given above with concrete experimental results showing (i) the use of noun phrases as relations is integral to existing knowledge graphs, and (ii) the inadequacy of existing OpenIE methods for extracting facts centered around noun phrases. In Sections 5.4 and 5.5 we present our approach to seed fact extraction, their use for dependency parse pattern generation, and the deployment of these patterns to generate candidate facts that are passed to our scoring system. Our approach to fact scoring based on extraction pattern coherence is explained in Section 5.6. In Section 5.7, we present extensive experiments and analysis that further demonstrate the need for ReNoun, show the quality of facts from each stage of ReNoun's pipeline, and show places where ReNoun misses.

5.2. Related Work

5.2.1. Open Information Extraction

Open Information Extraction (OpenIE) was introduced by Etzioni et al. (2004) as an information extraction paradigm designed to allow for domain-independent discovery of relations and extraction of their instances from Web-scale text corpora in a scalable manner. This is in contrast to systems that are domain specific and are designed to extract instances of a predetermined set of relations. The first OpenIE framework, called KnowItAll, allowed for the manual specification of extraction rules in a domain-independent manner. These rules would subsequently trigger Web search engine queries both to find documents against which these rules can be matched and to automatically assess the quality of the resulting extractions. Later work introduced TextRunner (Banko et al., 2007; Yates et al., 2007), an OpenIE system that finds relations by looking for the text separating two noun phrases in a sentence and uses a classifier to judge the trustworthiness of an extraction. WOE^{parse} (Wu and Weld, 2010) extends this work by using dependency parsing to connect the subject and object. Both systems assume that the relation is located between its arguments, an assumption that ReNoun drops as it is unsuitable for noun phrase relations.

Closest to our work are ReVerb (Fader et al., 2011) and OLLIE (Mausam et al., 2012). ReVerb uses POS tag patterns to locate relations mediated by verbs, and then looks for noun phrases to the left and right of the relation for arguments. OLLIE uses ReVerb extractions as its seeds to train dependency parse patterns that can extract more triples. While OLLIE's patterns themselves are not limited to verb relations, the ReVerb seeds are limited to verbs, which makes OLLIE's coverage on noun relations also limited. In contrast, ReNoun takes a noun-centric approach from the start and extracts many that do not exist in OLLIE.

ClausIE (Corro and Gemulla, 2013) is an OpenIE framework that exploits knowledge about English grammar and linguistic structure to find clauses in a sentence using its dependency parse. The clauses are subsequently used to generate extractions at multiple levels of granularity, possibly with more than triples. While ClausIE comes with a predefined set of rules on how to extract facts from a dependency parse, ReNoun learns such rules from its seed facts (albeit for extractions composed of triples only).

Similar to ClausIE, Angeli et al. (2015) performs OpenIE relying on the clause structure of sentences. It subsequently uses natural logic (Lakoff, 1972), a proof system based on the syntax of natural language, to simplify extractions so that they are not overly specific, and hence more useful in downstream applications, while still entailed by the original sentence.

5.2.2. Bootstrapping and Distant Supervision

Bootstrapping, as used in this work, goes back to some of the early approaches to information extraction like DIPRE (Brin, 1998) and Snowball (Agichtein and Gravano, 2000) among others. Bootstrapping is achieved by manually compiling seed tuples or patterns, and then using these to learn more patterns. The process can proceed in an iterative manner, with each new set of facts contributing to finding more patterns. Compared to these works, our patterns are linguistically motivated and are not relation specific. DARE (Xu et al., 2007) is a framework for extracting instances of n-ary relations by bootstrapping. The ability to go beyond binary relations to n-ary relations is made possible by DARE's recursive and compositional rule framework. DARE learns these rules automatically from text starting with a set of seed facts.

An extension of the bootstrapping approach is distant supervision proposed by Mintz et al. (2009). Here, a knowledge graph is used as a source of seed facts for bootstrapping from a corpus. These bootstrapped facts are used to learn classifiers that can recognize the occurrence of knowledge graph relations in text. The classifier takes into consideration rich lexical, syntactic and NER features, and conjunctions of these. For training the classifier, negative examples are also needed. These are generated from sentences containing entity pairs not connected by the specific relation in the knowledge graph.

PROSPERA (Nakashole et al., 2011) is an extension of distant supervision that can automatically generalize extraction patterns and uses constraint-based reasoning for controlling the quality of patterns and facts. Pattern generalization, achieved through frequent itemset mining, allows for higher recall. By using MaxSat reasoning, PROSPERA can control the quality of the resulting extractions by checking for their mutual consistency. It builds on SOFIE (Suchanek et al., 2009) but is more scalable and provides higher recall. This is achieved by moving to so-called n-gram-itemset patterns rather the phrases or dependency paths for expressing relations.

Blessing and Schütze (2012) introduce crosslingual distant supervision. In this setting, multiple knowledge graphs in different languages are available with linkage among their entities. For a given relation, a pivot KG is chosen, which is one that provides good coverage on that specific relation. Using the linkage between the entities in the different language KGs, it is able to find occurrences of the arguments of a fact from the pivot KG in text in the target language as is done in traditional distant supervision. These annotations can subsequently be used to learn extractors for the specific relation in the target language. By working with multiple languages, this approach produces high accuracy extractions through crosslingual filtering and achieves higher coverage in cases where some relations are not covered by certain languages.

5.2.3. Relation Discovery

The system we propose here is not restricted to relations from an existing repository. Instead, in line with OpenIE, it can discover relations as part of the extraction process. Other works have focused on the task of relation discovery (or relation extraction). Patty (Nakashole et al., 2012) presents a scheme for large scale extraction of typed textual patterns denoting binary relations and organizing them into a subsumption hierarchy. Other works (e.g., Pasca and Durme (2007, 2008)) mine relational patterns for specific semantic classes (e.g., movies or musicians). In this setting, the problem is to determine what the relevant relations for a domain are.

5.2.4. Semantic Knowledge Graphs

In Section 2.1 we introduced the knowledge graphs DBpedia, Freebase, and Yago. These generally provide very limited coverage when it comes to relations. Additionally, these knowledge graphs contain predicates (strictly speaking identifiers such as URIs), but no natural language patterns for them. Such patterns are essential for downstream applications. One use of OpenIE extractions is to link predicates in a semantic knowledge graph with possible natural language manifestations (see, for example, Nakashole et al. (2012)) for use in such tasks as entity search, question answering, and semantic parsing.

5.2.5. Hypernym/Hyponym Extraction

In our work, noun phrase relations, also in combination with their subjects, can be seen as hypernyms for their corresponding objects (i.e., a hypernym/hyponym pair). For example, the fact (‘*NPR*’, ‘*legal affairs correspondent*’, ‘*Nina Totenberg*’) can also be seen as stating that '*Nina Totenberg*' is a member of the classes '*legal affairs correspondent*' and '*NPR legal affairs correspondent*'. The seminal work of Hearst (1992) introduced Hearst patterns for hypernym extraction from free text. The manually defined patterns we present in Section 5.4 are similar in spirit to Hearst patterns, but result in triples rather than pairs. By extracting triples, we essentially account for cases where complex noun phrases denoting the hypernym in the hypernym/hyponym setting can be decomposed. The work of Venetis et al. (2011) assigns class labels to Web table columns, obtaining a database of `isA` relation instances in the process.

5.2.6. Semantic Role Labeling

In principle, the task of extracting noun phrase mediated relations can be compared to that of semantic role labeling (SRL) for nouns. The task in SRL is to identify a relation, expressed either through a verb or a noun, map it to a semantic frame, and map the arguments of the relation to the various roles within the frame. State-of-the-art SRL systems, such as that of Johansson and Nugues (2008), are trained on NomBank (Meyers et al., 2004) for handling noun relations, which also means that they are limited by the knowledge this resource has. ReNoun, on the other hand, can extract the subject and object roles for a far larger number of noun phrase relations, as we demonstrate in our experiments. However, being restricted to triples, ReNoun is unable to extract values for any roles other than those of subject and object.

5.2.7. Applications of OpenIE

OpenIE has been used in several applications. Fader et al. (2013, 2014) used OpenIE extractions, coupled with KG facts for large-scale question answering. These works, along with several others (e.g., Berant et al. (2013) and Cai and Yates (2013)) use OpenIE extractions to find paraphrases for binary predicates in knowledge graphs.

Stanovsky and Dagan (2015) shows significant improvements on the text comprehension and word similarity tasks achieved by integrating OpenIE extractions into existing systems that tackle these tasks.

We used OpenIE extractions in Chapter 4 for creating an extended knowledge graph (XKG) and paraphrasing KG predicates. The goal there was to allow for flexible answering of triple-pattern queries by accounting for missing data in the KG and the terminological and structural gaps between a query and the KG.

5.3. Noun Phrases as Relations

ReNoun's goal is to extract facts for relations expressed as noun phrases. Two natural questions to ask are (i) how important are noun phrases as relations, and (ii) how far prior work on OpenIE goes towards the goal of extracting facts centered around

Knowledge Graph	**%Nouns**	**%Verbs**
Freebase	97	3
DBpedia	96	4

Table 5.2.: Percentage of relations expressed as nouns phrases among the 100 relations with the most facts.

noun phrases. Answering the first question provides justification for going after noun phrases as relations, as KGs generally contain relations deemed to be most interesting because, for example, they answer a large number of user queries in a Web search setting. The answer to this question also provides some guidance for the case where we want to exploit OpenIE extractions to populate KG predicates. The answer to the second question informs us how large of a gap our work needs to fill.

To answer the first question, we look at two major knowledge graphs, Freebase (Bollacker et al., 2008) and DBpedia (Auer et al., 2007), to see how important noun phrases are as relations in these knowledge graphs. For this, we consider the top 100 relations in terms of the number of instances they have in each of the two knowledge graphs. Table 5.2 shows that the vast majority of these fat head relations are noun phrases. If we consider the long tail of relations in each of the two knowledge graphs, then the share of relations expressed as nouns increases.

We answer our second question, as to whether existing, verb-centric OpenIE approaches are sufficient, negatively. We present concrete experimental results to support this answer. As this requires a detailed explanation of the experimental setup, we refer the reader to Section 5.7.2. The most interesting conclusion is that on a large corpus, a state-of-the-art OpenIE scheme extract facts for only 31% of the noun phrase relations which our work extracts facts for. Here, we notice that as relation names get longer (e.g., '*chief privacy officer*', '*automotive division*'), we are less likely to be able to extract their instances with verb-based OpenIE schemes.

Given the answers to the two questions, it is clear that a noun-centric OpenIE scheme is desired. Noun phrases constitute the vast majority of relations in KGs, which implies that they have great importance as relations. The answer to the second question clearly shows that existing OpenIE approaches are not sufficient when it comes to noun phrases as relations.

It is also important to consider what the answers to the two questions mean together. For this, let's consider the setting where we want to use existing verb-centric OpenIE schemes to find more instances of existing KG relations. From the sentence *"Obama advised Merkel on saving the Euro"*, an OpenIE scheme will be able to extract the fact ('*Obama*','*advised*', '*Merkel*'). However, we would not want to use this fact to populate the `advisor` predicate in a KG (e.g., `(GeorgeBush, advisor, CondoleezzaRice)`), as it does not imply that Obama is an advisor of Angela Merkel in the common sense

1. *the* R *of* S*,* O – the CEO of Google, Larry Page
2. *the* R *of* S *is* O – the CEO of Google is Larry Page
3. O, S R – Larry Page, Google CEO
4. O, S*'s* R – Larry Page, Google's CEO
5. O*,* [*the*] R *of* S – Larry Page, [the] CEO of Google
6. SRO – Google CEO Larry Page
7. S R*,* O – Google CEO, Larry Page
8. S*'s* R, O – Google's CEO, Larry Page

Table 5.3.: High precision patterns used for seed fact extraction along with an example of each. Here, the object (O) and the relation (R) corefer and the subject (S) is in close proximity. In all examples, the resulting fact is (`Google`, '*CEO*', `Larry Page`). Patterns are not relation specific.

of `advisor`. Applying existing verb-based OpenIE schemes with the goal of populating existing KG predicates with more instances would require that we paraphrase these verb-based extractions to noun-based ones, requiring us to solve the paraphrasing problem (Madnani and Dorr, 2010).

5.4. Seed Fact Extraction

The input to ReNoun is only relations, without any instances of these relations. The first stage in ReNoun's pipeline is to extract a set of high-precision *seed facts* that are used to train more general extraction patterns. For seed fact extraction, ReNoun uses a set of manually crafted patterns over tokens, noun phrase chunks, named entity annotations, and coreference annotations. Table 5.3 shows the eight patterns ReNoun uses for extracting seed facts. Each pattern is shown along with an example for how it applies to the '*CEO*' relation. The application of the these patterns is tailored to our setting of noun phrase relations.

Specifically, when we apply an extraction pattern to generate a triple *(S,R,O)* we require that the relation *R* is a noun phrase, obtained via a noun phrase chunker and is in our relation repository. The subject *S* and the object *O* are both named entities. Additionally, utilizing the fact that our relations are noun phrases, we require that *R* and *O* corefer. For example, in Figure 5.1, '*CEO*' is in our relation set (Biperpedia) and the coreference resolver annotates '*CEO*' and '*Larry Page*' to be part of the same coreference cluster (cluster 2). The use of coreference follows from the simple observation that objects will often be referred to by nominals, many of which are our relations of interest. Since the sentence matches our sixth extraction pattern, ReNoun extracts the triple (`Google`, '*CEO*', `LarryPage`).

Document:
"[Google]$_1$ [CEO]$_2$ [Larry Page]$_2$ started his term in 2011, when [he]$_2$ succeeded [Eric Schmidt]$_3$. [Schmidt]$_3$ has since assumed the role of executive chairman of [the company]$_1$."

(a)

Coreference clusters:

#	**Phrases**	**Freebase ID**
1	**Google**, the company	`/m/045c7b`
2	**Larry Page**, CEO, he	`/m/0gjpq`
3	**Eric Schmidt**, Schmidt	`/m/01gqf4`

(b)

Figure 5.1.: Coreference clusters: (a) a document annotated with coreference clusters; (b) a table showing each cluster with the representative phrases in bold and the Freebase ID to which each cluster maps.

The seed fact extraction patterns are very similar to Hearst patterns in spirit (Hearst, 1992). However, the goal here is different. Whereas Hearst patterns extract pairs (a noun phrase and its hyponym), our seed patterns extract triples (a relation and its two arguments). In our extractions, the subject and relation can be considered a noun phrase and its hypernym, respectively. However, many of our relations (e.g., '*daughter*'), are rarely interesting as hypernyms in a larger application.

We rely on a coreference resolver in the spirit of Haghighi and Klein (2009). For each cluster of coreferring mentions, the resolver also chooses a representative mention, which is a proper noun or proper adjective (e.g., *Canadian*). Each cluster is possibly linked to a Freebase entity with a unique ID. Figure 5.1(b) shows the coreference cluster from the example document of Figure 5.1(a), with representative phrases in bold, along with the Freebase ID of each cluster. Note that in this example, the noun phrase '*executive chairman*', which is in our relation set, is not part of any coreference cluster. The fact centered around this relation will not be extracted at this point using the patterns of Table 5.3, but could be extracted in the next phase. The resulting facts use Freebase IDs for subject and object (for readability, we will use entity names in the rest of this chapter).

An important consideration in our seed fact extraction patterns is that they also specify the assignment of relation arguments to the subject (S) and object (O) roles based on the semantics of the resulting triple. This is important since it allows us to correctly make the role assignments for the arguments in our dependency parse patterns later in Section 5.5.1.

In summary, our seed extraction process proceeds in two steps. First, we find candidate relation-object pairs that corefer and in which the relation is in our relation repository. Second, we match these sentences against our hand-crafted rules to generate the extractions. In Section 5.7.4 we show that the precision of our seed facts is 65% for the fat head relations and 80% for the long tail ones.

5.5. Pattern and Candidate Fact Generation

With the seed facts in hand, we can now proceed to generate dependency parse patterns, which we later use for finding new instances of our relations that we have not observed before in our seed facts. For our purposes, a *dependency parse* of a sentence is a directed graph whose vertices correspond to tokens in the sentence and edges correspond to grammatical relations that hold between these tokens (Marneffe et al., 2006). A vertex is labeled with a pair of token and part-of-speech tag, and edges are labeled with one of a predefined set of grammatical relations.

A *dependency pattern* is a subgraph of a dependency graph where the words in zero or more vertices have been replaced by placeholder variables, while maintaining the POS tag (a process called *delexicalization of a vertex*). Applying such a pattern to a sentence results in zero or more subgraphs of the sentence's dependency parse that match the pattern, along with assignments of the placeholder variables in the pattern to concrete tokens. The benefit of using dependency patterns is their ability to generalize, as they ignore extra tokens in the sentence that do not belong to the dependency subgraph of interest.

5.5.1. Dependency Pattern Generation

We present the procedure for generating dependency patterns in Algorithm 5.1 along with an example run in Figure 5.2. The procedure expects as input a set of seed facts (Section 5.4) and our annotated corpus 5.7.1.

The procedure iterates over sentences in the corpus, looking for a sentence s matching a seed fact $f = (S, R, O)$, where a match occurs if s has a noun phrase matching R and two entity annotations matching S and O (they belong to a coreference cluster that maps to the same Freebase entity as the subject and object of f). When a match occurs, we hypothesize that the sentence s expresses the fact f. Our experiments and error analysis later in this chapter indicate that this is a reasonable hypothesis. Additionally, our scoring scheme presented later will allow us to remove noise obtained through this assumption.

We now proceed to extract a dependency pattern from the matched sentence. In practice, a pattern P is composed of two distinct patterns, P_{SR} connecting the subject to the relation, and P_{OR} connecting the object to the relation. In what follows, we focus on P_{SR} as the generation of P_{OR} is analogous. P_{SR} is taken to be the path in the dependency parse of s connecting the lexical head of the phrase corresponding to

S with the lexical head of R. To generate a pattern, we delexicalize the two tokens at the ends of P_{SR} (the lexical heads of S and R) and assign them the variables `S` and `R` corresponding to their roles. Additionally, if the head of the phrase corresponding to O occurs in P_{SR}, we delexicalize it as well and assign it the variable `O`. Relations have nouns as their head words. The subject and object heads can be nouns, adjectives (e.g., American), or pronouns (e.g., its). We lift all POS tags corresponding to nouns in the nodes corresponding to the heads of S or O to N.

In principle, the head of the token need not be delexicalized. However, as pattern matching is an expensive step, we do this to reduce the number of distinct dependency patterns matched, as many patterns would otherwise differ only at the relation head vertex. Each pattern is produced with a list of the relations to which it applies.

Algorithm 5.1: Dependency pattern generation.

```
  input : Seed facts $I$, Corpus $D$.
1 $\mathcal{P}$ := An empty set of dependency pattern-relation pairs.
2 foreach sentence $s \in D$ do
3     foreach triple $t = (S, R, O)$ found in $s$ do
4         if $t \in I$ then
5             $G(s)$ = dependency parse of $s$
6             $P'$ = minimal subgraph of $G(s)$ containing the head tokens of
              $S$, $R$ and $O$
7             $P$ = Delexicalize($P'$, $S$, $R$, $O$)
8             $\mathcal{P} = \mathcal{P} \cup \{\langle P, R \rangle\}$
9 return $\mathcal{P}$
```

It is important to note that our dependency patterns clearly distinguish the roles of subject and object. This is facilitated by the manner in which we generate our seed facts (Section 5.4), which show which argument will take the role of subject and which will take the role of object. This is in contrast to earlier work (e.g., OLLIE), where the assignment depends on the order in which the arguments are expressed in text. The distinction can be crucial for applications that utilize our extracted facts. For example, given the sentence *"Opel was described as GM's most successful subsidiary"* and the seed fact (`GM`, '*subsidiary*', `Opel`), the pattern ReNoun generates will consistently produce facts like (`BMW`, '*subsidiary*', `Rolls-Royce`), and not the incorrect inverse.

5.5.2. Dependency Pattern Application

Given the learned patterns, we can now generate new extractions by applying these patterns. Each match of a pattern against the corpus will indicate the heads of the

Relations: $\mathcal{R}$ ={`executive chairman`}
Seed fact: I = {`(Google, executive chairman, Eric Schmidt)` }
Sentence: s = *"An executive chairman, like Eric Schmidt of Google, wields influence over company operations."*

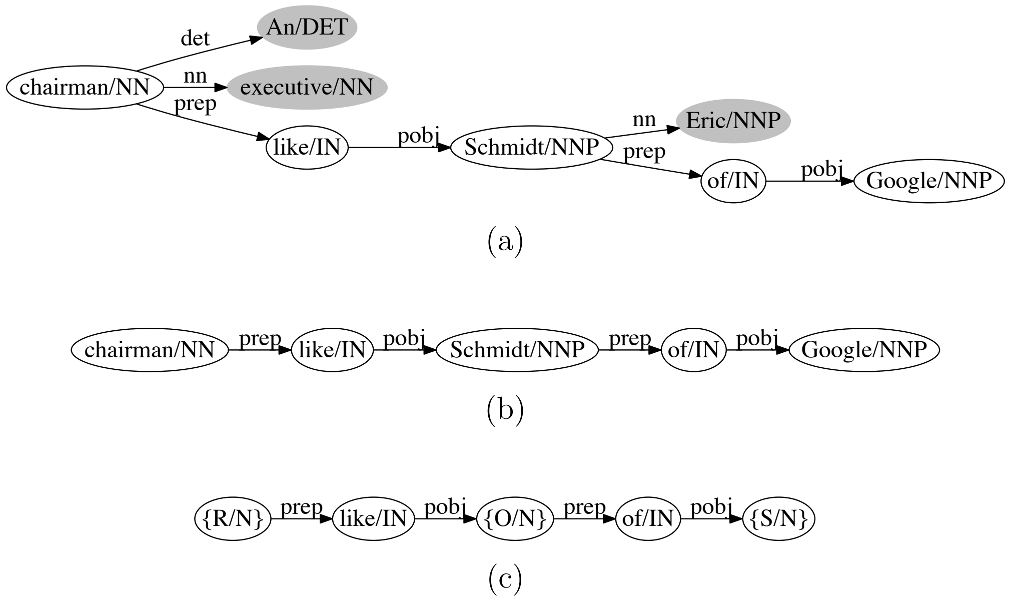


Figure 5.2.: Dependency pattern generation using seed facts, corresponding to Algorithm 5.1: (a) shows the input to the procedure (dependency parse partially shown); (b) P'; (c) P.

potential subject, relation, and object. The noun phrase headed by the token matching the {`R/N`} vertex is checked against the set of relations to which the pattern is applicable. If it is found among these relations, then a triple `(S,R,O)` is constructed from the relation and the Freebase entities to which the phrases headed by the tokens corresponding to the `S` and `O` nodes in the pattern are resolved. This triple is then emitted as an extraction along with the pattern that generated it. Figure 5.2(b) and (c) show two sentences that match the dependency pattern in our running example and the resulting extractions.

Finally, we aggregate our extractions by their generated facts. For each fact f, we save the distinct dependency patterns that yielded f and the total number of times f was found in the corpus by each pattern.

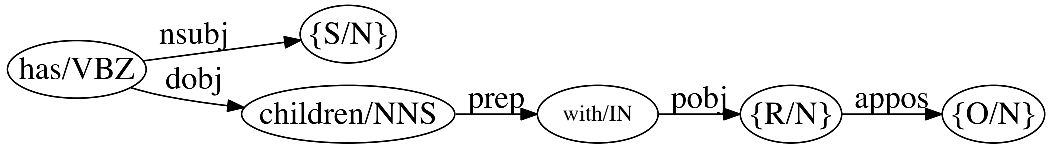
5.6. Fact Scoring

The scheme described above for fact extraction can produce a large number of facts, some of which can be erroneous either due to the assumption made in Section 5.5 or simply due to erroneous annotations from the NLP pipeline. Our approach to tackle this problem is to assign each fact a score that reflects our confidence that the fact actually holds. A consumer of the facts generated can subsequently use these scores either as information to a larger learning framework or to impose a cutoff threshold on the score based on the error tolerance of her application.

ReNoun's scoring scheme is based on two observations about extraction patterns. The first relates to pattern frequency, while the second relates to pattern coherence:

1. **Pattern frequency:** Our observation is that patterns that produce a large number of extractions will always produce correct extractions (in addition to incorrect ones). We define *pattern frequency* as the absolute number of facts a dependency pattern is able to extract.

2. **Pattern coherence:** On the other hand, generic patterns that apply to a wide variety of relations result in more noise than specialized patterns that apply to a targeted set of relations from a limited domain. To capture this intuition, we introduce the notion of *pattern coherence*, which reflects how targeted a pattern is, based on the set of relations to which it applies. For example, we observed that if an extraction pattern yields facts for the coherent set of relations '*ex-wife*', '*boyfriend*', and '*ex-partner*', then its output is consistently good. On the other hand, a pattern that yields facts for the less coherent set of relations '*ex-wife*', '*general manager*', and '*subsidiary*' is more likely to produce noisy extractions. Generic patterns that apply to incoherent relations are more sensitive to noise in the linguistic annotation of a document. Figure 5.3 shows an example pattern for each case, along with its frequency and coherence, computed as described next.

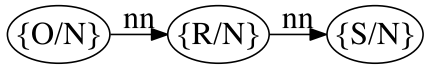


relations: { `ex-wife`, `boyfriend`, `ex-partner` }

$frequency(P) = 574, coherence(P) = 0.429$

Example: *"Putin has two children with his ex-wife, Lyudmila."*

(a)



relations: { `ex-wife`, `general manager`, `subsidiary`,... }

$frequency(P) = 52349038$, $coherence(P) = 0.093$

Example: *"Chelsea F.C. general manager José Mourinho..."*

(b)

Figure 5.3.: Example of two dependency patterns generated by ReNoun along with the set of relations they apply to, (a) has high coherence and low frequency while (b) has low coherence and high frequency.

As stated above, the computation of a pattern's frequency is simply the number of facts it extracts. We capture the coherence a pattern through the coherence of the relations to which it applies, based on the intuition outlined above. We compute the coherence of relations using their distributed vector representations following the scheme described by Mikolov et al. (2013). This work has shown its effectiveness in measuring the semantic similarity between words or concepts. We use the skip-gram model, which is generated by training a neural network to predict the vectors of the context words given the vector of the current word. The similarity between a pair of words or phrases can then be measured using the similarity of their respective vector representations (e.g., using cosine similarity, as is common in the literature).

In our setting, given a dependency pattern P that applies to the relations $\{R_1, ..., R_n\}$, with each relation R_i having the vector representation $v(R_i)$, we define the coherence of P to be the average pairwise similarity of the vectors corresponding to the relation to which it applies:

$$coherence(P) = \frac{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} sim(R_i, R_j)}{n(n-1)} = \frac{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{v(R_i)\cdot v(R_j)}{\|v(R_i)\|\|v(R_j)\|}}{n(n-1)}.$$

Finally, we score a fact f by summing the product of frequency and coherence for each pattern P generating f as follows:

$$score(f) = \sum_{P \in Pat(f)} frequency(P) \times coherence(P),$$

where $Pat(f)$ is the set of all patterns that generate f, $frequency(P)$ is the number of distinct facts generated by P, and $coherence(P)$ is as defined above.

5.7. Experimental Evaluation

In this section, we present experiments to:

i. demonstrate the need for our noun-centric approach to open information extraction,

ii. evaluate the quality of the facts obtained at each step of ReNoun's pipeline, and

iii. demonstrate the effectiveness of our proposed scheme for scoring extractions.

We start by describing our experimental setup.

5.7.1. Setup

We use Freebase as our reference knowledge graph against which named entity mentions are resolved. It is important to stress that Freebase relations are not used at all, as we seek to stay within the OpenIE paradigm. We work with a corpus of 400M news documents which have been annotated offline with noun phrase chunks, dependency parses, coreference links, and named entities linked to Freebase.

While relations can be discovered as part of ReNoun's extraction scheme, we chose to work with an existing repository of relations generated using the scheme described by Gupta et al. (2014). Doing this allows us to later analyze what percentage of potentially interesting relations ReNoun can extract facts for.

To define the sets of fat head (FH) and long tail (LT) relations, we order the relations in descending order of the number of times they occur in our text corpus. We define cutoff point N as the position in this list where the total frequency of relations above and below this point is equal. All relations above this cutoff are defined to be in FH while all those below it are in LT. In our corpus FH contains 218 relations (i.e., $N = 218$) and LT contains 60K relations.

ReNoun	**OLLIE** (Mausam et al., 2012)
flagship company	-
railway minister	-
legal affairs correspondent	-
spokesperson	be spokesperson of
president-elect	be president elect of
co-founder	be co-founder of

Table 5.4.: Long tail ReNoun relations with and without a corresponding OLLIE relation.

When evaluating the correctness of facts, we take the majority vote of three human judges unless otherwise stated. The judges were explicitly instructed to consider facts with inverted subject/object arguments as incorrect. For example, the fact (`GM`, '*subsidiary*', `Opel`) is considered correct, while its inverse is not.

5.7.2. Comparison to State-of-the-Art OpenIE

State-of-the-art OpenIE systems such as OLLIE (Mausam et al., 2012) assume that relations are expressed using verb phrases, which can include a noun phrase within them. In this experiment, we show that this is not sufficient and that our approach is required for more comprehensive coverage of noun phrase relations. We compare the number of attributes covered by ReNoun to those covered by OLLIE. We use a custom implementation of OLLIE that supports multi-word relations and run it on the same corpus as ReNoun.

For more insight, we report on results for FH and LT. We randomly sampled each of the two sets for 100 relations for which ReNoun extracts facts and asked a human judge to find OLLIE extractions involving a similar relation. We did not require that the two be exactly the same. For example, the relation '*advised by*' should be considered a match to the ReNoun attribute '*advisor*'. This is somewhat permissive, as nouns and their corresponding verbs do not necessarily carry the same meaning, and should not be considered synonyms. For example, the sentence *"Obama advised Merkel on saving the Euro"* does not imply that Obama is Merkel's '*advisor*'. This observation suggests a subtle difference between the meaning of verb and noun-based expressions. This experiment, therefore, gives us an upper bound on the number of ReNoun relations OLLIE can cover.

The results show a big discrepancy in OLLIE's ability to cover the long tail compared to the fat head of relations. On FH, OLLIE can extract facts for 99 of the 100 randomly sampled relations. However, for LT, OLLIE manages to extract facts for 31 of the 100 relations sampled, despite our permissive setting. The majority of these relations were multi-word noun phrases. Table 5.4 shows examples of LT relations with and without OLLIE extractions.

We make a similar comparison in the other direction, by sampling random relations for which OLLIE extracts facts, and checking if ReNoun generates facts for the same relations. For this, we randomly sampled 100 such relations, and checked them against ReNoun's extractions. We only managed to find matches for 48, with 52 having no corresponding facts in ReNoun. 25 of the 52 missed relations were not in our initial list of relations, which meant that we cannot extract facts for them. The remaining 27 cases have various explanations. First, some relations express actions that need to be part of a larger verb phrase (i.e., noun phrases do not suffice). In general, these cannot be considered relations of the subject entity (e.g., '*citation of*' in *"Obama's citation of the Bible"*). Second, some relations have the object, a common noun, embedded within them (e.g., '*have microphone in*'), and do not have corresponding relations. The remaining relations either have meaningless extractions or use common noun phrases as arguments. ReNoun focuses on facts with proper nouns (i.e., entities) for arguments because facts with common nouns as arguments are rarely interesting without a larger context. Finally, we note that the majority of the 25 OLLIE relations that are not in our initial set of relations fall into one of the three categories above.

5.7.3. Comparison to NomBank

In Section 5.2 we presented the task of semantic role labeling (SRL) and contrasted it with our task. To demonstrate the limitation of having SRL systems trained on such resources as NomBank (Meyers et al., 2004), we asked a judge to manually search NomBank for 100 relations randomly drawn from FH and LT for which ReNoun extracts facts. Again, we take a liberal approach to matching similar to what we did above with OLLIE to establish an upper bound. We declare a match if the head word of a relation is found in NomBank. In this way, we could match 80 FH relations and 42 LT ones. For example, we could not find matches for '*coach*' (FH) or '*linebacker*' (LT). This is rather unsurprising given that NomBank contains 4700 relations only.

Looking deeper into NomBank's frames, some nouns are not associated with frames that allow the extraction of triples. For example, all frames for the noun '*member*' specify one argument only, which means that in the sentence *"John became a member of ACM"*, the output relation is the pair (`ACM`, '*member*') instead of the desired triple (`ACM`, '*member*', `John`). Of course, some frames in NomBank allow for the extraction of more than triples, which is beyond ReNoun's ability.

Finally, similar to what we did for OLLIE, we consider the NomBank nouns for which ReNoun offers no extractions. Out of a random sample of 100 NomBank nouns, ReNoun does not extract facts for 29 (four of which are not in our attribute set). The majority of the missed nouns cannot be used by ReNoun because they expect a single argument or take a prepositional phrase or common nouns rather than proper nouns corresponding to entities as their second arguments.

5.7.4. Seed Fact Quality

We now present an experimental evaluation of the quality of the seed facts extracted using the manually specified patterns described in Section 5.4. Applying these patterns to our corpus resulted in 139M extractions, which boiled down to about 680K unique facts covering 11,319 attributes. We evaluated 100 randomly chosen facts from each of FH and LT and obtained precisions of 65% and 80%, respectively. We comment on these findings in what follows.

First, the precision of the LT facts at 80% is high, making it suitable for use as a building block in a bootstrapping scheme as we will show later. ReNoun, is primarily interested in LT attributes, which earlier approaches cannot deal with satisfactorily as we demonstrated in Section 5.7.2.

Second, LT attributes have higher precision than FH ones. One reason is that multi-word relations (which tend to be in LT) are sometimes incorrectly chunked, causing only the head to be recognized as the relation and resulting in a fact not actually being expressed in text. For example, in the phrase *"America's German coach, Klinsmann"*, the correct relation is '*German coach*' (LT), but incorrect chunking produces the relation '*coach*' (FH) and the incorrect fact (`Germany`, '*coach*', `Klinsmann`). Another reason for the lower precision on FH relations is that they are more likely to occur in a speculative context where the presence of the relation is not necessarily an assertion of a fact. In principle, both LT and FH attributes can be subject to this problem, but we observe this much more for FH attributes. For example, before a person is a '*railway minister*' (LT), there is little mention of her along with the relation. On the other hand, before a person is elected '*president*' (FH), there is usually media coverage of her candidacy. Speculative contexts, combined with incorrect linguistic analysis in some instances, can result in incorrect seed facts. For example, from the sentence '*Republican favorite for US president, Mitt Romney, visited Ohio*', we extract the fact (`US`, '*president*', `Romney`).

5.7.5. Candidate Generation

Using seed facts, we deploy our candidate fact extraction scheme as described in Section 5.5. We call the facts extracted at this stage candidates since they will be subject to scoring in the next stage, which will affect how upstream applications use them, if at all. In the first step, we produced a total of 2M unique dependency patterns. A third of these patterns could extract values for exactly one attribute. Manual inspection of these patterns showed that they were either noise, or do not generalize. We kept patterns supported by at least 10 distinct seed facts, yielding more than 30K patterns.

We then applied the patterns to our corpus, resulting in over 460M extractions, aggregated into about 40M distinct facts. Of these, about 22M facts were for LT attributes and 18M for FH. In general, we find that these facts contain a great deal of noise, which presents a need to score them. We evaluate the quality of these facts in conjunction with our fact scoring scheme.

5.7.6. Scoring Extracted Facts

ReNoun produces a large number of extractions. However, most of these are noisy. In Section 5.6 we presented our scheme for assigning scores to facts, which applications on top of ReNoun can use. In this section we present experiments that demonstrate the effectiveness of our scoring scheme. We do this by (i) comparing it to other natural scoring baselines, and (ii) showing the quality of the top-k scored facts produced using this scheme at various cutoff thresholds k.

In the first experiment on our scoring scheme (which we call FREQ_COH), we compare it against three natural baselines. The first, FREQ scores a fact f based on the sum of the frequencies of the patterns that extract it. The second baseline, PATTERN, scores a fact by the distinct number of patterns that extract it. The final baseline, PATTERN_COH, weighs each pattern by its coherence.

Scheme	**Definition,** $score(f) =$	**Spearman's** ρ
FREQ	$\sum_{P \in Pat(f)} frequency(P)$	0.486
FREQ_COH	$\sum_{P \in Pat(f)} frequency(P) \times coherence(P)$	**0.495**
PATTERN	$\sum_{P \in Pat(f)} 1$	0.265
PATTERN_COH	$\sum_{P \in Pat(f)} coherence(P)$	0.257

Table 5.5.: Scoring schemes.

We generated 250 random facts with no entity disambiguation errors by the underlying annotation pipeline. The justification is that none of the schemes we consider here is capable of dealing with such errors. To account for such errors would require the consideration of additional signals from the named entity disambiguator, which we leave for future work. For each scoring scheme, we compute the Spearman's rank correlation coefficient ρ between the scores and the manual judgments (Baeza-Yates and Ribeiro-Neto, 1999). A larger ρ indicates more correlation. All ρ values were determined to be statistically significant (p-value ≤ 0.01).

FREQ and FREQ_COH dominate, but both have similar performance. This result shows that considering pattern frequency is very helpful. We observed that coherence helped in cases where facts have very similar frequencies. In this case, considering coherence usually helped score the correct fact higher than the incorrect one. This effect, however, is tempered when we consider a large number of facts.

	FH		**LT**	
k	Precision	#Rel	Precision	#Rel
10^2	1.00	8	1.00	50
10^3	0.98	36	1.00	294
10^4	0.96	78	0.98	1548
10^5	0.82	106	0.96	5093
10^6	0.74	124	0.70	7821
All	0.18	141	0.26	11178

Table 5.6.: Precision of random samples of the top-k scoring facts, along with the relation yield.

In the second experiment, we evaluate our scoring scheme by considering the precision of the top-k facts for various cutoff thresholds k. Here, for each value of k we randomly sample 50 facts from the set of k facts with the highest scores and present them to our judges for evaluation. We would expect a good scoring scheme to surface correct facts to the top and incorrect ones to the bottom of the raking it induces. This means that as the value of k increases, we should expect a drop in precision. Table 5.6 shows the precision values at each cutoff threshold as well as the number of distinct relations for which facts are found at this threshold (#Rel).

The results show that precision decreases as we increase k, which is the behavior we would expect from a good scoring scheme. At a cutoff of 1M, we obtain precision values ≥ 0.70, which is satisfactory compared to other scheme such as that of Fader et al. (2011) which produces similar numbers at the same cutoff threshold.

There are several sources of error here. First, incorrect dependency parses were sometimes generated, mainly due to incorrect removal of boilerplate text from web pages. The second source of errors was incorrect coreference resolution of pronouns. Incorrect entity disambiguation against Freebase was another source of errors. Finally, some relations require that we go beyond triples for meaningful extractions. The '*ambassador*' relation with two countries as its arguments is an example of such case.

5.7.7. Analysis of missed extractions

As explained above, ReNoun expects a set of relations for which it extracts facts. Out of 60K relations we use in our experiments, we extract facts for less than 12K. It is important to understand what extraction opportunities have been missed by ReNoun in order to extend it in the future. We considered all 77 FH relations for which ReNoun did not generate any facts (out of a total of 218 FH relations), and a random sample of 100 LT relations with the same issue. Table 5.7 shows a breakdown of the issues.

The first three categories in Table 5.7 are cases that are currently outside the scope

Cause	**FH**	**LT**	**Example**
Vague	23	37	‘*culture*’
Numeric	4	26	‘*rainfall*’
Object not KG entity	11	6	‘*email*’
Plural	30	15	‘*member firms*’
Bad relation / misspell	3	4	‘*newsies*’
Value expected	6	12	‘*nationality*’
Total	77	100	

Table 5.7.: Analysis of relations with no extractions.

of ReNoun: vague relations whose values are long narratives, numeric attributes, and relations whose values are not modeled as Freebase entities (e.g., ‘*email*’). The next two categories are due limitations in the relation repository we use (Biperpedia). We take relations as they are in Biperpedia. If they are in plural, we do not attempt to place them in singular. Additionally, some relations are simply bad, often resulting from misspelled words. Finally, the “Value expected” category contains relations for which ReNoun *should* have extracted instances. This is the most interesting category of attributes in this analysis. One reason for missing values is that the corpus itself does not contain values for all relations. Another reason is that some relations are not verbalized in text. For example, relations like ‘*nationality*’ are usually not explicitly stated when expressed in text.

5.8. Discussion

In this chapter we presented an approach for open information extraction focused on relations mediated by noun phrases. The goal here was to complement the existing state-of-the-art which has generally focused on relations mediated through verbs. We took a bootstrapping approach where an initial set of seed facts is extracted from a text corpus using a small number of manually specified seed patterns. These seed facts are subsequently matched against the corpus to find more elaborate ways of expressing them. From these matches we generate dependency parse patterns that we run over the corpus to gather before unseen facts. These facts are scored taking into consideration the frequency with which they were observed and the trustworthiness of the patterns that extract them, for which we use pattern coherence as a proxy.

We presented comprehensive experiments that show a clear gap in the state-of-the-art in open information extraction, justifying the need for ReNoun. Moreover, we presented experiments that demonstrate the effectiveness of ReNoun's extraction and scoring scheme, particularly when it comes to the long tail of relations. Finally, we presented an analysis of ReNoun's shortcomings, which were due to a mixture of fundamental limitations (e.g., numerical attributes) and particularities of the corpus at hand. The data produced by ReNoun is in active use in a major search engine.

One direction for future research is to move beyond triples while staying within the OpenIE paradigm. A relation like '*ambassador*' needs at least three arguments (person, represented country, and host country) if not more. The analysis presented at the end of this work shows some clear directions for extension. One clear direction is attributes, where the arguments are not entities but rather literals. The main challenge here will be how to reconcile possibly conflicting numbers and dates and to recognize when triples are sufficient or when there is clear need go beyond triples (e.g., GDP changes yearly). Another class of relations is ones that require a narrative (e.g., '*culture*') where the task of OpenIE might intersect with that of text summarization.

An important consideration is how to extend our approach and other similar approaches beyond English. While we were willing to invest the time to come up with a handful of patterns for seed fact extraction, even this seemingly simple task can become a burden if we need to support multiple languages. Can we, for instance, start with a system designed for a specific language and then rely on parallel copora to automatically learn similar patterns for languages? Faruqui and Kumar (2015) presents an initial effort towards this goal by performing multilingual OpenIE relying on machine translation and crosslingual relation projection.

An interesting question is how the data generated here can be used. There are some clear uses such as extending knowledge graphs or finding paraphrases for relations in a KG. However, we can think of others, in part inspired by the analysis presented in our experiments. We saw some shortcomings of a resource like NomBank in our experiments, an interesting question is how our extractions can be used to automatically extend such a resource – which, in turn, presents a strong case for moving beyond triples. Using OpenIE extractions to extend linguistic resources can have a big impact on tasks using these resources. For instance, a task like abstract meaning representation (AMR) parsing would greatly benefit from the automated extension of NomBank.

6. Conclusion

6.1. Summary

This dissertation presented three contributions to facilitate effective querying of knowledge graphs and acquisition of knowledge. The contributions of this thesis are in the areas of question answering over knowledge graphs, relaxed knowledge graph querying combining structured and unstructured data, and open information extraction.

Our first contribution, DEANNA, allows users to query knowledge graphs using natural language questions. This allows for the expression of sophisticated information needs which can be satisfied with crisp answers in the form of tuples of entities. It also shields users from the complexities of the knowledge graph, which include a potentially large vocabulary and a continuously evolving structure. The main issue DEANNA deals with is that of ambiguity in mapping a question to a query. This ambiguity is both structural and terminological. DEANNA's contribution here is an ILP framework that tackles these ambiguities jointly, as the resolution of one informs the other. Furthermore, to make up for any possible errors in the disambiguation or incompleteness in the knowledge graph, DEANNA will iteratively relax query conditions with unsatisfactory answers by casting them into textual conditions that are evaluated over textual contexts of facts.

Our second contribution, TriniT, tackles the problem unsatisfactory answers for triple pattern queries over knowledge graphs. The issue here is mismatches between a query and the knowledge graph, either due to the incompleteness of the knowledge graph or the user's lack of familiarity with its terminology. Unlike earlier approaches, TriniT compensates for these issues without sacrificing the structure in the query, which is essential for capturing concrete relations between entities. TriniT addresses the two issues using a combination of knowledge graph extension and query relaxation. Knowledge graph extension entails extending the data model and the corresponding query model to allow for facts extracted from text. These facts are allowed to have textual tokens in any of their SPO components. To allow a user's query to make use of this extension, TriniT supports query relaxation, where automatically mined relaxation rules are used to rewrite query conditions during query evaluation. Our scoring scheme guides this process, ensuring both effectiveness and efficiency.

Our final contribution in this dissertation is ReNoun, an OpenIE framework targeting relations mediated through noun phrases. ReNoun complements the state of the art in OpenIE, which has generally been restricted to relations mediated by verbs, thereby missing a great deal of extraction opportunities, particularly when it comes to relations in the long tail. ReNoun adopts a bootstrapping approach to extraction, where a small set of manually specified lexical patterns is used to extract seed facts centered around noun phrases. These seed facts are subsequently used to find dependency parse patterns that can express instances of each of the detected relations, allowing for very elaborate ways of expressing facts to be captured. In contrast to earlier work, ReNoun takes care in assigning the subject and object roles in a fact, thereby ensuring consistency, which is crucial for upstream applications. To compensate for noise in the extraction process, ReNoun assigns scores to each fact, reflecting the confidence in the fact. A fact is scored by considering the frequency with which it was extracted, and the trustworthiness of each of the patterns that extract it, which we capture using a notion of relation coherence.

6.2. Outlook

While the approaches presented in this work go a long way towards facilitating effective querying and acquisition of knowledge, the problems we tackle in this dissertation are far from being solved. We addressed the shortcomings and opportunities for extension of each of our contributions in the respective chapters. We conclude this dissertation by discussing themes for future research that we believe are crucial for effective querying and acquisition of knowledge.

Beyond English

The work presented in this dissertation is heavily focused on English in several ways: question analysis and parsing for DEANNA assume that the question provided is in English, TriniT relies on large entity-annotated corpora, and ReNoun relies on English-specific patterns for seed fact extraction and corpora that have been annotated with dependency parses, which are not universally available for all languages. An important research question is: how can we extend the work in this dissertation to be multilingual. The two obstacles in this respect are (i) the availability of NLP tools such as dependency parsers, noun phrase chunkers, and entity disambiguation systems, and (ii) the availability of data in the form of dictionaries, and annotated corpora. The long term goal here would be *truly language-agnostic approaches.*

Beyond Question/Answer

In both DEANNA and TriniT, as is the case for other similar systems, human and machine take turns in issuing input on one side, and coming up with an answer on the other. However, this is unnatural. Machines should be able to ask humans to help out in resolving ambiguity, they should also be able to ask for more context whenever this can help the answering process. Humans, on the other hand, should be able to see explanations of how machines arrive at an answer. They should then be able to ask for refinements or modifications based on this information. Achieving this presents challenges that across several communities including natural language processing, information retrieval, dialog systems and human-computer interaction. The long term here would be *interactive question answering*.

Appendices

A. DEANNA Workload

#	**Query**
1	Who was the successor of John F. Kennedy?
2	Who is the mayor of Berlin?
3	Give me all professional skateboarders from Sweden.
4	Give me a list of all trumpet players that were bandleaders.
5	Give me all members of Prodigy.
6	Give me all cars that are produced in Germany.
7	Give me all people that were born in Vienna and died in Berlin.
8	What is the capital of Canada?
9	Who was the father of Queen Elizabeth II?
10	Sean Parnell is the governor of which U.S. state?
11	Give me all movies directed by Francis Ford Coppola.
12	Give me all actors starring in movies directed by and starring William Shatner.
13	Give me all current Methodist national leaders.
14	Give me all Australian nonprofit organizations.
15	In which military conflicts did Lawrence of Arabia participate?
16	Who developed Skype?
17	Give me all companies in Munich.
18	List all boardgames by GMT.
19	Who founded Intel?
20	Who is the husband of Amanda Palmer?
21	Give me all breeds of the German Shepherd dog.
22	Which cities does the Weser flow through?
23	Which countries are connected by the Rhine?
24	Which professional surfers were born on the Philippines?
25	In which UK city are the headquarters of the MI6?
26	Which other weapons did the designer of the Uzi develop?
27	Give me all Frisian islands that belong to the Netherlands.
28	What is the ruling party in Lisbon?
29	Which Greek goddesses dwelt on Mount Olympus?
30	Give me the Apollo 14 astronauts.

31	What is the time zone of Salt Lake City?
32	Give me a list of all lakes in Denmark.
33	Which instruments did John Lennon play?
34	Which ships were called after Benjamin Franklin?
35	Who are the parents of the wife of Juan Carlos I?
36	In which U.S. state is Area 51 located?
37	List the children of Margaret Thatcher.
38	Who was called Scarface?
39	Which books by Kerouac were published by Viking Press?
40	Give me a list of all American inventions.
41	Who created the comic Captain America?
42	What is the largest city in Australia?
43	Who composed the music for Harold and Maude?
44	Where is the residence of the prime minister of Spain?
45	Who wrote the lyrics for the Polish national anthem?
46	Who painted The Storm on the Sea of Galilee?
47	Which country does the creator of Miffy come from?
48	Who produces Orangina?

Table A.1.: QALD-2 Questions

#	**Query**
1	birthplace stephen hawking
2	canada's capital
3	capital australia
4	director of pulp fiction
5	richard feynmans advisor
6	author of the waste lands
7	awarded guitarists
8	barcelona artists
9	chicago comedians
10	coastal cities
11	companies based in silicon valley
12	corporate execs born in sydney
13	emmy award actors
14	guitarists awarded a grammy
15	ivy league school US presidents
16	lead guitarists
17	physicists that won the nobel physics prize

18	protest song musician
19	reggae rock trios
20	spanish painters
21	turing award mathematicians
22	woody allen awards

Table A.2.: Telegraphic query workload of Pound et al. (2012)

B. TriniT Workload

#	**Query**
INEX_XER-100	Operating systems to which Steve Jobs related
INEX_XER-108	State capitals of the United States of America
INEX_XER-114	Formula one races in Europe
INEX_XER-115	Formula One World Constructors' Champions
INEX_XER-116	Italian nobel prize winners
INEX_XER-128	Bond girls
INEX_XER-129	Science fiction book written in the 1980
INEX_XER-130	Star Trek Captains
INEX_XER-133	EU countries
INEX_XER-136	Japanese players in Major League Baseball
INEX_XER-140	Airports in Germany
INEX_XER-141	Universities in Catalunya
INEX_XER-143	Hanseatic league in Germany in the Netherlands Circle
INEX_XER-144	chess world champions
INEX_XER-64	Alan Moore graphic novels adapted to film
INEX_XER-81	Movies about English hooligans
INEX_XER-88	Nordic authors who are known for children's literature
INEX_XER-95	Tom Hanks movies where he plays a leading role.
INEX_XER-96	Pure object-oriented programing languages
INEX_XER-99	Computer systems that have a recursive acronym for the name
INEX_XER-106	Noble english person from the Hundred Years' War
INEX_XER-109	National capitals situated on islands
INEX_XER-110	Nobel Prize in Literature winners who were also poets
INEX_XER-113	Formula 1 drivers that won the Monaco Grand Prix
INEX_XER-117	Musicians who appeared in the Blues Brothers movies
INEX_XER-119	Swiss cantons where they speak German
INEX_XER-124	Novels that won the Booker Prize
INEX_XER-125	countries which have won the FIFA world cup
INEX_XER-126	toy train manufacturers that are still in business
INEX_XER-127	german female politicians

INEX_XER-132	living nordic classical composers
INEX_XER-134	record-breaking sprinters in male 100-meter sprints
INEX_XER-135	professional baseball team in Japan
INEX_XER-138	National Parks East Coast Canada US
INEX_XER-139	Films directed by Akira Kurosawa
INEX_XER-147	Chemical elements that are named after people
INEX_XER-60	olympic classes dinghy sailing
INEX_XER-62	Neil Gaiman novels
INEX_XER-63	Hugo awarded best novels
INEX_XER-65	Pacific navigators Australia explorers
INEX_XER-67	Ferris and observation wheels
INEX_XER-72	films shot in Venice
INEX_XER-73	magazines about indie-music
INEX_XER-74	circus mammals
INEX_XER-79	Works by Charles Rennie Mackintosh
INEX_XER-86	List of countries in World War Two
INEX_XER-87	Axis powers of World War II
INEX_XER-91	Paul Auster novels
INEX_XER-94	Hybrid cars sold in Europe
INEX_XER-97	Compilers that can compile both C and C++
INEX_XER-98	Makers of lawn tennis rackets
QALD2_te-2	Who was the successor of John F. Kennedy?
QALD2_te-21	What is the capital of Canada?
QALD2_te-27	Sean Parnell is the governor of which U.S. state?
QALD2_te-28	Give me all movies directed by Francis Ford Coppola.
QALD2_te-33	Give me all Australian nonprofit organizations.
QALD2_te-35	Who developed Skype?
QALD2_te-40	List all boardgames by GMT.
QALD2_te-42	Who is the husband of Amanda Palmer?
QALD2_te-43	Give me all breeds of the German Shepherd dog.
QALD2_te-51	Give me all Frisian islands that belong to the Netherlands.
QALD2_te-55	Which Greek goddesses dwelt on Mount Olympus?
QALD2_te-58	What is the time zone of Salt Lake City?
QALD2_te-59	Which U.S. states are in the same timezone as Utah?
QALD2_te-6	Give me all professional skateboarders from Sweden.
QALD2_te-60	Give me a list of all lakes in Denmark.
QALD2_te-77	Who was called Scarface?
QALD2_te-82	Give me a list of all American inventions.
QALD2_te-87	Who composed the music for Harold and Maude?

QALD2_te-88	Which films starring Clint Eastwood did he direct himself?
QALD2_te-89	In which city was the former Dutch queen Juliana buried?
QALD2_te-90	Where is the residence of the prime minister of Spain?
QALD2_te-91	Which U.S. State has the abbreviation MN?
QALD2_te-95	Who wrote the lyrics for the Polish national anthem?
QALD2_te-98	Which country does the creator of Miffy come from?
QALD2_te-99	For which label did Elvis record his first album?
QALD2_te-100	Who produces Orangina?
QALD2_te-14	Give me all members of Prodigy.
QALD2_te-17	Give me all cars that are produced in Germany.
QALD2_te-19	Give me all people that were born in Vienna and died in Berlin.
QALD2_te-22	Who is the governor of Texas?
QALD2_te-24	Who was the father of Queen Elizabeth II?
QALD2_te-29	Give me all actors starring in movies directed by and starring William Shatner.
QALD2_te-3	Who is the mayor of Berlin?
QALD2_te-31	Give me all current Methodist national leaders.
QALD2_te-34	In which military conflicts did Lawrence of Arabia participate?
QALD2_te-39	Give me all companies in Munich.
QALD2_te-41	Who founded Intel?
QALD2_te-44	Which cities does the Weser flow through?
QALD2_te-45	Which countries are connected by the Rhine?
QALD2_te-46	Which professional surfers were born on the Philippines?
QALD2_te-48	In which UK city are the headquarters of the MI6?
QALD2_te-49	Which other weapons did the designer of the Uzi develop?
QALD2_te-53	What is the ruling party in Lisbon?
QALD2_te-57	Give me the Apollo 14 astronauts.
QALD2_te-63	Give me all Argentine films.
QALD2_te-64	Give me all launch pads operated by NASA.
QALD2_te-65	Which instruments did John Lennon play?
QALD2_te-66	Which ships were called after Benjamin Franklin?
QALD2_te-67	Who are the parents of the wife of Juan Carlos I?
QALD2_te-72	In which U.S. state is Area 51 located?
QALD2_te-75	Which daughters of British earls died in the same place they were born in?
QALD2_te-76	List the children of Margaret Thatcher.
QALD2_te-8	To which countries does the Himalayan mountain

	system extend?
QALD2_te-81	Which books by Kerouac were published by Viking Press?
QALD2_te-84	Who created the comic Captain America?
QALD2_te-9	Give me a list of all trumpet players that were bandleaders.
QALD2_te-97	Who painted The Storm on the Sea of Galilee?
QALD2_tr-10	In which country does the Nile start?
QALD2_tr-21	Which states border Illinois?
QALD2_tr-22	In which country is the Limerick Lake?
QALD2_tr-3	Who is the daughter of Bill Clinton married to?
QALD2_tr-36	Through which countries does the Yenisei river flow?
QALD2_tr-41	Give me all soccer clubs in Spain.
QALD2_tr-42	What are the official languages of the Philippines?
QALD2_tr-43	Who is the mayor of New York City?
QALD2_tr-44	Who designed the Brooklyn Bridge?
QALD2_tr-54	Who was the wife of U.S. president Lincoln?
QALD2_tr-63	Give me all actors starring in Batman Begins.
QALD2_tr-65	Which companies work in the aerospace industry as well as on nuclear reactor technology?
QALD2_tr-70	Give me all films produced by Hal Roach.
QALD2_tr-77	Which music albums contain the song Last Christmas?
QALD2_tr-78	Give me all books written by Danielle Steel.
QALD2_tr-79	Which airports are located in California, USA?
QALD2_tr-8	Which states of Germany are governed by the Social Democratic Party?
QALD2_tr-85	In which films did Julia Roberts as well as Richard Gere play?
QALD2_tr-87	Who wrote the book The pillars of the Earth?
QALD2_tr-89	Give me all soccer clubs in the Premier League.
QALD2_tr-9	Which U.S. states possess gold minerals?
QALD2_tr-1	Give me all female Russian astronauts.
QALD2_tr-15	Who created Goofy?
QALD2_tr-16	Give me the capitals of all countries in Africa.
QALD2_tr-18	Which museum exhibits The Scream by Munch?
QALD2_tr-23	Which television shows were created by Walt Disney?
QALD2_tr-25	In which films directed by Garry Marshall was Julia Roberts starring?
QALD2_tr-28	Which European countries have a constitutional monarchy?
QALD2_tr-29	Which awards did WikiLeaks win?
QALD2_tr-31	What is the currency of the Czech Republic?
QALD2_tr-32	Which countries in the European Union adopted the Euro?

QALD2_tr-35	Who is the owner of Universal Studios?
QALD2_tr-38	Which monarchs of the United Kingdom were married to a German?
QALD2_tr-4	Which river does the Brooklyn Bridge cross?
QALD2_tr-45	Which telecommunications organizations are located in Belgium?
QALD2_tr-49	Give me all companies in the advertising industry.
QALD2_tr-52	Which presidents were born in 1945?
QALD2_tr-53	Give me all presidents of the United States.
QALD2_tr-55	Who developed the video game World of Warcraft?
QALD2_tr-6	Where did Abraham Lincoln die?
QALD2_tr-62	Who created Wikipedia?
QALD2_tr-64	Which software has been developed by organizations founded in California?
QALD2_tr-68	Which actors were born in Germany?
QALD2_tr-71	Give me all video games published by Mean Hamster Software.
QALD2_tr-72	Which languages are spoken in Estonia?
QALD2_tr-73	Who owns Aldi?
QALD2_tr-74	Which capitals in Europe were host cities of the summer olympic games?
QALD2_tr-80	Give me all Canadian Grunge record labels.
QALD2_tr-82	In which programming language is GIMP written?
QALD2_tr-83	Who produced films starring Natalie Portman?
QALD2_tr-84	Give me all movies with Tom Cruise.
QALD2_tr-86	Give me all female German chancellors.
SemSearch_LS-10	did nicole kidman have any siblings
SemSearch_LS-14	gods who dwelt on Mount Olympus
SemSearch_LS-17	houses of the Russian parliament
SemSearch_LS-19	kenya's captain in cricket
SemSearch_LS-2	Arab states of the Persian Gulf
SemSearch_LS-20	kublai khan siblings
SemSearch_LS-24	matt berry tv series
SemSearch_LS-29	nations where Portuguese is an official language
SemSearch_LS-3	astronauts who landed on the Moon
SemSearch_LS-30	orders (or 'choirs') of angels
SemSearch_LS-32	presidents depicted on mount rushmore who died of shooting
SemSearch_LS-34	ratt albums
SemSearch_LS-35	republics of the former Yugoslavia

SemSearch_LS-42	twelve tribes or sons of Israel
SemSearch_LS-43	what books did paul of tarsus write?
SemSearch_LS-44	what languages do they speak in afghanistan
SemSearch_LS-46	where the British monarch is also head of state
SemSearch_LS-5	books of the Jewish canon
SemSearch_LS-50	wonders of the ancient world
SemSearch_LS-7	Branches of the US military
SemSearch_LS-1	Apollo astronauts who walked on the Moon
SemSearch_LS-11	dioceses of the church of ireland
SemSearch_LS-12	first targets of the atomic bomb
SemSearch_LS-13	five great epics of Tamil literature
SemSearch_LS-16	hijackers in the September 11 attacks
SemSearch_LS-18	john lennon, parents
SemSearch_LS-21	lilly allen parents
SemSearch_LS-22	major leagues in the united states
SemSearch_LS-25	members of u2?
SemSearch_LS-26	movies starring erykah badu
SemSearch_LS-31	permanent members of the UN Security Council
SemSearch_LS-33	provinces and territories of Canada
SemSearch_LS-36	revolutionaries of 1959 in Cuba
SemSearch_LS-37	standard axioms of set theory
SemSearch_LS-38	states that border oklahoma
SemSearch_LS-39	ten ancient Greek city-kingdoms of Cyprus
SemSearch_LS-4	Axis powers of World War II
SemSearch_LS-41	the four of the companions of the prophet
SemSearch_LS-49	who invented the python programming language
SemSearch_LS-6	boroughs of New York City
SemSearch_LS-8	continents in the world
SemSearch_LS-9	degrees of Eastern Orthodox monasticism
TREC_Entity-11	Donors to the Home Depot Foundation.
TREC_Entity-14	Authors awarded an Anthony Award at Bouchercon in 2007.
TREC_Entity-20	Scotch whisky distilleries on the island of Islay.
TREC_Entity-4	Professional sports teams in Philadelphia.
TREC_Entity-5	Products of Medimmune, Inc.
TREC_Entity-6	Organizations that award Nobel prizes.
TREC_Entity-9	Members of The Beaux Arts Trio.
TREC_Entity-1	Carriers that Blackberry makes phones for.
TREC_Entity-10	Campuses of Indiana University.
TREC_Entity-12	Airlines that Air Canada has code share flights with.

TREC_Entity-15	Universities that are members of the SEC conference for football.
TREC_Entity-16	Sponsors of the Mancuso quilt festivals.
TREC_Entity-17	Chefs with a show on the Food Network.
TREC_Entity-18	Members of the band Jefferson Airplane.
TREC_Entity-19	Companies that John Hennessey serves on the board of.
TREC_Entity-2	Winners of the ACM Athena award.
TREC_Entity-7	Airlines that currently use Boeing 747 planes.

Table B.1.: Questions from ESQ workload

#	**Query**
1	Find novels written by Jane Austen.
2	Find football players who were FIFA Player of the Year.
3	Find companies acquired by Oracle.
4	Find films directed by Steven Spielberg.
5	Find US Open champions.
6	Find Turing Award recipients.
7	Find Eagles songs.
8	Find staring Robert De Niro.
9	Find computer game companies.
10	Find football players who transfered to Real Madrid.
11	Find capitals of states in the US.
12	Find persons born in Spain.
13	Find Celine Dion songs.
14	Find Pulitzer Prize for Drama winners.
15	Find province capitals in China.
16	Find companies in Silicon Valley.
17	Find Turing Award recipients who are affiliated with IBM.
18	Find US presidents who graduated from Harvard.
19	Find Brazilian football players who played for Real Madrid.
20	Find Australian actors who won some Best Actress award.
21	Find computer game companies acquired by Microsoft.
22	Find Nobel Prize winners and Big Ten universities. The winners held professorship in the universities.
23	Find films starring Robert De Niro and please tell directors of these films.
24	Find films and Australian actors. The film are an Academy Award winning film staring the actors.
25	Find companies and their founders. The company must be in

	Silicon Valley and the founders are Stanford University graduates.
26	Find football players and Italian football clubs. The player was an FIFA Player of the Year and joined the clubs sometime.
27	Find NBA champion teams and their leading players who won the NBA final MVP.
28	Find novels and their Academy Award winning film adaptations.

Table B.2.: Questions from ERQ workload

#	**Query**
1	Which albums won an award which Barbra Streisand was nominated for?
2	Who starred in a movie directed by Hal Ashby?
3	Who acted in films in which an actor starring in Kuffs acted?
4	Who starred in a movie directed by Guinevere Turner's writing partner?
5	Which awards were won by movies Dev Patel starred in?
6	Which film did the actor who directed The Mirror Has Two Faces star in?
7	Which awards were won by actors who starred in Charly?
8	Which rock band was the lead singer of Death Cab for Cutie a member of?
9	Who is the lieutenant governor of the province where Ottawa is located.
10	Archbishop of the fourth largest city of Germany.
11	Currency of the country whose president is James Mancham.
12	Who is the drummer for the band Brian Fair is a vocalist for?
13	Albums by the rock band Chad Smith is a drummer for?
14	Scientists who won the same award as Linus Pauling.
15	Physicist who won the same award as Paul Dirac.
16	Departments at the university where Ernest Rutherford worked.
17	Professors at the university where Enrico Fermi worked.
18	Computer scientists who are professors at the university where Frederick Terman was a professor.
19	Computer scientists who worked at an institute Albert Einstein worked at.
20	Computer scientists who are professors at the university the inventor of Pascal worked at.
21	Programming languages created by professors at the university the inventor of Pascal worked at.
22	Who founded the university which Randy Pausch taught at?
23	Computer scientists who worked at the same organization as Richard Hamming.
24	Programming language invented at the organization Richard Hamming worked at.
25	Who established an award won by The New Yorker?

26	Editors of magazines which won an award established by Joseph Pulitzer?
27	Awards won by magazines edited by William Shawn.
28	Who was married to a person who won the Nobel Peace Prize?
29	Awards the husband of Tipper Gore won?
30	Which author was the stepbrother of Jacqueline Kennedy Onassis?
31	Writers who won an award named after Hugo Gernsback
32	Which spiritual leader won the same award as a US vice president?
33	Computer games created by the same company that created Castle Crashers.
34	Who was the governor of the state Bernie Sanders represented?
35	Who directed a movie in which the actor who played Jack Sparrow acted?
36	Awards won by Bill Clinton's vice president.
37	Which director was married to an actor from the country where IKEA was founded?
38	Which German federal state borders a country that borders Italy?
39	Who is the chancellor of the university that awarded an honorary PhD to Nick Cave?
40	Kings of the city which led the Peloponnesian League.
41	Who is the mayor of the city where The Scream was stolen?
42	Governor of the state where Johns Hopkins University is based.
43	Who is married to an actor who went to an Ivy League university?
44	Which airports are named after a physicist who influenced Isaac Newton?
45	Which radio show is hosted by a graduate from an Ivy League university?
46	What Museums are located in the city where the Fritz Haber Institute of the MPG is located?
47	Who starred in a film in which Kevin Kline acted?
48	Who was the wife of the brother of Joseph Smith?
49	Where was the brother of Joseph Smith killed?
50	Who was the chief minister of the state whose capital is Chandigarh?
51	Which newspaper has writers who won a Pulitzer Prize?
52	Which universities were computer scientists who invented a programming language professors at?
53	Which football player played for a team in the city where Prada is located.
54	Which states were represented by senators who became presidents of the United States.
55	Which baseball players were born in the same city as a US president.
56	Which programming languages were invented by people who won the Turing Award.
57	Mayor of the largest city in Texas.

58	Computer games released by the same company that created The Legend of Zelda
59	Representatives from the same state as senator John McCain.
60	Movies by the director of Apocalypse Now
61	Economists influenced by Karl Marx
62	Computer games created by a subsidiary of Electronic Arts.
63	Which movie does the ex-wife of Brad Pitt star in?
64	Programming languages created by programmers who worked at Sun Microsystems.
65	Universities in the same city as Emory University.
66	Universities in the capital of Georgia (USA).
67	Companies founded by the creator of Star Wars.
68	Companies the creator of Tetris works at.
69	Companies headed by George Bush's vice president.
70	Who was the child of the successor of Neville Chamberlain?

Table B.3.: Questions used for COMPLEX benchmark

List of Figures

List of Tables

Bibliography

Adolphs, P., Theobald, M., Schäfer, U., Uszkoreit, H., and Weikum, G. (2011). YAGO-QA: Answering Questions by Structured Knowledge Queries. In *Proceedings of the 5th International Conference on Semantic Computing (ICSC 2011), Palo Alto, CA, USA, September 18-21, 2011*, pages 158–161.

Aggarwal, N. (2012). Cross Lingual Semantic Search by Improving Semantic Similarity and Relatedness Measures. In *Proceedings of the 11th International Semantic Web Conference (ISWC 2012), Boston, MA, USA, November 11-15, 2012*, pages 375–382.

Agichtein, E. and Gravano, L. (2000). *Snowball*: Extracting Relations from Large Plain-text Collections. In *Proceedings of the 5th Conference on Digital Libraries (DL 2000), San Antonio, Texas, USA*, pages 85–94.

Agrawal, S., Chaudhuri, S., and Das, G. (2002). DBXplorer: A System for Keyword-Based Search over Relational Databases. In *Proceedings of the 18th International Conference on Data Engineering (ICDE 2002), San Jose, CA, USA, February 26 - March 1, 2002*, pages 5–16.

Amer-Yahia, S., Koudas, N., Marian, A., Srivastava, D., and Toman, D. (2005). Structure and Content Scoring for XML. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB 2005), Trondheim, Norway, August 30 - September 2, 2005*, pages 361–372.

Amer-Yahia, S., Lakshmanan, L. V. S., and Pandit, S. (2004). FleXPath: Flexible Structure and Full-Text Querying for XML. In *Proceedings of the International Conference on Management of Data (SIGMOD 2004), Paris, France, June 13-18, 2004*, pages 83–94.

Angeli, G., Premkumar, M. J. J., and Manning, C. D. (2015). Leveraging Linguistic Structure For Open Domain Information Extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL-IJCNLP 2015), Beijing, China, July 26-31, 2015*, pages 344–354.

Anh, V. N. and Moffat, A. (2006). Pruned Query Evaluation Using Pre-computed Impacts. In *Proceedings of the 29th International Conference on Research and Development in Information Retrieval (SIGIR 2006), Seattle, WA, USA, August 6-11, 2006*, pages 372–379.

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. G. (2007). DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International Semantic Web Conference, and the 2nd Asian Semantic Web Conference (ISWC-ASWC 2007), Busan, Korea, November 11-15, 2007*, pages 722–735.

Baeza-Yates, R. A. and Ribeiro-Neto, B. A. (1999). *Modern Information Retrieval*. ACM Press / Addison-Wesley.

Balog, K., Bron, M., and de Rijke, M. (2011). Query Modeling for Entity Search Based on Terms, Categories, and Examples. *ACM Transactions on Information Systems*, 29(4):22.

Balog, K., Fang, Y., de Rijke, M., Serdyukov, P., and Si, L. (2012). Expertise Retrieval. *Foundations and Trends in Information Retrieval*, 6(2-3):127–256.

Balog, K. and Neumayer, R. (2013). A Test Collection for Entity Search in DBpedia. In *Proceedings of the 36th International Conference on Research and Development in Information Retrieval (SIGIR 2013), Dublin, Ireland, July 28 - August 01, 2013*, pages 737–740.

Banko, M., Cafarella, M. J., Soderland, S., Broadhead, M., and Etzioni, O. (2007). Open Information Extraction from the Web. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), Hyderabad, India, January 6-12, 2007*, pages 2670–2676.

Bast, H., Bäurle, F., Buchhold, B., and Haußmann, E. (2014). Semantic Full-text Search with Broccoli. In *Proceedings of the 37th International Conference on Research and Development in Information Retrieval (SIGIR 2014), Gold Coast , QLD, Australia, July 6-11, 2014*, pages 1265–1266.

Bast, H. and Buchhold, B. (2013). An Index for Efficient Semantic Full-text Search. In *Proceedings of the 22nd International Conference on Information and Knowledge Management (CIKM 2013), San Francisco, CA, USA, October 27 - November 1, 2013*, pages 369–378.

Beckett, D., Berners-Lee, T., Prud'hommeaux, E., and Carothers, G. (2014). RDF 1.1 Turtle. URL: `http://www.w3.org/TR/2014/REC-turtle-20140225/`.

Berant, J., Chou, A., Frostig, R., and Liang, P. (2013). Semantic Parsing on Freebase from Question-Answer Pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP 2013), Seattle, WA, USA, October 18-21, 2013*, pages 1533–1544.

Berant, J. and Liang, P. (2014). Semantic Parsing via Paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014), Baltimore, MD, USA, June 22-27, 2014*, pages 1415–1425.

Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., and Sudarshan, S. (2002). Keyword Searching and Browsing in Databases using BANKS. In *Proceedings of the 18th International Conference on Data Engineering (ICDE 2002), San Jose, CA, USA, February 26 - March 1, 2002*, pages 431–440.

Blessing, A. and Schütze, H. (2012). Crosslingual Distant Supervision for Extracting Relations of Different Complexity. In *Proceedings of the 21st International Conference on Information and Knowledge Management (CIKM 2012), Maui, HI, USA, October 29 - November 02, 2012*, pages 1123–1132.

Boldi, P., Bonchi, F., Castillo, C., and Vigna, S. (2011). Query Reformulation Mining: Models, Patterns, and Applications. *Information Retrieval*, 14(3):257–289.

Bollacker, K. D., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the International Conference on Management of Data (SIGMOD 2008), Vancouver, BC, Canada, June 10-12, 2008*, pages 1247–1250.

Brill, E., Dumais, S., and Banko, M. (2002). An Analysis of the AskMSR Question-Answering System. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002), Philadelphia, PA, USA, July 6-7, 2002*, pages 257–264.

Brin, S. (1998). Extracting Patterns and Relations from the World Wide Web. In *Proceedings of the 1998 International Workshop on the Web and Databases (WebDB 1998), Valencia, Spain, March 27-28, 1998*, pages 172–183.

Bron, M., Balog, K., and de Rijke, M. (2013). Example Based Entity Search in the Web of Data. In *Proceedings of the 35th European Conference on Information Retrieval (ECIR 2013), Moscow, Russia, March 24-27, 2013.*, pages 392–403.

Büttcher, S., Clarke, C. L. A., and Cormack, G. V. (2010). *Information Retrieval - Implementing and Evaluating Search Engines*. MIT Press.

Cabrio, E., Cojan, J., Aprosio, A. P., Magnini, B., Lavelli, A., and Gandon, F. (2012). QAKiS: an Open Domain QA System based on Relational Patterns. In *Proceedings of the 11th International Semantic Web Conference (ISWC 2012), Boston, USA, November 11-15, 2012*.

Cafarella, M. J., Halevy, A. Y., Wang, D. Z., Wu, E., and Zhang, Y. (2008). WebTables: Exploring the Power of Tables on the Web. *Proceedings of the VLDB Endowment*, 1(1):538–549.

Cai, Q. and Yates, A. (2013). Large-scale Semantic Parsing via Schema Matching and Lexicon Extension. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013), Sofia, Bulgaria, August 4-9, 2013*, pages 423–433.

Chambers, N., Cer, D., Grenager, T., Hall, D., Kiddon, C., MacCartney, B., de Marneffe, M.-C., Ramage, D., Yeh, E., and Manning, C. D. (2007). Learning Alignments and Leveraging Natural Logic. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 165–170, Prague.

Chaudhuri, S., Das, G., Hristidis, V., and Weikum, G. (2004). Probabilistic Ranking of Database Query Results. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB 2004), Toronto, Canada, August 31 - September 3, 2004*, pages 888–899.

Chu-Carroll, J., Fan, J., Boguraev, B., Carmel, D., Sheinwald, D., and Welty, C. (2012). Finding Needles in the Haystack: Search and Candidate Generation. *IBM Journal of Research and Development*, 56(3):6.

Clarke, J., Goldwasser, D., Chang, M., and Roth, D. (2010). Driving Semantic Parsing from the World's Response. In *Proceedings of the 14th Conference on Computational Natural Language Learning (CoNLL 2010), Uppsala, Sweden, July 15-16, 2010*, pages 18–27.

Cohen, S., Mamou, J., Kanza, Y., and Sagiv, Y. (2003). XSEarch: A Semantic Search Engine for XML. In *Proceedings of 29th International Conference on Very Large Data Bases (VLDB 2003), Berlin, Germany, September 9-12, 2003*, pages 45–56.

Corro, L. D. and Gemulla, R. (2013). ClausIE: Clause-based Open Information Extraction. In *Proceedings of the 22nd International World Wide Web Conference (WWW 2013), Rio de Janeiro, Brazil, May 13-17, 2013*, pages 355–366.

Croft, W. B., Metzler, D., and Strohman, T. (2009). *Search Engines - Information Retrieval in Practice*. Pearson Education.

Dalton, J., Dietz, L., and Allan, J. (2014). Entity Query Feature Expansion Using Knowledge Base Links. In *Proceedings of the 37th International Conference on Research and Development in Information Retrieval (SIGIR 2014), Gold Coast, QLD, Australia, July 6-11, 2014*, pages 365–374.

Dang, H. T., Kelly, D., and Lin, J. J. (2007). Overview of the TREC 2007 Question Answering Track. In *Proceedings of the 16th Text Retrieval Conference (TREC 2007), Gaithersburg, Maryland, USA, November 5-9, 2007*.

Downey, D., Broadhead, M., and Etzioni, O. (2007). Locating Complex Named Entities in Web Text. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), Hyderabad, India, January 6-12, 2007*, pages 2733–2739.

Elbassuoni, S. and Blanco, R. (2011). Keyword Search over RDF Graphs. In *Proceedings of the 20th International Conference on Information and Knowledge Management (CIKM 2011), Glasgow, United Kingdom, October 24-28, 2011*, pages 237–242.

Elbassuoni, S., Ramanath, M., Schenkel, R., Sydow, M., and Weikum, G. (2009). Language-model-based Ranking for Queries on RDF-graphs. In *Proceedings of the 18th International Conference on Information and Knowledge Management (CIKM 2009), Hong Kong, China, November 2-6, 2009*, pages 977–986.

Elbassuoni, S., Ramanath, M., and Weikum, G. (2011). Query Relaxation for Entity-Relationship Search. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC 2011), Heraklion, Crete, Greece, May 29 - June 2, 2011*, pages 62–76.

Epstein, E. A., Schor, M. I., Iyer, B., Lally, A., Brown, E. W., and Cwiklik, J. (2012). Making Watson Fast. *IBM Journal of Research and Development*, 56(3):15.

Etzioni, O., Banko, M., Soderland, S., and Weld, D. S. (2008). Open Information Extraction from the Web. *Communications of the ACM*, 51(12):68–74.

Etzioni, O., Cafarella, M. J., Downey, D., Kok, S., Popescu, A., Shaked, T., Soderland, S., Weld, D. S., and Yates, A. (2004). Web-scale Information Extraction in KnowItAll: (preliminary results). In *Proceedings of the 13th International World Wide Web Conference (WWW 2004), New York, NY, USA, May 17-20, 2004*, pages 100–110.

Fader, A., Soderland, S., and Etzioni, O. (2011). Identifying Relations for Open Information Extraction. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP 2011), Edinburgh, UK, July 27-31, 2011*, pages 1535–1545.

Fader, A., Zettlemoyer, L., and Etzioni, O. (2014). Open Question Answering over Curated and Extracted Knowledge Bases. In *Proceedings of the 20th International Conference on Knowledge Discovery and Data Mining (KDD 2014), New York, NY, USA - August 24 - 27, 2014*, pages 1156–1165.

Fader, A., Zettlemoyer, L. S., and Etzioni, O. (2013). Paraphrase-Driven Learning for Open Question Answering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013), 4-9 August 2013, Sofia, Bulgaria*, pages 1608–1618.

Fagin, R., Lotem, A., and Naor, M. (2003). Optimal Aggregation Algorithms for Middleware. *Journal of Computer and System Sciences*, 66(4):614–656.

Fan, J., Kalyanpur, A., Gondek, D., and Ferrucci, D. A. (2012). Automatic knowledge extraction from documents. *IBM Journal of Research and Development*, 56(3):5.

Fang, H. and Zhai, C. (2007). Probabilistic Models for Expert Finding. In *Proceedings of the 29th European Conference on Information Retrieval (ECIR 2007), Rome, Italy, April 2-5, 2007*, pages 418–430.

Faruqui, M. and Kumar, S. (2015). Multilingual Open Relation Extraction Using Cross-lingual Projection. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2015), Denver, Colorado, USA, May 31 - June 5, 2015*, pages 1351–1356.

Fellbaum, C. (1998). *WordNet An Electronic Lexical Database*. MIT Press.

Ferrucci, D. A. (2012). Introduction to "This is Watson". *IBM Journal of Research and Development*, 56(3):1.

Ferrucci, D. A., Brown, E. W., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A., Lally, A., Murdock, J. W., Nyberg, E., Prager, J. M., Schlaefer, N., and Welty, C. A. (2010). Building watson: An overview of the deepqa project. *AI Magazine*, 31(3):59–79.

Finkel, J. R., Grenager, T., and Manning, C. D. (2005). Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005), Ann Arbor, MI, USA June 25-30, 2005.*

Frank, A., Krieger, H., Xu, F., Uszkoreit, H., Crysmann, B., Jörg, B., and Schäfer, U. (2007). Question Answering from Structured Knowledge Sources. *Journal of Applied Logic*, 5(1):20–48.

Gabrilovich, E. and Markovitch, S. (2007). Computing Semantic Relatedness Using Wikipedia-based Explicit Semantic Analysis. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), Hyderabad, India, January 6-12, 2007*, pages 1606–1611.

Green, Jr, B. F., Wolf, A. K., Chomsky, C., and Laughery, K. (1961). Baseball: An Automatic Question-answerer. In *Proceedings of the 1961 IRE-AIEE-ACM Western Joint Computer Conference, New York, NY, USA*, pages 219–224.

Grishman, R. and Sundheim, B. (1996). Message Understanding Conference-6: A Brief History. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING 1996), Copenhagen, Denmark, August 5-9, 1996*, pages 466–471.

Guo, J., Xu, G., Cheng, X., and Li, H. (2009). Named Entity Recognition in Query. In *Proceedings of the 32nd International Conference on Research and Development in Information Retrieval (SIGIR 2009), Boston, MA, USA, July 19-23, 2009*, pages 267–274.

Gupta, R., Halevy, A. Y., Wang, X., Whang, S. E., and Wu, F. (2014). Biperpedia: An Ontology for Search Applications. *Proceedings of the VLDB Endowment*, 7(7):505–516.

Haghighi, A. and Klein, D. (2009). Simple Coreference Resolution with Rich Syntactic and Semantic Features. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP 2009), Singapore, August 6-7, 2009*, pages 1152–1161.

He, H., Wang, H., Yang, J., and Yu, P. S. (2007). BLINKS: Ranked Keyword Searches on Graphs. In *Proceedings of the International Conference on Management of Data (SIGMOD 2007), Beijing, China, June 12-14, 2007*, pages 305–316.

Hearst, M. A. (1992). Automatic Acquisition of Hyponyms from Large Text Corpora. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING 1992), Nantes, France, August 23-28, 1992*, pages 539–545.

Heath, T. and Bizer, C. (2011). *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers.

Hirschman, L. and Gaizauskas, R. J. (2001). Natural Language Question Answering: the View from Here. *Natural Language Engineering*, 7(4):275–300.

Hoffart, J. (2015). *Discovering and Disambiguating Named Entities in Text*. PhD thesis, Saarland University.

Hoffart, J., Milchevski, D., and Weikum, G. (2014a). AESTHETICS: Analytics with Strings, Things, and Cats. In *Proceedings of the 23rd International Conference on on Information and Knowledge Management (CIKM 2014), Shanghai, China, November 3-7, 2014*, pages 2018–2020.

Hoffart, J., Milchevski, D., and Weikum, G. (2014b). STICS: Searching with Strings, Things, and Cats. In *Proceedings of the 37th International Conference on Research and Development in Information Retrieval (SIGIR 2014), Gold Coast , QLD, Australia, July 6-11, 2014*, pages 1247–1248.

Hoffart, J., Suchanek, F. M., Berberich, K., and Weikum, G. (2013). YAGO2: A Spatially and Temporally Enhanced Knowledge Base From Wikipedia. *Artificial Intelligence*, 194:28–61.

Hoffart, J., Yosef, M. A., Bordino, I., Fürstenau, H., Pinkal, M., Spaniol, M., Taneva, B., Thater, S., and Weikum, G. (2011). Robust Disambiguation of Named Entities in Text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP 2011), Edinburgh, UK, July 27-31, 2011*, pages 782–792.

Hovy, E. H., Gerber, L., Hermjakob, U., Junk, M., and Lin, C. (2000). Question Answering in Webclopedia. In *Proceedings of the 9th Text REtrieval Conference (TREC 2000), Gaithersburg, Maryland, USA, November 13-16, 2000.*

Hovy, E. H., Hermjakob, U., and Ravichandran, D. (2002). A Question/Answer Typology with Surface Text Patterns. In *Proceedings of the 2nd International Conference on Human Language Technology Research (HLT 2002), San Diego, California, March 24-27, 2002*, pages 247–251.

Hristidis, V. and Papakonstantinou, Y. (2002). DISCOVER: keyword search in relational databases. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB 2002), Hong Kong, China, August 20-23, 2002*, pages 670–681.

Huang, J., Abadi, D. J., and Ren, K. (2011). Scalable SPARQL Querying of Large RDF Graphs. *Proceedings of the VLDB Endowment*, 4(11):1123–1134.

Ilyas, I. F., Aref, W. G., and Elmagarmid, A. K. (2003). Supporting Top-k Join Queries in Relational Databases. In *Proceedings of 29th International Conference on Very Large Data Bases (VLDB 2003), Berlin, Germany, September 9-12, 2003*, pages 754–765.

Ilyas, I. F., Beskales, G., and Soliman, M. A. (2008). A Survey of Top-k Query Processing Techniques in Relational Database Systems. *ACM Computing Surveys*, 40(4).

Järvelin, K. and Kekäläinen, J. (2002). Cumulated Gain-based Evaluation of IR Techniques. *ACM Transactions on Information Systems*, 20(4):422–446.

Johansson, R. and Nugues, P. (2008). The Effect of Syntactic Representation on Semantic Role Labeling. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING 2008), Manchester, UK, August 18-22, 2008*, pages 393–400.

Joshi, M., Sawant, U., and Chakrabarti, S. (2014). Knowledge Graph and Corpus Driven Segmentation and Answer Inference for Telegraphic Entity-seeking Queries. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), Doha, Qatar, October 25-29, 2014*, pages 1104–1114.

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Kalyanpur, A., Boguraev, B., Patwardhan, S., Murdock, J. W., Lally, A., Welty, C., Prager, J. M., Coppola, B., Fokoue-Nkoutche, A., Zhang, L., Pan, Y., and Qiu, Z. (2012). Structured Data and Inference in DeepQA. *IBM Journal of Research and Development*, 56(3):10.

Kalyanpur, A., Murdock, J. W., Fan, J., and Welty, C. A. (2011). Leveraging Community-Built Knowledge for Type Coercion in Question Answering. In *Proceedings of the 10th International Semantic Web Conference (ISWC 2011), Bonn, Germany, October 23-27, 2011*, pages 144–156.

Kasneci, G., Suchanek, F. M., Ifrim, G., Ramanath, M., and Weikum, G. (2008). NAGA: Searching and Ranking Knowledge. In *Proceedings of the 24th International Conference on Data Engineering (ICDE 2008), Cancún, México, April 7-12, 2008*, pages 953–962.

Katz, B. (1997). Annotating the World Wide Web using Natural Language. In *Computer-Assisted Information Retrieval (Recherche d'Information et ses Applications) - RIAO 1997, 5th International Conference, McGill University, Montreal, Canada, June 25-27, 1997. Proceedings*, pages 136–159.

Krishnamurthy, J. and Mitchell, T. M. (2015). Learning a Compositional Semantics for Freebase with an Open Predicate Vocabulary. *Transactions of the Association for Computational Linguistics*, 3:257–270.

Kwiatkowski, T., Zettlemoyer, L. S., Goldwater, S., and Steedman, M. (2010). Inducing Probabilistic CCG Grammars from Logical Form with Higher-Order Unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP 2010), Cambridge, MA, USA, October 9-11, 2010*, pages 1223–1233.

Kwok, C. C. T., Etzioni, O., and Weld, D. S. (2001). Scaling Question Answering to the Web. In *Proceedings of the 10th International World Wide Web Conference, (WWW 10), Hong Kong, China, May 1-5, 2001*, pages 150–161.

Lakoff, G. (1972). Linguistics and Natural Logic. In Davidson, D. and Harman, G., editors, *Semantics of Natural Language*, volume 40 of *Synthese Library*, pages 545–665. Springer Netherlands.

Lally, A., Prager, J. M., McCord, M. C., Boguraev, B., Patwardhan, S., Fan, J., Fodor, P., and Chu-Carroll, J. (2012). Question Analysis: How Watson Reads a Clue. *IBM Journal of Research and Development*, 56(3):2.

Lee, T., Wang, Z., Wang, H., and Hwang, S. (2013). Attribute Extraction and Scoring: A Probabilistic Approach. In *Proceedings of the 29th International Conference on Data Engineering (ICDE 2013), Brisbane, Australia, April 8-12, 2013*, pages 194–205.

Li, C., Yan, N., Roy, S. B., Lisham, L., and Das, G. (2010). Facetedpedia: Dynamic Generation of Query-dependent Faceted Interfaces for Wikipedia. In *Proceedings of the 19th International World Wide Web Conference (WWW 2010), Raleigh, NC, USA, April 26-30, 2010*, pages 651–660.

Li, F. and Jagadish, H. V. (2014). NaLIR: an Interactive Natural Language Interface for Querying Relational Databases. In *Proceedings of the International Conference on Management of Data (SIGMOD 2014), Snowbird, UT, USA, June 22-27, 2014*, pages 709–712.

Li, X., Li, C., and Yu, C. (2012). Entity-Relationship Queries over Wikipedia. *ACM Transactions on Information Systems*, 3(4):70.

Li, X. and Roth, D. (2002). Learning Question Classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002), Taipei, Taiwan, August 24 - September 1, 2002.*

Li, Y., Yang, H., and Jagadish, H. V. (2007). NaLIX: A generic natural language search environment for XML data. *ACM Transactions on Database Systems*, 32(4).

Liang, P., Jordan, M. I., and Klein, D. (2011). Learning Dependency-Based Compositional Semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011), Portland, Oregon, USA, June 19-24, 2011*, pages 590–599.

Liang, P. and Potts, C. (2015). Bringing Machine Learning and Compositional Semantics Together. *Annual Reviews of Linguistics*, 1(1):355–376.

Lin, J. J. (2007). An Exploration of the Principles Underlying Redundancy-based Factoid Question Answering. *ACM Transactions on Information Systems*, 25(2).

Lopez, V., Unger, C., Cimiano, P., and Motta, E. (2013). Evaluating Question Answering over Linked Data. *J.ournal of Web Semantics*, 21:3–13.

Madnani, N. and Dorr, B. J. (2010). Generating Phrasal and Sentential Paraphrases: A Survey of Data-Driven Methods. *Computational Linguistics*, 36(3):341–387.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press.

Marneffe, M., Maccartney, B., and Manning, C. (2006). Generating Typed Dependency Parses from Phrase Structure Parses. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC-2006)*, Genoa, Italy. European Language Resources Association (ELRA). ACL Anthology Identifier: L06-1260.

Mausam, Schmitz, M., Soderland, S., Bart, R., and Etzioni, O. (2012). Open Language Learning for Information Extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2012), Jeju Island, Korea, July 12-14, 2012*, pages 523–534.

McCord, M. C., Murdock, J. W., and Boguraev, B. (2012). Deep parsing in Watson. *IBM Journal of Research and Development*, 56(3):3.

Mendes, P. N., Daiber, J., Rajapakse, R. K., Sasaki, F., and Bizer, C. (2012). Evaluating the Impact of Phrase Recognition on Concept Tagging. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012), Istanbul, Turkey, May 23-25, 2012*, pages 1277–1280.

Metzler, D. and Croft, W. B. (2005). A Markov Random Field Model for Term Dependencies. In *Proceedings of the 28th International Conference on Research and Development in Information Retrieval (SIGIR 2005), Salvador, Brazil, August 15-19, 2005*, pages 472–479.

Meyers, A., Reeves, R., Macleod, C., Szekely, R., Zielinska, V., Young, B., and Grishman, R. (2004). Annotating Noun Argument Structure for NomBank. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004), Lisbon, Portugal, May 26-28, 2004*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *Computing Research Repository (CoRR)*, abs/1301.3781.

Milne, D. and Witten, I. H. (2008). An Effective, Low-Cost Measure of Semantic Relatedness Obtained From Wikipedia Links. In *in Proceeding of AAAI Workshop on Wikipedia and Artificial Intelligence: an Evolving Synergy, AAAI*, pages 25–30. Press.

Mintz, M., Bills, S., Snow, R., and Jurafsky, D. (2009). Distant Supervision for Relation Extraction without Labeled Data. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL-IJCNLP 2009), Singapore, August 2-7, 2009*, pages 1003–1011.

Mottin, D., Marascu, A., Roy, S. B., Das, G., Palpanas, T., and Velegrakis, Y. (2013). A Probabilistic Optimization Framework for the Empty-Answer Problem. *Proceedings of the VLDB Endowment*, 6(14):1762–1773.

Murdock, J. W., Kalyanpur, A., Welty, C., Fan, J., Ferrucci, D. A., Gondek, D., Zhang, L., and Kanayama, H. (2012). Typing Candidate Answers Using Type Coercion. *IBM Journal of Research and Development*, 56(3):7.

Nadeau, D. and Sekine, S. (2007). A Survey of Named Entity Recognition and Classification. *Linguisticae Investigationes*, 30(1):3–26.

Naisbitt, J. (1982). *Megatrends: Ten New Directions Transforming Our Lives.* Warner Books, Inc.

Nakashole, N., Theobald, M., and Weikum, G. (2011). Scalable Knowledge Harvesting with High Precision and High Recall. In *Proceedings of the 4th International Conference on Web Search and Web Data Mining (WSDM 2011), Hong Kong, China, February 9-12, 2011*, pages 227–236.

Nakashole, N., Weikum, G., and Suchanek, F. M. (2012). PATTY: A Taxonomy of Relational Patterns with Semantic Types. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2012), Jeju Island, Korea, July 12-14, 2012*, pages 1135–1145.

Neumann, T. and Weikum, G. (2008). RDF-3X: a RISC-style engine for RDF. *Proceedings of the VLDB Endowment*, 1(1):647–659.

Nie, Z., Ma, Y., Shi, S., Wen, J., and Ma, W. (2007). Web Object Retrieval. In *Proceedings of the 16th International World Wide Web Conference (WWW 2007), Banff, Alberta, Canada, May 8-12, 2007*, pages 81–90.

Pasca, M. and Durme, B. V. (2007). What You Seek Is What You Get: Extraction of Class Attributes from Query Logs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), Hyderabad, India, January 6-12, 2007*, pages 2832–2837.

Pasca, M. and Durme, B. V. (2008). Weakly-Supervised Acquisition of Open-Domain Classes and Class Attributes from Web Documents and Query Logs. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL 2008), Columbus, OH, USA, June 15-20, 2008*, pages 19–27.

Pasca, M. and Harabagiu, S. M. (2001). High Performance Question/Answering. In *Proceedings of the 24th International Conference on Research and Development in Information Retrieval (SIGIR 2001), New Orleans, Louisiana, USA, September 9-13, 2001*, pages 366–374.

Ponte, J. M. and Croft, W. B. (1998). A Language Modeling Approach to Information Retrieval. In *Proceedings of the 21st International Conference on Research and Development in Information Retrieval (SIGIR 1998), Melbourne, Australia, August 24-28, 1998*, pages 275–281.

Popescu, A., Etzioni, O., and Kautz, H. A. (2003). Towards a Theory of Natural Language Interfaces to Databases. In *Proceedings of the 2003 International Conference on Intelligent User Interfaces (IUI 2003), Miami, FL, USA, January 12-15, 2003*, pages 149–157.

Pound, J., Hudek, A. K., Ilyas, I. F., and Weddell, G. E. (2012). Interpreting Keyword Queries over Web Knowledge Bases. In *Proceedings of the 21st International International Conference on Information and Knowledge Management (CIKM 2012), Maui, HI, USA, October 29 - November 2, 2012*, pages 305–314.

Pound, J., Ilyas, I. F., and Weddell, G. E. (2010a). Expressive and Flexible Access to Web-extracted Data: a Keyword-based Structured Query Language. In *Proceedings of the International Conference on Management of Data (SIGMOD 2010), Indianapolis, IN, USA, June 6-10, 2010*, pages 423–434.

Pound, J., Mika, P., and Zaragoza, H. (2010b). Ad-hoc Object Retrieval in the Web of Data. In *Proceedings of the 19th International World Wide Web Conference (WWW 2010), Raleigh, NC, USA, April 26-30, 2010*, pages 771–780.

QALD-2 (2012). Second Workshop on Question Answering over Linked Data (QALD-2). URL: `http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=challenge&q=2`.

Ravichandran, D. and Hovy, E. H. (2002). Learning Surface Text Patterns for a Question Answering System. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002), Philadelphia, PA, USA, July 6-12, 2002*, pages 41–47.

Ritter, A., Clark, S., Mausam, and Etzioni, O. (2011). Named Entity Recognition in Tweets: An Experimental Study. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP 2011), Edinburgh, UK, July 27-31, 2011*, pages 1524–1534.

Sagiv, Y. (2013). A Personal Perspective on Keyword Search over Data Graphs. In *Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 21–32.

Salton, G., Allan, J., and Buckley, C. (1993). Approaches to Passage Retrieval in Full Text Information Systems. In *Proceedings of the 16th International Conference on Research and Development in Information Retrieval (SIGIR 1993), Pittsburgh, PA, USA, June 27 - July 1, 1993*, pages 49–58.

Sang, E. F. T. K. and Meulder, F. D. (2003). Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of the 7th Conference on Natural Language Learning, (CoNLL 2003), Edmonton, Canada, May 31 - June 1, 2003*, pages 142–147.

Sawant, U. and Chakrabarti, S. (2013). Learning Joint Query Interpretation and Response Ranking. In *Proceedings of the 22nd International World Wide Web Conference (WWW 2013), Rio de Janeiro, Brazil, May 13-17, 2013*, pages 1099–1110.

Schuhmacher, M. and Ponzetto, S. P. (2014). Knowledge-based Graph Document Modeling. In *Proceedings of the 7th International Conference on Web Search and Data Mining (WSDM 2014), New York, NY, USA, February 24-28, 2014*, pages 543–552.

Serdyukov, P. and Hiemstra, D. (2008). Modeling Documents as Mixtures of Persons for Expert Finding. In *Proceedings of the 30th European Conference on Information Retrieval (ECIR 2008), Glasgow, UK, March 30 - April 3, 2008*, pages 309–320.

Sowa, J. F. (2000). *Knowledge Representation: Logical, Philosophical and Computational Foundations*. Brooks/Cole Publishing Co., Pacific Grove, CA, USA.

Srihari, R. K. and Li, W. (1999). Information Extraction Supported Question Answering. In *Proceedings of the 8th Text REtrieval Conference (TREC 1999), Gaithersburg, MD, USA, November 17-19, 1999*.

Stanovsky, G. and Dagan, I. (2015). Open IE as an Intermediate Structure for Semantic Tasks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL-IJCNLP 2015), Beijing, China, July 26-31, 2015*, pages 303–308.

Suchanek, F. M. (2009). *Automated Construction and Growth of a Large Ontology.* PhD thesis, Saarland University.

Suchanek, F. M., Kasneci, G., and Weikum, G. (2007). Yago: a Core of Semantic Knowledge. In *Proceedings of the 16th International World Wide Web Conference (WWW 2007), Banff, Alberta, Canada, May 8-12, 2007*, pages 697–706.

Suchanek, F. M., Sozio, M., and Weikum, G. (2009). SOFIE: a self-organizing framework for information extraction. In *Proceedings of the 18th International World Wide Web Conference (WWW 2009), Madrid, Spain, April 20-24, 2009*, pages 631–640.

Tellex, S., Katz, B., Lin, J. J., Fernandes, A., and Marton, G. (2003). Quantitative Evaluation of Passage Retrieval Algorithms for Question Answering. In *Proceedings of the 26th International Conference on Research and Development in Information Retrieval (SIGIR 2003), Toronto, Canada, July 28 - August 1, 2003*, pages 41–47.

Tesauro, G., Gondek, D., Lenchner, J., Fan, J., and Prager, J. M. (2012). Simulation, Learning, and Optimization Techniques in Watson's Game Strategies. *IBM Journal of Research and Development*, 56(3):16.

Theobald, M., Bast, H., Majumdar, D., Schenkel, R., and Weikum, G. (2008). TopX: Efficient and Versatile Top-k Query Processing for Semistructured Data. *VLDB Journal*, 17(1):81–115.

Theobald, M., Schenkel, R., and Weikum, G. (2005). Efficient and Self-tuning Incremental Query Expansion for Top-k Query Processing. In *Proceedings of the 28th ACM International Conference on Research and Development in Information Retrieval (SIGIR 2005), Salvador, Brazil, August 15-19, 2005*, pages 242–249.

Tran, T., Wang, H., Rudolph, S., and Cimiano, P. (2009). Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data. In *Proceedings of the 25th International Conference on Data Engineering (ICDE 2009), Shanghai, China, March 29 - April 2, 2009*, pages 405–416.

Tummarello, G., Cyganiak, R., Catasta, M., Danielczyk, S., Delbru, R., and Decker, S. (2010). Sig.ma: Live Views on the Web of Data. In *Proceedings of the 19th International World Wide Web Conference (WWW 2010), Raleigh, NC, USA, April 26-30, 2010*, pages 1301–1304.

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, 59:433–460.

Tylenda, T., Kondreddi, S. K., and Weikum, G. (2014). Spotting Knowledge Base Facts in Web Texts. In *Proceedings of the 4th Workshop on Automated Knowledge Base Construction (AKBC 2014), Montreal, Canada, December 13, 2014.*

Unger, C., Bühmann, L., Lehmann, J., Ngomo, A. N., Gerber, D., and Cimiano, P. (2012a). Template-based question answering over RDF data. In *Proceedings of the 21st World Wide Web Conference (WWW 2012), Lyon, France, April 16-20, 2012*, pages 639–648.

Unger, C. and Cimiano, P. (2011). Pythia: Compositional Meaning Construction for Ontology-Based Question Answering on the Semantic Web. In *Proceedings of the 16th International Conference on Applications of Natural Language to Information Systems (NLDB 2011), Alicante, Spain, June 28-30, 2011*, pages 153–160.

Unger, C., Cimiano, P., Lopez, V., Motta, E., Buitelaar, P., and Cyganiak, R., editors (2012b). *The Workshop Interacting with Linked Data (ILD).*

Vallet, D. and Zaragoza, H. (2008). Inferring the most Important Types of a Query: a Semantic Approach. In *Proceedings of the 31st International Conference on Research and Development in Information Retrieval (SIGIR 2008), Singapore, July 20-24, 2008*, pages 857–858.

Venetis, P., Halevy, A. Y., Madhavan, J., Pasca, M., Shen, W., Wu, F., Miao, G., and Wu, C. (2011). Recovering Semantics of Tables on the Web. *Proceedings of the VLDB Endowment*, 4(9):528–538.

Voorhees, E. M. (1994). Query Expansion Using Lexical-Semantic Relations. In *Proceedings of the 17th International Conference on Research and Development in Information Retrieval (SIGIR 1994). Dublin, Ireland, July 3-6, 1994*, pages 61–69.

Wang, Y., Berant, J., and Liang, P. (2015). Building a Semantic Parser Overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, (ACL-IJCNLP 2015), Beijing, China, July 26-31, 2015*, pages 1332–1342.

Weissenborn, D., Hennig, L., Xu, F., and Uszkoreit, H. (2015). Multi-Objective Optimization for the Joint Disambiguation of Nouns and Named Entities. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL-IJCNLP 2015), Beijing, China, July 26-31, 2015*, pages 596–605.

Wood, P. T. (2012). Query Languages for Graph Databases. *SIGMOD Record*, 41(1):50–60.

Woods, W. A., Kaplan, R., and Webber, N. B. (1972). The LUNAR Sciences Natural Language Information System: Final Report. Technical Report BBN Report No. 2378, Bolt Beranek and Newman, Cambridge, Massachusetts.

Wu, F. and Weld, D. S. (2010). Open Information Extraction Using Wikipedia. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010), Uppsala, Sweden, July 11-16, 2010*, pages 118–127.

Xu, F., Uszkoreit, H., and Li, H. (2007). A Seed-driven Bottom-up Machine Learning Framework for Extracting Relations of Various Complexity. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL 2007), Prague, Czech Republic, June 23-30, 2007*.

Xu, J. and Croft, W. B. (1996). Query Expansion Using Local and Global Document Analysis. In *Proceedings of the 19th International Conference on Research and Development in Information Retrieval (SIGIR 1996), Zurich, Switzerland, August 18-22, 1996*, pages 4–11.

Yahya, M., Barbosa, D., Berberich, K., Wang, Q., and Weikum, G. (2016). Relationship Queries on Extended Knowledge Graphs. In *Proceedings of the 9th International Conference on Web Search and Data Mining (WSDM 2016), San Francisco, CA, USA, February 22-25, 2016*.

Yahya, M., Berberich, K., Elbassuoni, S., Ramanath, M., Tresp, V., and Weikum, G. (2012a). Deep Answers for Naturally Asked Questions on the Web of Data. In *Proceedings of the 21st World Wide Web Conference (WWW 2012), Lyon, France, April 16-20, 2012*, pages 445–449.

Yahya, M., Berberich, K., Elbassuoni, S., Ramanath, M., Tresp, V., and Weikum, G. (2012b). Natural language questions for the web of data. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2012), Jeju Island, Korea, July 12-14, 2012*, pages 379–390.

Yahya, M., Berberich, K., Elbassuoni, S., and Weikum, G. (2013). Robust question answering over the web of linked data. In *Proceedings of the 22nd International Conference on Information and Knowledge Management (CIKM 2013), San Francisco, CA, USA, October 27 - November 1, 2013*, pages 1107–1116.

Yahya, M., Whang, S., Gupta, R., and Halevy, A. Y. (2014). ReNoun: Fact Extraction for Nominal Attributes. In *Proceedings of the 2014 Conference on Empirical Methods* *in Natural Language Processing (EMNLP 2014), Doha, Qatar, October 25-29, 2014*, pages 325–335.

Yang, S., Wu, Y., Sun, H., and Yan, X. (2014). Schemaless and Structureless Graph Querying. *Proceedings of the VLDB Endowment*, 7(7):565–576.

Yates, A., Banko, M., Broadhead, M., Cafarella, M. J., Etzioni, O., and Soderland, S. (2007). TextRunner: Open Information Extraction on the Web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations (NAACL-Demonstrations 2007), Rochester, NY, USA, April 22-27, 2007*, pages 25–26.

Yu, J. X., Qin, L., and Chang, L. (2009). *Keyword Search in Databases.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

Zelle, J. M. and Mooney, R. J. (1996). Learning to Parse Database Queries Using Inductive Logic Programming. In *Proceedings of the 13th National Conference on Artificial Intelligence and 8th Innovative Applications of Artificial Intelligence Conference, (AAAI-IAAI 1996), Portland, OR, USA August 4-8, 1996*, pages 1050–1055.

Zettlemoyer, L. S. and Collins, M. (2007). Online Learning of Relaxed CCG Grammars for Parsing to Logical Form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2007), Prague, Czech Republic, June 28-30, 2007*, pages 678–687.

Zettlemoyer, L. S. and Collins, M. (2009). Learning Context-Dependent Mappings from Sentences to Logical Form. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL-IJCNLP 2009), Singapore, August 2-7, 2009*, pages 976–984.

Zhai, C. (2008). *Statistical Language Models for Information Retrieval.* Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Zhai, C. and Lafferty, J. D. (2004). A Study of Smoothing Methods for Language Models Applied to Information Retrieval. *ACM Transactions on Information Systems*, 22(2):179–214.

Zhou, X., Gaugaz, J., Balke, W., and Nejdl, W. (2007). Query Relaxation using Malleable Schemas. In *Proceedings of the International Conference on Management of Data (SIGMOD 2007), Beijing, China, June 12-14, 2007*, pages 545–556.