Interpreting and Answering Keyword Queries using Web Knowledge Bases

by

Jeffrey Pound

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2013

Abstract

Many keyword queries issued to Web search engines target information about real world entities, and interpreting these queries over Web knowledge bases can allow a search system to provide exact answers to keyword queries. Such an ability provides a useful service to end users, as their information need can be directly addressed and they need not scour textual results for the desired information. However, not all keyword queries can be addressed by even the most comprehensive knowledge base, and therefore equally important is the problem of recognizing when a reference knowledge base is not capable of modelling the keyword query's intention. This may be due to lack of coverage of the knowledge base or lack of expressiveness in the underlying query representation formalism.

This thesis presents an approach to computing structured representations of keyword queries over a reference knowledge base. Keyword queries are annotated with occurrences of semantic constructs by learning a sequential labelling model from an annotated Web query log. Frequent query structures are then mined from the query log and are used along with the annotations to map keyword queries into a structured representation over the vocabulary of a reference knowledge base. The proposed approach exploits coarse linguistic structure in keyword queries, and combines it with rich structured query representations of information needs.

As an intermediate representation formalism, a novel query language is proposed that blends keyword search with structured query processing over large Web knowledge bases. The formalism for structured keyword queries combines the flexibility of keyword search with the expressiveness of structures queries. A solution to the resulting disambiguation problem caused by introducing keywords as primitives in a structured query language is presented. Expressions in our proposed language are rewritten using the vocabulary of the knowledge base, and different possible rewritings are ranked based on their syntactic relationship to the keywords in the query as well as their semantic coherence in the underlying knowledge base.

The problem of ranking knowledge base entities returned as a query result is also explored from the perspective of personalized result ranking. User interest models based on entity types are learned from a Web search session by cross referencing clicks on URLs with known entity homepages. The user interest model is then used to effectively rerank answer lists for a given user. A methodology for evaluating entity-based search engines is also proposed and empirically evaluated.

Acknowledgements

I would like to thank my supervisors Grant Weddell and Ihab Ilyas for their support and guidance, and for allowing me the independence to explore my research interests. I would like to thank the many collaborators I have worked with over the years: David Toman, Jiewen Wu, Alexander Hudek, Peter Mika, Roi Blanco, Hugo Zaragoza, Stelios Paparizos, Panayiotis Tsaparas, and others. I would also like to thank my committee members: Tamer Özsu, Ming Li, Lukasz Golab, and Nick Koudas for taking the time to read and critique my thesis.

I would also like to thank the many friends I've met during my studies. It's the (more than occasional) beer and coffee with friends that keep a person sane throughout a PhD.

Most of all I want to thank my family from the bottom of my heart. My parents Linda and Steve, my brother Brad; mi suegra Monica, suegro Horacia, hermano Alejandro y hermanita Macarena; my wife Sol and beautiful baby daughter Sofia. You have all always stood by me and believed in me, I would not be where I am today if it wasn't for your support. *Muchas gracias.*

Dedication

To my beloved wife Sol and my beautiful daughter Sofia…

Table of Contents

List of Figures

Chapter 1

Introduction

If Edison had a needle to find in a haystack, he would proceed at once with the diligence of the bee to examine straw after straw until he found the object of his search... I was a sorry witness of such doings, knowing that a little theory and calculation would have saved him ninety per cent of his labor.

Nikola Tesla (1857 - 1943),
New York Times, October 19, 1931

1.1 Search in Information Systems

Consider a person who searches an information system using the keywords *"toronto restaurants"*. The person types their query and clicks the search button, anxiously awaiting their results. But what information should a search system return in order to satisfy this user's information need? Certainly a Web search engine, such as Google or Bing, has many choices. There are millions of documents on the Web that mention the terms *"toronto"* and *"restaurant"*[1]. There are also structured databases encoding information about various restaurants, some of which are likely located in the city of Toronto. Whether a search engine returns documents relevant to the query terms or structured results showing restaurants in Toronto, we cannot say that the search engine has done something right or wrong. Since the intent of the query is underspecified – that is to say, we cannot know exactly what it is the user is looking for due to the inherent ambiguity of keyword queries – there

[1]According to http://www.google.com, there are 156,000,000 documents matching the query *"toronto restaurant"* at the time of writing. Retrieved April 27, 2013

is no definition of a “correct” query answer. However, it seems clear that some answers may be better than others. Results providing information about restaurants in Toronto would be considered more *relevant* than results that coincidently mention those two terms together, but do not provide information explicitly about restaurants that are located in Toronto.

1.1.1 Query Interpretation in Search Systems

The nature of search in information systems – be it the traditional information retrieval that searches document collections given keyword queries, or more recent adaptations of search that integrate large structured data repositories – is one of uncertainty and ambiguity. Results are judged by their *quality*, not by a formal definition of correctness. At the time of writing, a Bing[2] search for our example query *“toronto restaurants”* returns a list of documents relevant to the query terms. Among them are various Web pages describing restaurants in Toronto. A Google[3] search for the same query terms however, produces a fundamentally different search result: a list of structured data items, each item describing a particular *entity* that is a type of `RESTAURANT` with a *location* of `Toronto`. The difference in the two results is not about access to data, indeed a search initiated from Bing’s local page[4] returns a similar structured result as Google. The difference is in the *understanding* and *representation* of the query. While Bing views this query as a collection of terms describing textual documents of interest, Google views the query as having a more explicit structure: a search for particular restaurant entities that are located in Toronto.

Special handling of queries for vertical domains is a common way of integrating domain knowledge into search engines. Examples include the local business search provided by major search engines, which aims to interpret queries over locations and business types; the book search facility provided by Amazon, which aims to interpret queries over author names, book titles, and ISBN numbers; or the job search from Monster.com, which aims to interpret queries over locations, companies, and job types. While building domain specific vertical search engines achieves the goal of integrating domain knowledge into the search process, it comes at the expense of engineering a specific solution for each target domain.

[2]www.bing.com
[3]www.google.com
[4]www.bing.com/local

1.1.2 Semantic Search

A general search architecture that can integrate arbitrary external knowledge sources, interpret keyword queries over these knowledge sources, and provide exact answers to keyword queries is a highly desirable goal, and the focus of this thesis. This process is often referred to as the *semantic search problem* – the problem of integrating reference semantic information in the search process. A semantic search system allows the integration of domain knowledge into the search process in a generic way. For example, a product catalogue could be used as a reference knowledge base, allowing queries to be interpreted in terms of products (attributes of products, brands, product types, etc...) enabling a powerful vertical product search with an underlying understanding of the concepts and entities in the reference knowledge. A query for *"10mp canon digital cameras"* could be modelled as a query for entities with a *megapixel* value of 10, the brand `Canon`, and an entity type `DIGITAL_CAMERA`. This semantic, or *structural*, understanding of the query allows the retrieval of specific entities and information about those entities.

The semantic search approach also allows one to incorporate general knowledge, or *fact collections*, into an open domain search engine as well. Similar to recent advances in the Google search engine[5] which aims to answer simple factual queries like birthdays of famous people or authors of popular books, a semantic search engine allows the integration of large amounts of reference knowledge in the search process. The potential for semantic search however is much greater than answering simple factual queries, as formal representations of knowledge and queries can enable more complex query interpretations. For example, a query for *"grammy award winning guitarists"* requires not only a structural understanding of the query (i.e., entities of type `GUITARIST` that have *hasWonPrize* relation to entities of type `GRAMMY_AWARD`), but also complex reasoning over the knowledge base to determine if an entity can be inferred to be a `GUITARIST` that has won some entity that can be inferred to be a type of `GRAMMY_AWARD`.

Modelling possible query interpretations also poses a difficult problem. Following our previous example query *"grammy award winning guitarists"*, how does a search system identify that *"winning"* denotes the relation *hasWonPrize*? Similarly, for the query *"toronto restaurants"* having an interpretation of `RESTAURANTS` that are *locatedIn* the city of `Toronto`, a search system must infer the relationship *locatedIn* which is not explicitly represented in the query. Furthermore, even if a query understanding algorithm can compute probable mappings of query terms to knowledge base items, how can a system determine that a query is looking for entities of type `RESTAURANT` that are *locatedIn* `Toronto`, as

[5]See for example, the Google result of the query *"moby dick author"* which provides the exact answer to the query, "Herman Melville". Retrieved April 27, 2013.

opposed to the query looking for information about the entity `Toronto` which is the location of various `RESTAURANT`s? As queries become more complex, the problem of modelling their interpretations becomes more complex as well. We discuss the problem of search in information systems, from the point of view of query understanding and knowledge incorporation in Chapters 2 and 3.

1.2 Representing Knowledge

In this section we describe the data model used for representing factual world knowledge about real-world entities, as well as linguistic knowledge that describes how the terms used to encode a knowledge base are related to each other. The world knowledge acts as a repository of searchable information, while the linguistic knowledge determines how query terms can match against knowledge base items. We adopt of formal model of knowledge representation based on an existing dialect of description logics, allowing us to give explicit semantics to the often vaguely used terms *knowledge base*, *knowledge graph*, and *concept*.

1.2.1 Representing World Knowledge

We represent factual world knowledge as a collection of assertions about real-world entities. Such assertions encode relationships between entities, such as

$$\langle \texttt{Jimi\_Hendrix},\ \textit{hasWonPrize},\ \texttt{Grammy\_Lifetime\_Achievement\_Award} \rangle,$$

as well as relationships between entities and the types they have, for example

$$\langle \texttt{Jimi\_Hendrix},\ \textit{type},\ \texttt{GUITARIST} \rangle,$$

and sub-class relationships between types

$$\langle \texttt{GUITARIST},\ \textit{subClassOf},\ \texttt{MUSICIAN} \rangle.$$

Knowledge bases are often represented as collection of subject, predicate, object triples following the RDF model. However RDF alone is not sufficient to describe the underlying semantics of the data. Missing are constructs that allow one to express semantics such as class hierarchies. There are a variety of proposals to extend triple-based models with richer schema-level semantics such as RDFS [15] and the family of OWL languages [120]. We adopt a formal model of knowledge representation as a collection of assertions based on the OWL2 EL Profile [6] extended with unary *entity* sets (allowing an entity to represent a class containing only itself), with attributes that map to values in a concrete domain such as strings or dates, and with inverse relations. This model of knowledge representation allows us the expressiveness to encode a wide variety of knowledge sources (including many publicly available knowledge bases), formalize constraints in the data as inclusion dependencies, and issue expressive concept-based search queries against the knowledge base. At the same time, the OWL2 EL Profile is simple enough to allow efficient concept-based search query processing over very large data sets with query-time inference.

The syntax and semantics of concept expressions are given as follows.

Definition 1 (Concept) Let $\{A, A_1, \ldots\}$, $\{R, R_1, \ldots\}$, $\{f, f_1, \ldots\}$, and $\{e, e_1, \ldots\}$ be countably infinite and disjoint sets of entity type names (a.k.a, entity class or primitive concept), relation names, attribute names, and entity names respectively. Let R^- denote the inverse of relation R, and f^- denote the inverse of attribute f. A *concept* is given by the following.

$$\begin{array}{rcl} C & ::= & A \\ & \mid & \exists R(C) \\ & \mid & \exists R^-(C) \\ & \mid & f(k) \\ & \mid & f^-(C) \\ & \mid & C_1 \sqcap C_2 \\ & \mid & \{e\} \end{array}$$

Instances of concepts are specified using a *knowledge representation interpretation function* (KR-interpretation) which maps concept expressions to sets of items in given domains. An *KR-interpretation* $\mathcal{I}$ is a 2-tuple $(\triangle \ \uplus \ \mathbb{S}, \cdot^{\mathcal{I}})$ in which $\triangle$ is an *abstract domain* of entities and $\mathbb{S}$ is a disjoint *concrete domain*, and $\cdot^{\mathcal{I}}$ a *KR-interpretation function*. The KR-interpretation function $\cdot^{\mathcal{I}}$ maps

- each concept name A to a set $(A)^{\mathcal{I}} \subseteq \triangle$;
- each relation name R to a relation $(R)^{\mathcal{I}} \subseteq (\triangle \times \triangle)$;
- each attribute f to a relation $(f)^{\mathcal{I}} \subseteq (\triangle \times \mathbb{S})$;

[6] http://www.w3.org/TR/owl-profiles/#OWL_2_EL

- each nominal entity set $\{e\}$ to itself $(\{e\})^{\mathcal{I}} = \{e\} \subseteq \triangle$;
- a finite string k to itself $(k)^{\mathcal{I}} = k \in \mathbb{S}$;

The KR-interpretation function is extended to all concepts as follows.

- $(C \sqcap D)^{\mathcal{I}} = (C)^{\mathcal{I}} \cap (D)^{\mathcal{I}}$
- $(\exists R.C)^{\mathcal{I}} = \{x \in \triangle \mid \exists y \in (C)^{\mathcal{I}} : (x, y) \in (R)^{\mathcal{I}}\}$
- $(\exists R^{-}.C)^{\mathcal{I}} = \{x \in \triangle \mid \exists y \in (C)^{\mathcal{I}} : (y, x) \in (R)^{\mathcal{I}}\}$
- $(f(k))^{\mathcal{I}} = \{x \in \triangle \mid (x, k) \in (f)^{\mathcal{I}}\}$
- $(f^{-}(C))^{\mathcal{I}} = \{k \in \mathbb{S} \mid \exists x \in (C)^{\mathcal{I}} : (x, k) \in (f)^{\mathcal{I}}\}$

□

An example of an entity type is `GUITARIST`, which denotes the set of all entities that are a type of guitarist. An example of a relation is *bornIn*, which maps entities to the places in which they were born. The inverse *bornIn*$^{-}$ maps places of birth to entities (which intuitively would represent a "birthplace of" relation). An example entity is `Jimi_Hendrix`, with an attribute *dateOfBirth* that maps `Jimi_Hendrix` to the constant *"1942-11-27"*. The inverse attribute *dateOfBirth*$^{-}$ would map the concrete value *"1942-11-27"* to the set of all entities known to be born on that day.

We allow all constructs to be used when defining concepts for queries (see Section 1.3), and omit use of inverses and nominals when defining knowledge bases to ensure tractability of reasoning.

Definition 2 (Knowledge Base Constraint) Let C_1 and C_2 be concepts free of nominals and inverses, A be an entity type, R a relation, f an attribute, e an entity and k is an element of a concrete domain. An *inclusion dependency* is expressed as follows.

$$C_1 \sqsubseteq C_2$$

An *assertion* has one of the following forms.

$$A(e) \;\mid\; R(e_1, e_2) \;\mid\; f(e, k)$$

A KR-interpretation $\mathcal{I}$ satisfies a knowledge base constraint of type:

- inclusion dependency $C_1 \sqsubseteq C_2$ if $(C_1)^{\mathcal{I}} \subseteq (C_2)^{\mathcal{I}}$;
- entity type assertion $A(e)$ if $e \in (A)^{\mathcal{I}}$;
- relation assertion $R(e_1, e_2)$ if $(e_1, e_2) \in (R)^{\mathcal{I}}$; and
- attribute assertion $f(e, k)$ if $(e, k) \in (f)^{\mathcal{I}}$.

$\square$

An inclusion dependency describes a subset relationship between all of the entities in the KR-interpretation of the left hand side, and all of the entities in the KR-interpretation of the right hand side. This construct can be used to encode subclass hierarchies, for example

$$\texttt{GUITARIST} \sqsubseteq \texttt{MUSICIAN}$$

expresses the fact that all guitarists are also musicians. Inclusion dependencies can also describe more complex dependencies, for example the inclusion dependency

$$\texttt{ARTIST} \sqsubseteq \texttt{PERSON} \sqcap \exists \textit{created}(\texttt{CREATIVE\_WORK})$$

defines the concept of an artist as any person that has created some thing considered to be a creative work.

Facts about entities are represented as assertion constraints. The first assertion constraint asserts that an entity e has type A, e.g., $\texttt{GUITARIST}(\texttt{Jimi\_Hendrix})$. The second assertion constraint asserts that a relation R holds between two entities, for example: $\textit{created}(\texttt{Jimi\_Hendrix}, \texttt{Little\_Wing})$. This also implies that the inverse relation holds: $\textit{created}^{-}(\texttt{Little\_Wing}, \texttt{Jimi\_Hendrix})$. The last constraint asserts that an entity has a particular value for a given attribute. This can be thought of as a special case of a relation assertion, where the entity relates to a concrete value. For example, the attribute encoding a date of birth: $dateOfBirth(\texttt{Jimi\_Hendrix}, \textit{“1942-11-27”})$.

Definition 3 (Knowledge Base) An *ontology* $\mathcal{O}$ is a set of inclusion dependencies. We use the symbol $\mathcal{A}$ to refer to a set of assertions. Given an ontology $\mathcal{O}$, and a set of assertions $\mathcal{A}$, a *knowledge base* $\mathcal{K}$ is defined as a two-tuple:

$$\mathcal{K} := \langle \mathcal{O},\ \mathcal{A} \rangle.$$

A knowledge base $\mathcal{K}$ *entails* that $a \in (\triangle \ \uplus \ \mathbb{S})$ is an instance of concept C, written $\mathcal{K} \models C(a)$, if $a \in (C)^{\mathcal{I}}$ for all KR-interpretations $\mathcal{I}$ that satisfy all constraints in $\mathcal{K}$. A knowledge base $\mathcal{K}$ *entails* that a concept C_2 *subsumes* a concept C_1, written $\mathcal{K} \models C_1 \sqsubseteq C_2$, if $(C_1)^{\mathcal{I}} \subseteq (C_2)^{\mathcal{I}}$ for all KR-interpretations $\mathcal{I}$ that satisfy all constraints in $\mathcal{K}$.

A knowledge base $\mathcal{K} := \langle \mathcal{O},\ \mathcal{A} \rangle$ is a *knowledge graph* when all of the inclusion dependencies in $\mathcal{O}$ are of the form $A_1 \sqsubseteq A_2$.

□

An ontology (also known as a *TBox*) defines schema-level constraints over the knowledge base, while the set of assertions (also known as an *ABox*) define concrete information about entities and their relationships to entity types and constants.

A knowledge graph restricts inclusion dependencies to have primitive left and right hand sides. The phrase knowledge graph comes from the fact that the knowledge base can be represented visually as a graph where nodes represent entities, types, and constants; and edges represent relations, attributes, and inclusion dependencies between entity types. While most of the work in this thesis applies to general knowledge bases, our implementation of back-end knowledge base query processing exploits the knowledge graph nature of the test data sets for efficiency. An example knowledge graph is shown in Figure 1.1. We represent entity type assertions using the edge label *type* and inclusion dependencies using the edge label *subClassOf*. Entities are represented as ovals, entity types as shaded boxes, concrete domain values as rectangles, and relations as solid lined arrows.

Throughout this thesis, we adopt the following notational conventions.

- *Entities* are written in a typed font with leading capitals (e.g., `John_Lennon`, `Canada`).
- *Entity types* are written in a typed font in all capitals (e.g., `COUNTRY`, `GUITARIST`).
- *Relations* and *attributes* are written in italic camel caps (e.g., *hasWonPrize*, *dateOfBirth*).
- *Keyword queries* and *concrete values* (or *constants*) are written in quoted italic lower case (e.g., *"guitarists"*, *"toronto restaurants"*, *"1940-10-09"*).

We now present a lemma concerning the constraints that must occur in a knowledge graph in order to determine if a knowledge base entails that an entity has a given type. We use the syntactic abbreviation $\{A_1 \sqsubseteq A_2 \ldots A_n \sqsubseteq A\}$ to denote the set of inclusion dependencies $\{A_1 \sqsubseteq A_2, A_2 \sqsubseteq A_3, \ldots, A_{n-1} \sqsubseteq A_n, A_n \sqsubseteq A\}$.

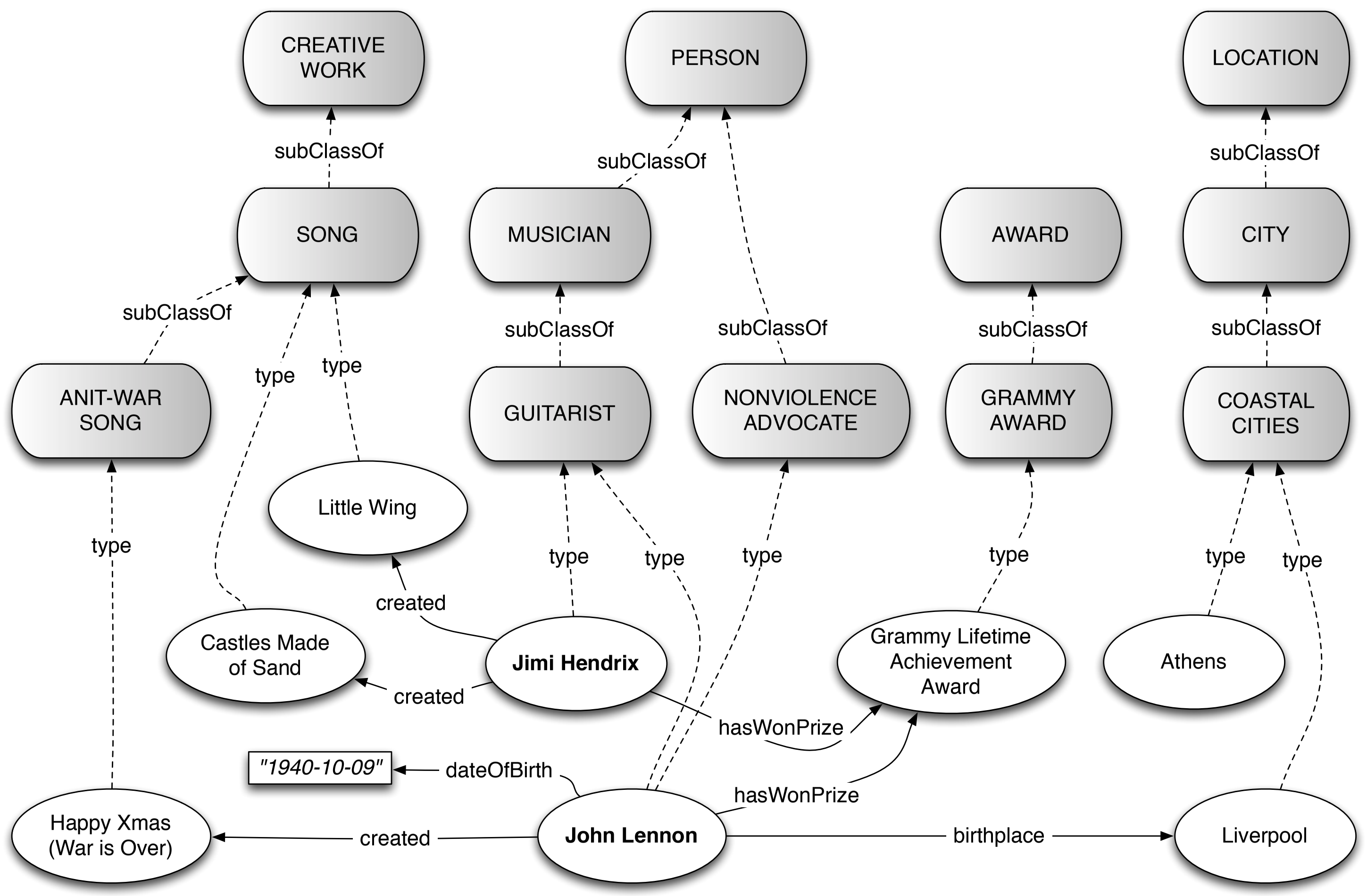


Figure 1.1: An example knowledge graph.

Lemma 1 (Entity Type Inference) Given a knowledge graph $\mathcal{K} = \langle \mathcal{O}, \mathcal{A} \rangle$, an entity type A, and an entity e, $\mathcal{K} \models A(e)$ if and only if either of the following sets of constraints exist in the knowledge graph $\mathcal{K}$:

$$\begin{aligned} &1.\ \{A(e)\} \text{ or,} \\ &2.\ \{A_1(e)\} \cup \{A_1 \sqsubseteq A_2 \ldots A_n \sqsubseteq A\}. \end{aligned} \tag{1.1}$$

Proof: See Appendix B.1.

□

1.2.2 Sources of Web Knowledge Bases

Knowledge bases on the Web are created both manually and in automated ways. Interestingly, both methods tend to produce very large and heterogeneous data sets. For example, DBpedia is automatically extracted from various structured parts of Wikipedia, and produces a KB of over 500 million assertions for ~3.64 million entities (as of DBpedia 3.7). Freebase on the other hand is built from user contributions (in the same style as Wikipedia), and it also yields a large heterogeneous KB consisting of hundreds of millions of assertions ~20 million entities.

Automatically extracted knowledge bases come from various sources. Some are built by directly extracting facts from natural language text with predefined target schemas (e.g., [23, 53]). Others extract arbitrary facts from text without a fixed target schema by a method known as *open information extraction* [5, 17, 19, 37]. Knowledge bases extracted from text can be very large due to the vast amounts of text available for extraction, however achieving high extraction accuracy can be difficult. Textual extraction frameworks are also left with a very difficult *resolution* problem – the problem of determining when two different strings reference the same entity, entity type, attribute, or relation. For example, such knowledge bases may contain redundancies encoding "Einstein's birthday", "Albert Einstein's date of birth", and "Dr. Einstein's birth date."

To exploit existing structure on the Web, some projects have aimed to extract high precision knowledge bases from semi-structured sources. Two prominent examples are DBpedia [11] and YAGO [96]. The DBpedia knowledge base uses custom scripts to extract structured facts from Wikipedia infoboxes and categories. Measures are taken to resolve common rewording of relations to represent facts of the same type using a common schema (e.g., "dateOfBirth" to represent the many ways of expressing the birthday relation). This produces a very large and high quality collection of facts over real world entities, with categorical types for entities. The extraction of the YAGO knowledge base uses a similar methodology as DBpedia, but exploits Wikipedia categories to a greater extent. In YAGO, Wikipedia categories are cross-referenced with WordNet concepts to produce a rich taxonomy over entities. For example, using the YAGO knowledge base one can deduce that `Jimi_Hendrix` is a `Musician`, because the entity `Jimi_Hendrix` is known to be a `Lead_Guitarist`, which is known to be a type of `Guitarist`, which in turn is a type of `Musician`. Google's WebTables project [18] is another example of a large KB extracted from partially structured Web data. HTML tables on Web pages, which can often encode structured data, are processed to build a collection of structured facts with schema information extracted from table headers.

Manually constructing large scale knowledge bases is another possible method for creating Web KBs. While such a goal may seem infeasibly ambitious, several projects have produced large scale high quality knowledge bases from manual labour. In particular, the Cyc project [63] pioneered work in the manual construction of a general domain knowledge base. However the limited number of contributors resulted in a rather small output. The Freebase project [113] on the other hand took a community approach, similar in nature to how Wikipedia was built. Freebase now consists of a large collection of general knowledge facts over 20 million real world entities. Another interesting manually constructed knowledge base is WordNet [76]. WordNet is a linguistic ontology, encoding linguistic relationships between words and grouping words into conceptual units called "synsets." While WordNet is a valuable resource, it should be noted that the type of information it contains is substantially different than the Web knowledge bases we consider in this thesis, which primarily encode factual assertions about real world entities.

There are also a number of smaller scale knowledge bases manually constructed for particular vertical domains. These include, for example, SNOMED CT [125] for medical informatics, the GENE Ontology [114] for genetics, the MusicBrainz ontology [119] for musical works and artists, and many others.

With all of the individual efforts to construct knowledge bases, the question of how an information system can use multiple KBs becomes an issue. The LinkedData [116] effort aims to address this issue by *linking* (defining equivalences over) common resources described in different data sets. The LinkedData Web consists of over 30 billion assertions as of September 2011[7], and has been growing so quickly statistics about its current size are difficult to estimate. Additionally, there are efforts to manually create small ontologies to serve as high level classification or typing systems from which specific knowledge bases can provide finer granularity types as well as data level instances of types. By using common

[7]http://lod-cloud.net/state/. Retrieved May 25, 2013.

high level ontologies, integrating multiple knowledge bases into an information system is easier. Such efforts include UMBEL [127] as well as `schema.org` [121], an effort jointly put forward by industry search leaders Yahoo, Microsoft, and Google.

Independent of how a Web KB is created, they generally share a few common characteristics. They are very large in terms of the number of assertions they contain, and they are heterogenous in the types of information they encode. This makes search using a KB a challenging task, as one cannot assume the user will have familiarity with the contents of an underlying knowledge base.

We have found the formalism presented in Section 1.2 is sufficient to capture many of these real world Web knowledge sources, such as the knowledge graphs DBpedia [11], Freebase [113], and YAGO [96]; as well as the medical knowledge base SNOMED CT [125].

1.2.3 Representing Linguistic Knowledge

Web knowledge bases encode information about real-world entities. This information ultimately makes use of English labels to represent the entities, types, relations, attributes and attribute values. Linguistic knowledge not only connects the knowledge base items to their textual representations, but also encodes information about the linguistic relationships between words in a vocabulary (e.g., synonymy between words, or one word being the lemmatized form of another). Linguistic knowledge can also capture the roles words play in expressing information (i.e., their parts-of-speech). For example, *"city"* is the lemmatized form of *"cities"*, is a synonym of *"metropolis"*, and is a noun.

We encode linguistic knowledge by extending the formalism used to encode world knowledge. We link constructs in a knowledge base to their textual labels by stating an assertion over a *label* relation. For example,

$$\textit{label}(\texttt{GUITARIST},\ \textit{“guitarist”})$$

encodes the label for the type `GUITARIST`. Labels with multiple terms (e.g., *"grammy award"*) can have a mapping to each term in the vocabulary (note that we need to retain the order of the terms, we do this by using a *term* relation that maintains a count *term1*, *term2*, etc. . .). We can then encode various linguistic relationships between the term *"guitarist"* and related terms by extending the knowledge base constraints defined in Definition 2 to allow relations between constants,

$$R_L(k_1, k_2),$$

where R_L is a linguistic relation and k_1 and k_2 are constants. An example of such assertions are given below.

$$\begin{gathered}
\textit{synonym}(\textit{``guitarist''},\ \textit{``guitar player''}) \\
\textit{lemma}(\textit{``guitarists''},\ \textit{``guitarist''}) \\
\textit{term1}(\textit{``guitar player''},\ \textit{``guitar''}) \\
\textit{term2}(\textit{``guitar player''},\ \textit{``player''})
\end{gathered}$$

We can additionally encode part-of-speech relations by allowing assertions from constants to a fixed set of part-of-speech tags.

$$\textit{part-of-speech}(k,\ \texttt{pos-tag})$$

where k is a constant and `pos-tag` is a part-of-speech tag from a reserved set of symbols (disjoint from the abstract domain $\triangle$). For example:

$$\begin{gathered}
\textit{part-of-speech}(\textit{``guitarist''},\ \texttt{noun}) \\
\textit{part-of-speech}(\textit{``guitar''},\ \texttt{noun}) \\
\textit{part-of-speech}(\textit{``hendrix''},\ \texttt{proper\_noun}).
\end{gathered}$$

This model provides us with a formal way of describing how query terms map to knowledge base items. It is ultimately the text representation of knowledge base items that we will use to map keyword query terms into the knowledge base. Figure 1.2 shows a sample of our example knowledge graph with the associated connections to an example linguistic knowledge base. We abbreviate "part-of-speech" as "pos" in the example.

1.2.4 Sources of Linguistic Knowledge Bases

Linguistic knowledge bases can be created manually or automatically extracted from text collections or query logs. WordNet is a manually constructed and expert curated collection of linguistic knowledge [76]. It encodes a high precision collection of English linguistic knowledge including a large English vocabulary, *synsets* which are relations among groups of semantically synonymous terms, and part-of-speech annotations of terms. Vocabularies can also be built automatically by analyzing text collections of interest, such as document collections or query logs, and enumerating a set of terms. Synonyms can be discovered automatically as well using statistical methods (see for example, [28]). Also, Web page hypertext links can also be used to extract synonyms for entities on the Web [77]. For example, if the text *"The Great One"* is often used to link to the `Wayne_Gretzky` Wikipedia page (and not often used to link to other pages), then it is likely that this label is a synonym for the entity `Wayne_Gretzky`.

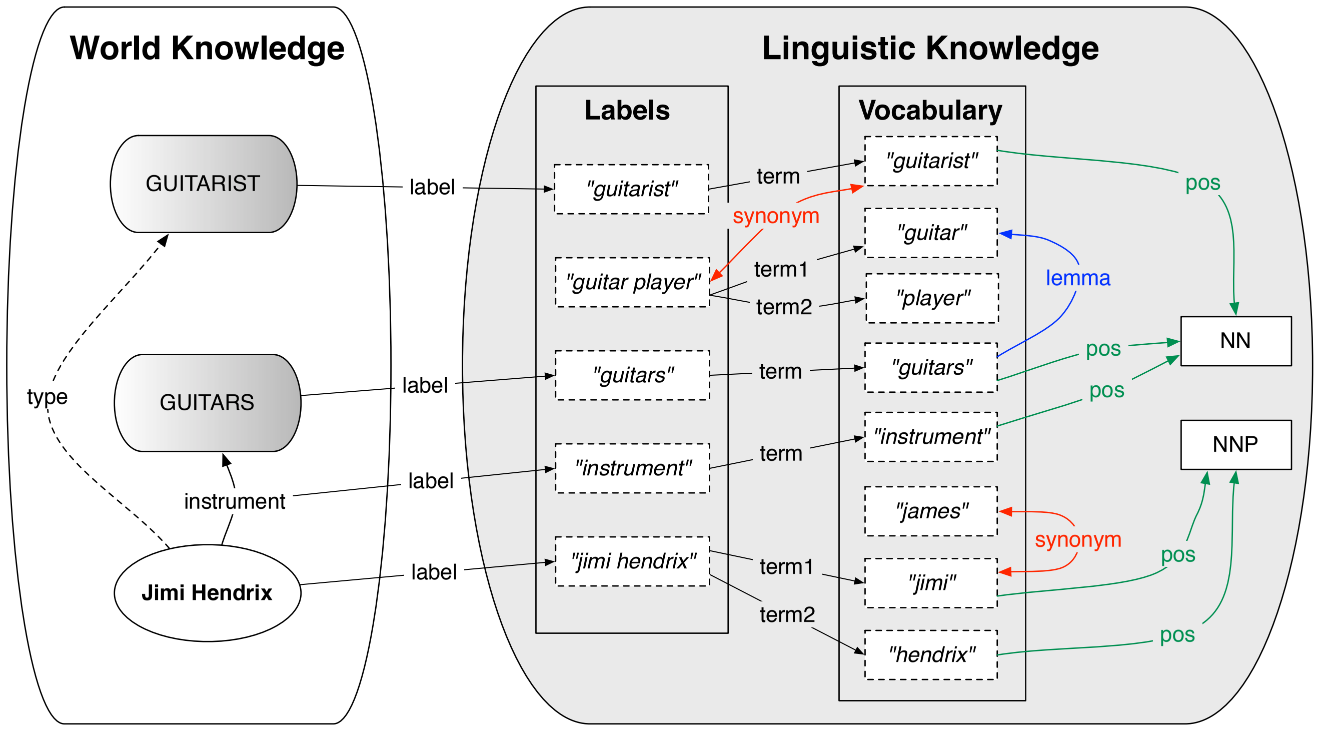


Figure 1.2: Part of an example knowledge graph extended with linguistic knowledge. The relation "pos" represents part-of-speech, NN denotes noun and NNP denotes proper noun. The "label" relation connects knowledge graph items to their syntactic representations.

1.2.5 A Knowledge Graph with Textual and Linguistic Annotations

We compile a number of publicly available resources to construct a knowledge graph containing tens of millions of facts over millions of real-world entities. Our knowledge graph is centred around the fact collection and type hierarchy compiled by the YAGO [96] project. We make use of the entity labels extracted from the anchor text of cross-page links in Wikipedia that are included in YAGO. We then integrate WordNet relations to model synonymy among terms in the labels of the graph nodes and edges. We use a mapping of terms to part-of-speech tags obtained from the well known Brown and Wall Street Journal corpora (which have been manually annotated) by taking the most frequent tag for each term as its part-of-speech. We compute the lemmatized form relations on-the-fly during query processing using the Stanford natural language processing toolkit [126]. We also cross-reference YAGO entities with DBpedia [11] (using existing Linked Data mappings) in order to integrate integrate Wikipedia abstracts for each entity as a textual description the entity.

Overall, the knowledge graph is built entirely from publicly available sources and consists of ~3.3 million entities, ~300,000 entity types, ~11 million constants (strings, numbers, dates, etc...), and ~38 million assertions (edges) over these KB items.

Experiments conducted as part of this thesis make use of various subsets of this knowledge graph depending on the nature of the experiment. In particular, some experiments make use of only the core YAGO data set with linguistic annotations (Chapters 3 and 4). Other experiments make use of an *entity database* subset, consisting of the entity sets of YAGO and DBpedia, along with the extracted entity labels, synonyms, abstracts, and linguistic annotations (Chapter 5).

1.3 Representing Query Intentions

We will ultimately use concepts in the given KB formalism to represent the explicit intention or information need of an ambiguously specified user keyword query. We adopt the following definition of a concept-based search query over a knowledge base, analogous to instance queries over description logic knowledge bases.

Definition 4 (Concept Search Query) A *concept search query* is given by a concept C expressed using the grammar in Definition 1. Given a knowledge base $\mathcal{K}$ and a concept search query C, the answer to the query C over $\mathcal{K}$ is the set of entities or constants inferred to be instances of C by $\mathcal{K}$,

$$answer(C, \mathcal{K}) = \{a \mid \mathcal{K} \models C(a)\}.$$

$\square$

As an example, the concept

$$\texttt{MUSICIAN} \sqcap \exists hasWonPrize(\texttt{GRAMMY\_AWARD})$$

is interpreted as a query for all entities that are of type MUSICIAN and that have a *hasWonPrize* relationship to some entity that has type GRAMMY_AWARD. Inferring which entities are answers to a concept search query may require inference over the knowledge base, as can be seen by considering the example query over the knowledge base illustrated in Figure 1.1. A query processor must exploit the knowledge that a GUITARIST is a type of MUSICIAN to infer that the entities Jimi_Hendrix and John_Lennon qualify.

This class of queries has a natural mapping to conjunctive queries (e.g., SPARQL queries [87]). Concept search queries correspond to conjunctive queries with a single head variable (hence the “search” or “retrieval” nature of the queries), with query bodies that are directed acyclic graphs. (Note however, that the reverse is not necessarily true, i.e., not *any* directed acyclic graph query can be expressed as a concept search query.) The example query above could be written in SPARQL the following way (assuming knowledge base inference has been accounted for).

```
SELECT  ?x
WHERE  {
      ?x  type  GUITARIST .
      ?x  hasWonPrize  ?y .
      ?y  type  GRAMMY_AWARD .
   }
```

Concept search queries essentially define selection conditions over knowledge base entities. Such functionality could also serve as a component of a larger knowledge base algebra, as explored in [85].

1.4 Thesis Overview

1.4.1 Contributions

This thesis makes the following contributions.

- At the core of this thesis is a novel method for interpreting keyword queries over Web knowledge bases that computes mappings of keyword queries to formal concept search queries over the vocabulary of a given knowledge base. The mappings form probable interpretations of the user's information need based on statistical models learned from real user query logs (Chapter 3). The result of this process is a set of the top-k structured concept search queries representing possible interpretations of a keyword query, which can be evaluated over the knowledge base to produce precise answers to the keyword query (Chapter 4). The latter process is able to exploit semantics encoded in the knowledge base by performing reasoning during query evaluation. We formalize the problem of mapping keyword queries to concept search queries over a knowledge base in Chapter 2.

- We present the results of an analysis of a real Web search log, giving insight into the types of entity-based queries asked by real users. Our analysis shows the importance of addressing entity-based queries due to the large number of these queries issued by users. The analysis also establishes relationships between linguistic structure and semantic structure, and gives insight into the types of latent semantic structures that repeatedly occur in keyword queries (Chapter 3).

- We propose a keyword-based structured query formalism that allows users to create expressive descriptions of their information need without having detailed knowledge of an underlying knowledge base vocabulary. The query formalism also acts as an intermediate language for representing possible structure in users' keyword queries. We propose a model for efficient and effective mapping of these structured keyword expressions into a large knowledge base (Chapter 4).

- We propose a model for re-ranking entity results produce by a baseline ranking algorithm to increase the effectiveness of results based on learned user profiles. We show that the re-ranking is effective for a number of popular baseline entity ranking methods (Chapter 5).

- We propose a methodology for ad-hoc entity retrieval evaluation. We design a framework for evaluating entity retrieval systems that builds upon existing evaluation approaches in document retrieval and maximizes the reusability of expensive human relevance judgments on query results. We deploy our proposed evaluation methodology on a real world data set and query workload. We show experimentally that our proposal is stable for some popular evaluation metrics, and that it can reliably distinguish among the effectiveness of systems with these metrics (Chapter 6).

1.4.2 Outline

The remainder of this thesis is organized as follows. Chapter 2 formalizes the keyword query understanding problem and gives an overview of our approach to semantic query understanding. The details of our approach to structuring keyword queries are presented in Chapter 3. Chapter 4 explores the problem of mapping structured graph queries with keyword predicates into a knowledge base. Such queries form an intermediate representation of keyword query interpretations in our system. Keyword queries interpreted and evaluated over Web knowledge bases ultimately produce a list of entity results. Chapter 5 address the entity ranking problem, with a focus on learning personalized user interest models from a query log, and ranking entity results based on a user's interests. Throughout the thesis, we conduct a number of evaluations involving entity retrieval. The theory and validation of the methodology used to evaluate entity retrieval is presented in Chapter 6. We conclude in Chapter 7 and discuss directions for future work.

Chapter 2

The Query Understanding Problem

As the amount of structured data on the Web continues to grow, the ability to exploit this data for keyword search becomes increasingly important. Efforts such as DBpedia [11], Freebase [113], and Linked Data [116] have produced large heterogeneous knowledge bases that encode great amounts of information about many real world entities. Web search queries seeking information about these entities could be better served by interpreting the query over a knowledge base in order to provide an exact answer to the query, or to enhance document retrieval by understanding the entities described by the query.

For example, consider a user searching for *"songs by jimi hendrix."* While a text search may retrieve relevant documents to the query terms, it leaves the user with the task of scouring textual results in search of the information they are seeking. This user's query could be better served by returning a list of particular songs by the musician Jimi Hendrix (precise answers to the query), possibly along with information about each song (e.g., structured facts from a knowledge base or documents resulting from a text search for the particular song). Similarly, the keyword query *"author of moby dick"* could be better answered if a search system understood that "moby dick" describes a particular book, "author of" describes a relationship, and the *query intent* is to find the unspecified entity that has an "author of" relationship to "moby dick."

We refer to the process of interpreting keyword queries over a knowledge base as *semantic query understanding*. This problem has the following characteristics that pose difficult technical challenges.

- **Ambiguity** Keyword queries tend to be short, ambiguous, and underspecified. For a given keyword query there may be multiple possible ways to interpret the underlying query intent. Semantic query understanding systems need to accurately interpret entity-based keyword queries when the underlying reference knowledge base contains the relevant information.

- **Representation & Coverage** Not all keyword queries have an entity focus (so called, *entity-based keyword queries*). A query such as *"corporate tax laws buyout"* may intentionally be seeking documents that mention the given query terms, and an attempt to represent the query in terms of knowledge base entities may produce an incorrect interpretation and harm the quality of results. Similarly, some entity-based queries may seek information that does not exist in a given reference knowledge base. A high accuracy semantic query understanding system will answer queries only when the query intent can be represented and evaluated over the reference knowledge base, and recognize when queries cannot be answered due to representation or KB coverage.

- **Scale** Web knowledge bases tend to be very large and heterogenous, meaning approaches cannot be engineered to exploit domain specific regularities or depend on small fixed schemas with complete data. Semantic query understanding techniques must scale to large heterogeneous Web knowledge bases.

Query understanding is a broad phrase used to describe many techniques applied to keyword queries in order to represent the underlying information need in a way that a search system can exploit. Some popular approaches include topic classification, in which the topic of a query is determined using a statistical classifier. Topic classification aids a search system by giving it context, or by allowing the system to select the appropriate data source to search. For example, the query

$$\text{\textit{``songs by jimi hendrix''}} \tag{2.1}$$

may be classified as a music query. However query classification does not give any insight into precisely what the query is looking for within the domain of music.

Another query understanding approach that has received recent attention from researchers is term annotation. Term annotation labels individual query terms with annotations that describe the role these terms play with respect to an underlying information system. Following our example query, the terms *"jimi hendrix"* may be annotated as a *name*, and the term *"songs"* annotated as an entity *type*. While query annotations can aid a search system in understanding the meaning of individual terms, the annotations do not describe the underlying structure of the query as a single coherent query interpretation. Continuing the example, we want to model the latent query structure that expresses the query intention as finding entities of type *song* that are created by an entity named *"jimi hendrix"* (as opposed to finding, for example, information about the person named *"jimi hendrix"* who is known to have written songs, or a song named *"jimi hendrix"*).

The goal of semantic query understanding is to compute a formal interpretation of an ambiguous information need. The interpretation is represented using some underlying structured query language. The information need is expressed as a keyword query, a short and often underspecified string of natural language words. The query understanding process is inherently uncertain, and approaches for semantic query understanding are generally based on inexact matching and ranking using probabilistic models. Figure 2.1 shows our proposed semantic query understanding process for a variation of our running example information need.

A formal modelling of semantics can aid in returning more relevant results, and can also allow the search system to decide on the type of results and the format for returning an answer. For example, a query for digital cameras can return structured shopping results, rather than (or in addition to) text documents. A query for a particular fact such as a celebrity's birthday can be better served by returning the exact date value, rather than a list of documents. Thus the more a search system understands about a query's intent, the more possibilities there are for addressing the user's information need.

Our example keyword query can be structurally understood as finding all entities of a type described as *"songs,"* that have a *"by"* relationship to an entity described as *"jimi hendrix"*. This is given as the following conjunctive query.

$$q(x)\text{:-}\exists y.\texttt{SONG}(x) \wedge \mathit{createdBy}(x, y) \wedge y = \texttt{Jimi\_Hendrix} \tag{2.2}$$

Where the predicates `SONG` and *createdBy*, and the entity `Jimi_Hendrix` are part of a Web knowledge base. The difficulty in mapping a keyword query to a formal interpretation lies in the inherent ambiguity in keyword queries, and the many possible mappings query terms can have over very large reference data collections. For example, the term *"hendrix"* matches 68 data items in our data set (see Section 1.2.5), the term *"songs"* matches 4,219 items, and the term *"by"* matches 7,179 items. This yields a space of over two billion possible conjunctive queries constructed by mapping each query term to a syntactically similar predicate. This does not count the possibility of additional predicates existing in the underlying conjunctive query that are not explicitly represented by a term in the keyword query. For example, the query *"hendrix songs"* shown in Figure 2.1 shares the same logical structure as example query 2.2, even though the *createdBy* relation is not explicitly represented by any query terms. It is the job of the query understanding system to infer the existence of such latent relations.

Another key challenge of the semantic query understanding problem is to determine when a query *cannot* be modelled using a given collection of reference knowledge. Indeed, no matter how large a data set is, it will always be possible to formulate a query that describes information outside of the scope of the reference knowledge. Identifying when a query's semantics cannot be modelled using the available knowledge is equally important to the problem of correctly modelling the query interpretation when the reference knowledge captures the required information. We consider the importance of this component of a query understanding system in the design of our approach and emphasize it in our evaluation.

In this thesis we propose a method for interpreting keyword queries over Web knowledge bases. We use an annotated Web search query log as training data to learn a mapping between keyword queries and their underlying semantic structures. Because of the very high cost in creating training data, we design an approach that maps high level representations of keyword queries to schema-level representations of structured queries, greatly reducing the amount of training data needed to accurately learn these mappings. Our approach integrates state-of-the-art techniques in natural language processing with top-k search over structured data and knowledge base query processing, bridging the gap between statistical representations of keyword queries and database formalisms for structured representations of information needs.

2.1 Formalizing The Query Understanding Problem

A surface keyword query is given by a user as a string. We define a keyword query as the ordered set of individual terms from a concrete domain $\mathbb{S}$ occurring in the input string. The only structure specified in a keyword query is the order of the terms.

Definition 5 (**Keyword Query**) A *keyword query* Q is a non-empty ordered set of n strings

$$Q = \langle q_1 q_2 \dots q_n \rangle$$

such that $q_i \in \mathbb{S}$ for all $1 \leq i \leq n$. □

We can think of a concept search query C as being composed of a number of items from the vocabulary of a knowledge base, where each knowledge base item is either an entity, entity type, relation, attribute, or constant. In order to express query interpretations as concepts, we define a notion of admissibility of a concept search query for a keyword query in terms of the knowledge base items used and the coverage of keywords the concept provides.

Definition 6 (**Admissible Keyword Query Interpretation**) Given a keyword query Q over $\mathbb{S}$, a knowledge base $\mathcal{K}$, and a concept search query C, let $item(\mathcal{K})$ denote the set of knowledge base items occurring in a knowledge base $\mathcal{K}$, let $item(C)$ denote the set of knowledge base items occurring in a concept search query C, and let $\mathcal{L}(i) \subseteq \mathbb{S}$ denote the set of textual labels in $\mathbb{S}$ that can be used to describe a knowledge base item i. A concept search query C is an *admissible keyword query interpretation* for keyword query Q iff

$$\forall i \in item(C), i \in item(\mathcal{K}); \tag{2.3}$$

$$\forall q \in Q, \exists i \in item(C), q \in \mathcal{L}(i); \text{ and} \tag{2.4}$$

$$|answer(C, \mathcal{K})| > 0. \tag{2.5}$$

□

Equation 2.3 constrains all predicates to be part of the given knowledge base, ensuring the query is safe. Equation 2.4 ensures coverage of all keyword query terms. Equation 2.5 ensures that the query is consistent with the knowledge base and that the knowledge base has sufficient data coverage to answer the query by only allowing concept search queries with non-empty answer sets (refer to Section 1.3 for a definition of concept search query semantics). In theory we expect that $\mathcal{L}(i)$ contains all possible representations of i, including all possible synonyms and variations. In practice, we approximate this by relaxing the constraint $q \in \mathcal{L}(i)$ to allow fuzzy and partial string matching. We also consider segmenting keyword queries into sequences of keyword phrases, and allow partial or fuzzy matching at the phrase level in Chapter 3 (e.g., match *"President Obama"* to *"Barack Obama"*).

Definition 7 (**Keyword Query Understanding Problem**) Given a keyword query Q and a knowledge base $\mathcal{K}$, the *query understanding problem* is to find the most probable concept search query C given the keyword query Q:

$$C = \text{argmax}_{C'} \ \Pr(C'|Q) \tag{2.6}$$

such that C is an admissible interpretation of the intention of Q with respect to a knowledge base $\mathcal{K}$. □

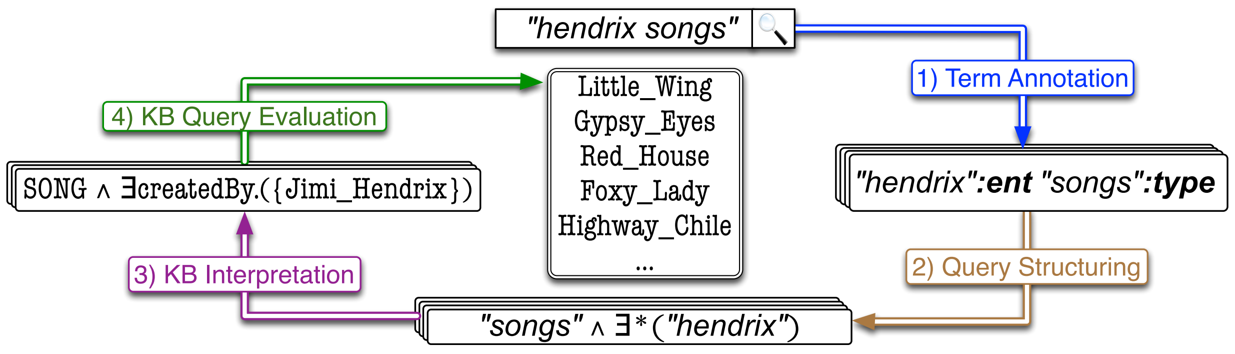


Figure 2.1: Overview of the query understanding process.

2.2 Solution Overview

Estimating the distribution in Equation 2.6 directly would require a large amount of labelled training examples. One would need to see each possible keyword in a vocabulary enough times to model the different possible semantic mappings that word can take in various contexts. The space of possible keyword queries mapping to possible concept search queries is very large, and estimating such a mapping directly is not feasible when training data is limited. Because of the manual effort required in creating labelled training data, we need to design an approach that can maximize the utility of a small collection of training examples. We propose to compute high level summaries of queries based on annotating query terms with semantic annotations, and learn a conditional distribution over structured query templates given these summaries.

Our semantic query understanding approach is summarized as a sequence of four steps, as illustrated in Figure 2.1.

1. **Keyword Query Annotation** Queries are first annotated with the semantic constructs from the knowledge representation language defined in Definition 1 (i.e., entity, type, relation, attribute, attribute value). We use part-of-speech tags as features that suggest probable semantic constructs for each query term. The mapping from part-of-speech tags to semantic constructs is learned from an annotated query log (Chapter 3).

2. **Keyword Query Structuring** Annotated queries are structured by computing the most probable structured query templates given the annotations as a semantic

summary of the query contents. This relationship between annotations and query structures is learned from an annotated query log. Learning a mapping directly from keywords to structured queries would require large amounts of training examples. By learning the mapping from semantic summaries to query templates, we take advantage of the redundancy in the training data caused by many queries sharing the same summaries and templates. For example, queries containing two entities tend to be structured in particular ways, while queries containing a type followed by an entity may exhibit different structures with regularity (Chapter 3).

3. **Knowledge Base Mapping** Semantically annotated keyword queries can be combined with a structured query template to form a structured representation of the keyword query known as a *structured keyword query.* This structured keyword query is essentially a structured graph query (in the form of a concept search query) with keyword predicates. We map structured keyword queries into concept search queries over the vocabulary of the knowledge base. Our method computes mappings that maximize syntactic similarity of query terms to KB items as well as semantic coherence of the concept search query given the knowledge base (Chapter 4).

4. **Knowledge Base Query Evaluation** Concept search queries are then executed over the knowledge base to find entities and values described by the query. This process performs query-time inference, exploiting the semantics encoded in the knowledge base to compute query answers using a custom knowledge base engine. Our knowledge management approach also does closure path caching, taking advantage of sets of queries that share query terms generated from the same keyword query by the query understanding process (Chapter 4).

We also address the entity ranking problem in Chapter 5 from the view of personalized entity ranking. In Chapter 6 we provide an in-depth analysis of evaluation of the entity retrieval task.

Figure 2.2 gives a detailed visualization of the query understanding and evaluation process for our example query *"jimi hendrix songs"*. The figure also illustrates the steps in which query log and knowledge base data are used.

Note that at steps 1 and 2 of the process, we are not yet concerned with how query terms map to particular knowledge base items as would be done in traditional keyword search over graphs. Inferring probable structures of the query terms first will allow us to constrain the possible mappings into the knowledge base. This can improve performance by fixing the structure of possible mappings into the knowledge base, and improve effectiveness by only allowing structures that have a high probability of being representative of query

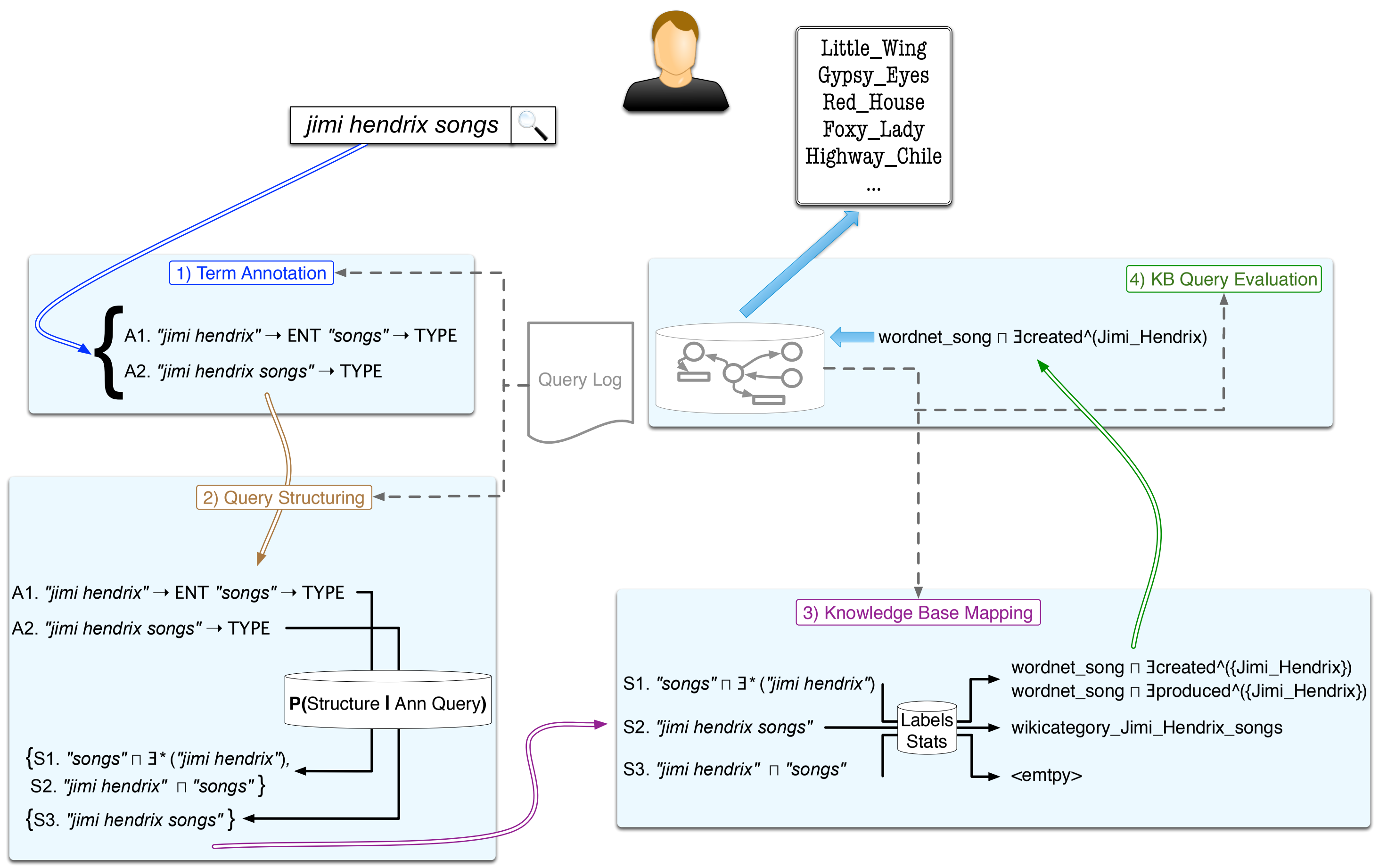

jimi hendrix songs
Little_Wing
Gypsy_Eyes
Red_House
Foxy_Lady
Highway_Chile
...
1) Term Annotation
A1. "jimi hendrix" → ENT "songs" → TYPE
A2. "jimi hendrix songs" → TYPE
Query Log
4) KB Query Evaluation
wordnet_song ⊓ ∃created^(Jimi_Hendrix)
2) Query Structuring
A1. "jimi hendrix" → ENT "songs" → TYPE
A2. "jimi hendrix songs" → TYPE
P(Structure | Ann Query)
{S1. "songs" ⊓ ∃* ("jimi hendrix"),
S2. "jimi hendrix" ⊓ "songs"}
{S3. "jimi hendrix songs"}
3) Knowledge Base Mapping
S1. "songs" ⊓ ∃* ("jimi hendrix")
S2. "jimi hendrix songs"
S3. "jimi hendrix" ⊓ "songs"
Labels
Stats
wordnet_song ⊓ ∃created^({Jimi_Hendrix})
wordnet_song ⊓ ∃produced^({Jimi_Hendrix})
wikicategory_Jimi_Hendrix_songs
<emtpy>

Figure 2.2: Detailed overview of the query understanding process.

"jimi hendrix songs"	User keyword query
$\hookrightarrow$ *"jimi hendrix"*:ent *"songs"*:type	1) Term annotation
$\hookrightarrow$ *"songs"* $\sqcap$ $*$ (*"jimi hendrix"*)	2) Query Structuring
$\hookrightarrow$ `SONG` $\sqcap$ $\exists \mathit{created}^{-}$ (`Jimi_Hendrix`)	3) Knowledge base mapping
$\hookrightarrow$ `{Little_Wing, Castles_Made_of_Sand}`	4) Knowledge base evaluation
"jimi hendrix birthplace"	User keyword query
$\hookrightarrow$ *"jimi hendrix"*:ent *"birthplace"*:relation	1) Term annotation
$\hookrightarrow$ *"birthplace"*(*"jimi hendrix"*)	2) Query Structuring
$\hookrightarrow \emptyset$	3) Knowledge base mapping
$\hookrightarrow \emptyset$	4) Knowledge base evaluation
"john lennon birth city"	User keyword query
$\hookrightarrow$ *"john lennon"*:ent *"birth"*:relation *"city"*:type	1) Term annotation
$\hookrightarrow$ *"city"* $\sqcap$ *"birth"* (*"john lennon"*)	2) Query Structuring
$\hookrightarrow$ `CITY` $\sqcap$ $\exists \mathit{birthplace}^{-}$ (`John_Lennon`)	3) Knowledge base mapping
$\hookrightarrow$ `{Liverpool}`	4) Knowledge base evaluation

Figure 2.3: Examples of the query understanding process.

intentions. Inferring query structures also gives us the ability to know precisely what piece of information is being requested, much like a projection in structured query languages.

At each stage of the query understanding process, we generate the top-k most probable outputs as input to the next step. The expectation is that poor annotations will not produce good structurings, which will then not find mappings into the knowledge base. On the other hand, correct annotations (even if not ranked as the most probable), will produce structured keyword queries that do map to coherent knowledge base queries. Similarly, queries searching for information that cannot be modelled by the knowledge base (for example, due to lack of coverage) will not resolve into concept search queries because generated candidates will not have mappings that satisfy both the candidate structure and keyword constraints.

Some examples of keyword queries, their semantically annotated forms, structures, and proper knowledge base interpretations, along with the final results after KB evaluation are illustrated in Figure 2.3.

2.3 Baseline Approaches

In this section we give an overview of the most relevant techniques used for answering keyword queries over knowledge graphs, and contrast it with our proposed approach for interpreting and answering keyword queries. A detailed discussion of specific related work is presented in Sections 3.5 and 4.8 and a detailed experimental comparison in Sections 3.4 and 4.7.

Existing methods that can be applied to the problem of answering keyword queries over knowledge bases are all based on some form of keyword search over graphs. The problem of keyword search over graphs is to find nodes in the knowledge graph who's labels match query terms, then find connections among these *seed* nodes. A connected subgraph of the knowledge graph that contains nodes matching all of the keyword query's terms is considered a result. Often these result graphs are restricted in order to improve performance. Some approaches restrict the result subgraphs to be trees, others also impose maximum size conditions to constrain the search space.

Once a subgraph is found that contains all query terms, it may be directly returned to the user as a result. Other approaches consider generalizing the returned graph, by replacing nodes not containing query terms (i.e., nodes included only to connect other nodes that do match query terms) with variables and then issuing the resulting graph as a query against the knowledge base.

There are a number of drawbacks to using approaches for keyword search over graphs for answering keyword queries over knowledge graphs. The first is that the result of the keyword search over graphs techniques is a subgraph, not a specific answer to the query. While the subgraphs will often contain the desired answer, this leaves the difficult problem of how to communicate a graph answer to an end user in such a way that the user can easily identify the answer to their query. An example is illustrated in Figure 2.4. The query terms *"john lennon birth city"* produce an answer graph consisting of three edge connections represented as three triples as follows.

$$\langle \texttt{John\_Lennon},\ \textit{birthplace},\ \texttt{Liverpool} \rangle$$
$$\langle \texttt{Liverpool},\ \textit{type},\ \texttt{COASTAL\_CITIES} \rangle$$
$$\langle \texttt{COASTAL\_CITY},\ \textit{subClassOf},\ \texttt{CITY} \rangle$$

Encoded in this answer graph is the query answer – Liverpool – and the task of locating the answer is left to the user. The difficulty of such a task will largely depend on how the search system manages to convey a graph as a result. An approach that can directly

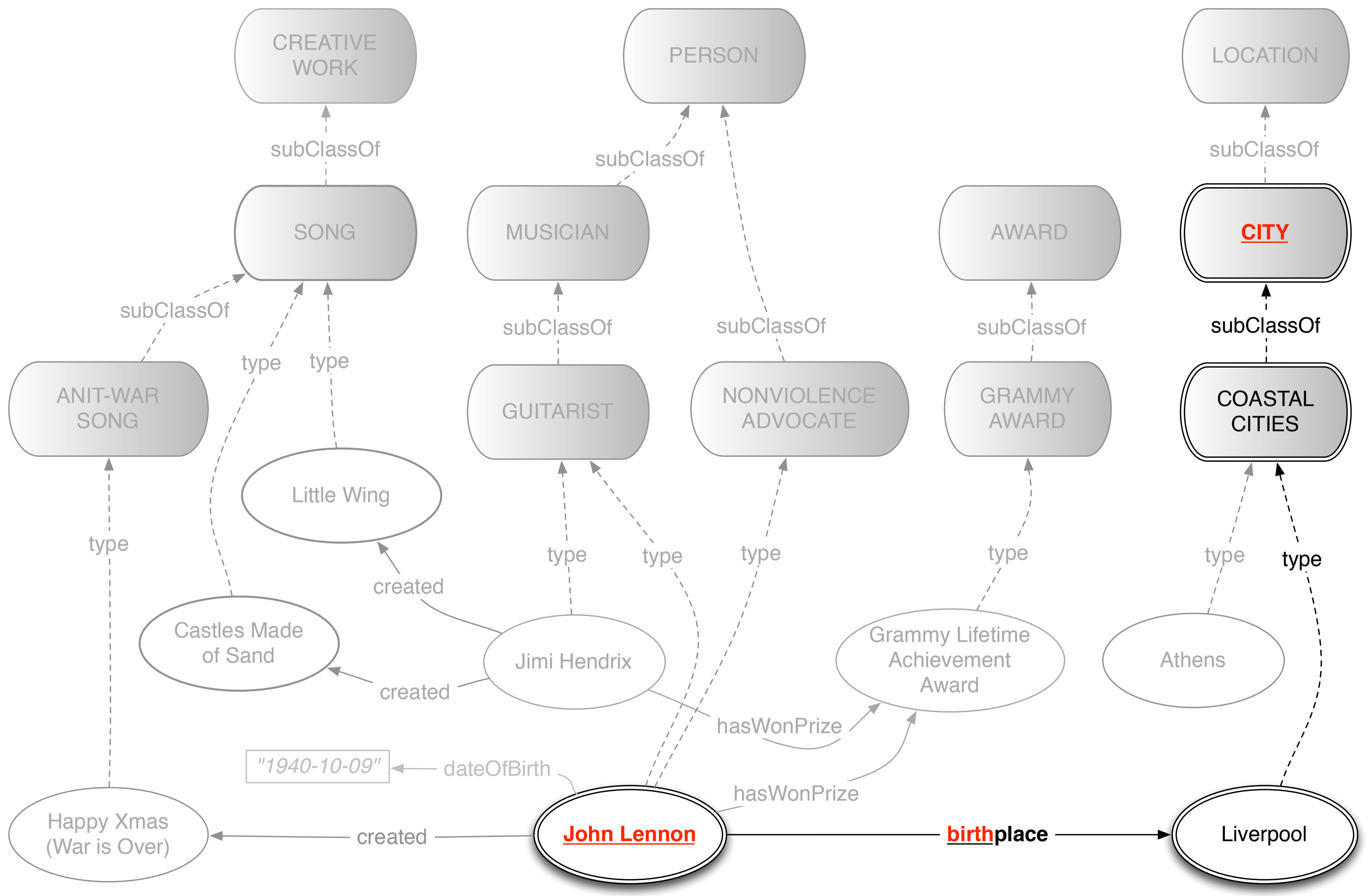


Figure 2.4: Example query result subgraph for the query *"john lennon birth city"*. Nodes included in the result are rendered with double-stroke boarders, nodes matching query terms have underlined text.

provide the answer to the user is likely preferable to an approach that produces potentially large and complex answer graphs.

The second drawback to keyword search over graph techniques is that there are no constraints on the shape of the answer graph, which can lead to meaningless interpretations that coincidentally match query terms. Figure 2.5 illustrates an example of this problem. The query seeks to find the birthplace of Jimi Hendrix. While our example knowledge base contains information about the entity `Jimi_Hendrix`, as well as information encoded with the *birthplace* relation, it does not contain the *birthplace* of `Jimi_Hendrix`. This type of sparsity is very common among automatically extracted knowledge graphs. Despite the query answer not existing in the graph, an answer graph can quite easily be found by linking the entity `Jimi_Hendrix` to the entity `Liverpool` which is the birthplace of John Lennon. The graph is connected via an award both entities have won (or alternatively by both entities having the same type `GUITARIST`).

The third potential drawback is the efficiency of keyword search over graph techniques, particularly in the case when the top-k answers are being queried and k answers do not exist in the graph. The only way to confirm that k answers don not exists is to exhaust the entire search space. There have however, been efforts to greatly improve the performance of search over graphs, such as [98] by use of indexing graph summarizations.

The final issue with keyword search over graphs is the difficult problem of ranking answer graphs. The answer graphs contain very little text making existing ranking methods from the information retrieval literature difficult to apply. Most approaches rely on heuristics based on the similarity of query terms to node labels and the size of the answer graph. Our proposed method for interpreting keyword queries over knowledge bases also requires ranking graph structured query interpretations, however our formalization leads to a natural probabilistic interpretation of candidate scoring. We are able to estimate the probability of keyword query annotations and structuring directly from annotated query logs, and while part of our ranking depends on heuristic scoring (mapping annotated queries into the knowledge base), we are able to empirically validate the effectiveness of these heuristics.

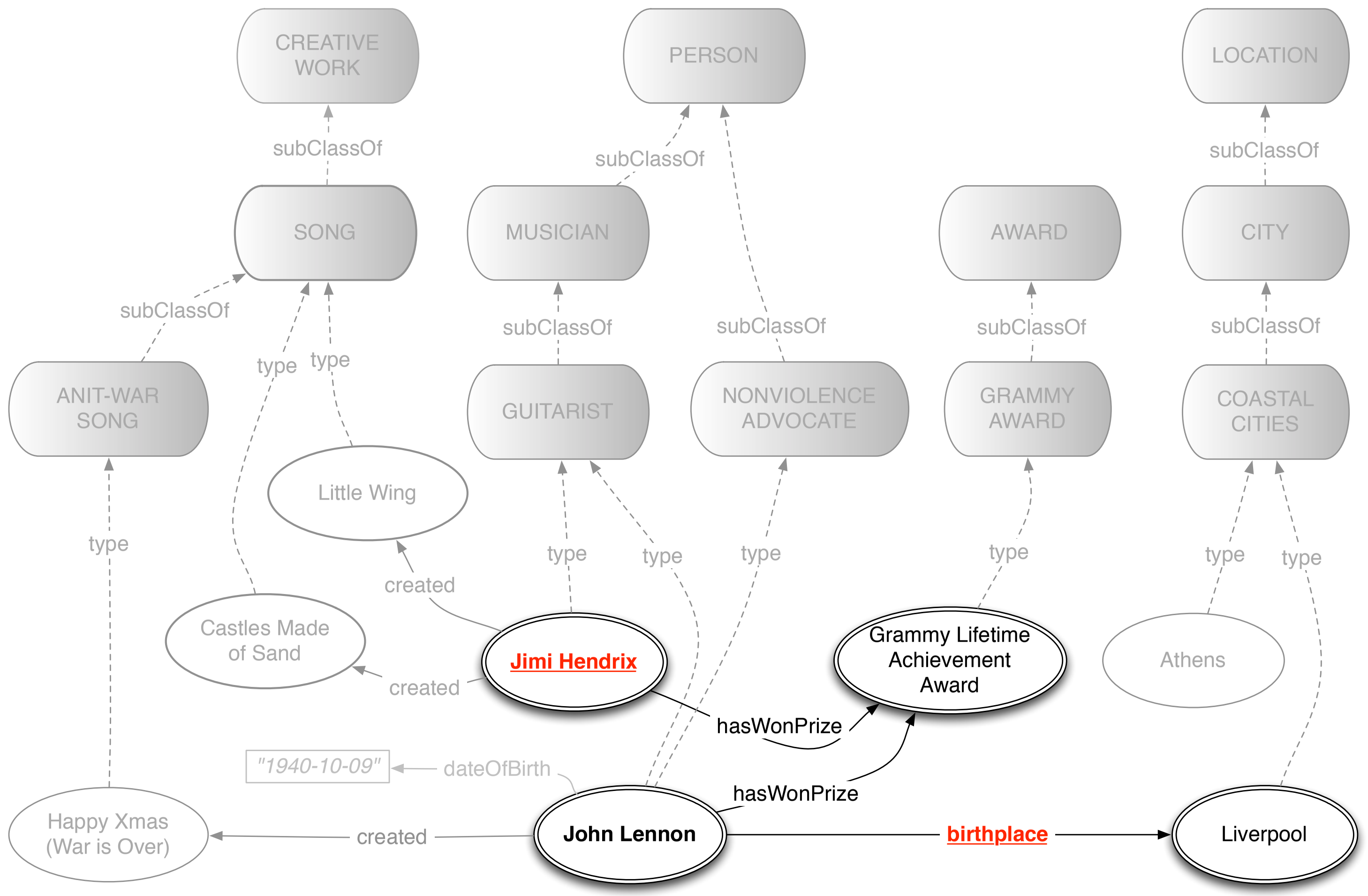


Figure 2.5: Example query result subgraph for the query *"jimi hendrix birthplace"*. Nodes included in the result are rendered with double-stroke boarders, nodes matching query terms have underlined text.

Chapter 3

Semantic and Structural Annotation of Keyword Queries

As a first step to modelling the semantics of a keyword query, we map keyword sequences to structured keyword expressions representing possible intents, called *structured keyword queries*. This corresponds to steps 1 and 2 of our query understanding process as illustrated in Figures 2.1 and 2.2. A structured keyword query describes a query in two ways: it breaks it into *segments* that represent particular semantic constructs, and it describes how these constructs relate to each other. In order to form structured keyword queries from a given keyword query, we will have to address these two problems. This chapter presents work originally published in [80].

3.1 Query Segmentation & Semantic Annotation

Recall that a keyword query Q is a sequence of query terms $Q = \langle q_1 q_2 \dots q_n \rangle$ over a concrete domain $\mathbb{S}$ (see Definition 5). A semantically annotated keyword query assigns *semantic constructs* from a knowledge representation language to subsequences of keyword query terms.

Definition 8 (Semantically Annotated Query) Given a keyword query Q, and the knowledge representation language defined in Definition 1, a *semantically annotated keyword query AQ* (or simply *annotated query* when clear from context) is a sequence of keyword-phrase/semantic-construct pairs

$$AQ = \langle q_1 q_2 \ldots q_i \rangle{:}a_1 \ \langle q_{i+1} \ldots q_j \rangle{:}a_2 \ \ldots \langle q_{j+1} \ldots q_n \rangle{:}a_n$$

where each $q_i \in \mathbb{S}$ and each $a_i \in \{ent, type, rel, attr, val\}$ the set of semantic constructs from the knowledge representation language.

□

A semantically annotated version of our example query is given by the following.

"songs":type *"by"*:rel *"jimi hendrix"*:ent

An algorithm that annotates keyword queries with semantic constructs must solve both the *segmentation problem* (the problem of determining the boundaries that separate multi-term phrases) and the annotation problem. The resulting annotations indicate which sub-sequences of query terms in a keyword query correspond to which semantic constructs. We want to compute the probability of an annotated query given a keyword query, $\Pr(AQ|Q)$.

Research has shown that part-of-speech (POS) tagging can be accurately performed over keyword queries [7]. Our approach to annotating queries exploits query terms, their POS tags, and sequential relationships between terms and tags to concurrently infer a segmentation and semantic annotation of a part-of-speech annotated keyword query. To do this, we use a conditional random field (CRF) [59], a state-of-the-art machine learning method for labelling sequence data. As a baseline method for comparison, we also try a naïve Bayes based technique that directly annotates terms independently with semantic constructs. We then segment the query by joining any adjacent terms sharing the same semantic construct.

3.1.1 Naïve Bayes Annotation

Our baseline term classification aims to exploit the relationship between part-of-speech tags and semantic constructs by using a term's part-of-speech as a proxy for the term. Intuitively, there is a relationship between semantic constructs (e.g., entities, relations) and the parts-of-speech used in describing instances of those constructs. For example, entities are generally expressed using proper nouns like "Jimi Hendrix" or "New York." Relations are often described by prepositions, such as "in" or "by." The mapping between parts-of-speech and semantic constructs however, is not always so clear. Relations can sometimes be described by parts-of-speech other than prepositions (e.g., the noun "birthplace"); nouns can often describe many different semantic constructs, such as types, relations, and attributes; and perhaps the most challenging side of using part-of-speech tags to infer semantic constructs is that many entities, types, and relations are made up of multi-word phrases that can contain many different parts-of-speech (e.g., the relation "has won prize," or the type "Chancellors of Germany").

We model an annotated query log as set of triples $L = \{\langle Q, \pi, \sigma \rangle\}_i$ where Q is a keyword query, π is a function mapping query terms to POS tags, and σ is a function mapping query terms to semantic constructs. To classify terms via their part of speech tags, we use the standard naïve Bayes classification method. We perform naïve Bayes classification by directly estimating the joint probability distribution of POS tags and their semantic constructs from the query log. The conditional probability of a particular semantic construct C given a POS tag P is then the frequency of that query term's POS tag P mapping to C, versus the frequency of P mapping to *any* semantic construct, $\Pr(\mathrm{C}|\mathrm{P}) = \Pr(\mathrm{C}, \mathrm{P}) / \Pr(\mathrm{P})$ which is estimated from the query log by the following frequencies.

$$\Pr(\mathrm{C}|\mathrm{P}) = \frac{|\{\langle Q, \pi, \sigma \rangle \in L \ s.t. \exists q \in Q, \pi(q) = \mathrm{P}, \sigma(q) = \mathrm{C}\}|}{|\{\langle Q, \pi, \sigma \rangle \in L \ s.t. \exists q \in Q, \pi(q) = \mathrm{P}\}|}$$

With a distribution over semantic constructs given the part-of-speech of a query term, we can estimate the probability of assignments of semantic constructs to individual part-of-speech tagged query terms for a whole query. Assuming independence of query terms for tractability, this yields the following equation.

$$\Pr(AQ|Q) = \Pi_{q:\mathrm{C} \in AQ} \ \Pr(\mathrm{C}| \ \pi(q))$$

To illustrate the process, consider our example query:

"songs by jimi hendrix"
$\hookrightarrow$ *"songs"*:NNS *"by"*:IN *"jimi"*:NNP *"hendrix"*:NNP
$\hookrightarrow$ *"songs"*:type *"by"*:rel *"jimi"*:ent *"hendrix"*:ent
$\hookrightarrow$ *"songs"*:type *"by"*:rel *"jimi hendrix"*:ent

Here, NNS denotes a plural noun, NNP a proper noun, and IN a preposition.

3.1.2 Conditional Random Field Annotation

A CRF is an undirected probabilistic graphical model for labelling sequential data. Given a trained model and an input sequence, the Viterbi algorithm [101] over a CRF enables the computation of the most probable, or k most probable labellings according to the model. Specifically, a labelling is an assignment of state labels $y_1, \ldots, y_n$, to an input sequence $x_1, \ldots, x_n$, where each y_i corresponds to a state in the model and each x_i corresponds to a feature vector.

We base our CRF model on the design for noun-phrase detection proposed by Sha and Pereira [92] since our problem shares similarities. For input position x_i corresponding to query term q_i, we define a feature vector containing the following features: all query terms and POS tags in a size five window around position x_i; all query term bigrams in a size three window; all POS tag bigrams in a size five window; and all POS tag trigrams in a size five window. We include the actual query terms as features to allow important repetitive terms to be captured (e.g., "in" describing a relation such as *"restaurants in barcelona"*), but discard any generated feature that appears only once to avoid over-fitting the particular terms in the training data.

We deviate from the model of Sha and Pereira in label design. The labels must encode both the semantic constructs we want to annotate as well as the boundaries between multi-term semantic constructs. We create two output labels for every semantic construct in our chosen knowledge representation language: a "begin" (B) and a "continue" (C) label. This encoding allows us to annotate boundaries as well as semantic constructs. To generate training data, we label each multi-term phrase in the training data with the begin and continue labels corresponding the phrase's semantic construct. For example, the correct labelling of our running example is the following.

"songs":type-B *"by"*:rel-B *"jimi"*:ent-B *"hendrix"*:ent-C

Here, the query term *"hendrix"* is annotated as a continuation of the entity starting with *"jimi"*, yielding the following annotated query.

"songs":type *"by"*:rel *"jimi hendrix"*:ent

Unlike the Naïve Bayes classification approach, the CRF model can distinguish between multiple instances of the same semantic construct occurring in succession. For example, the query *"park guell barcelona"* contains two consecutive entities. The CRF output labels can express the labelling that segments the entity *"park guell"* from the entity *"barcelona"*, while the Naïve Bayes approach would merge the three tokens as a single entity. Figure 3.1 shows three examples of the CRF-based query annotation and segmentation process assuming a single (most probable) annotation is used. In practice, we consider the top-k most probable annotations when interpreting queries.

"toronto restaurants"
↪ toronto:`ent-B` restaurants:`type-B`
↪ toronto:`ent` restaurants:`type`
"author of war and peace"
↪ author:`rel-B` of:`rel-C` war:`ent-B` and:`ent-C` peace:`ent-C`
↪ author_of:`rel` war_and_peace:`ent`
"songs by jimi hendrix"
↪ songs:`type-B` by:`rel-B` jimi:`ent-B` hendrix:`ent-C`
↪ songs:`type` by:`rel` jimi_hendrix:`ent`

Figure 3.1: Segmentation and annotation process of the CRF-based approach. Annotated labels are used to recover latent structure of multi-word phrases, as well as their semantic construct classification.

We train our model using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm (see [16]), a type of hill climbing approach for solving non-linear optimization problems. To avoid over fitting we use L2 regularization. The probability $\Pr(AQ|Q)$ is then given directly by the CRF model.

3.2 Structuring Annotated Queries

An annotated query reveals part of the latent structure of an entity-based keyword query by indicating the semantic role represented by various parts of the query. However term annotation alone does not describe how these various recognized semantic constructs relate to model the underlying query intention. In our running example query, we know (after annotation) that the query contains a type, a relation, and an entity. However there is still ambiguity in what the query is seeking. Is it ultimately describing entities of the given type that are related to the given entity? Or is it seeking information about the given entity within the context of the given type?

To illustrate the ambiguity in query structure, consider the following two queries: *"john smith dentist"* and *"new york restaurants"*. Both queries contain a two-term entity followed by a single-term type. The first query seeks information about the given entity (*"john smith"*), with a type (*"dentist"*) given as context to disambiguate among possible interpretations of the entity. Whereas the second query is seeking instances of the given type (*"restaurants"*), within the context of the given entity (*"new york"*). Despite both queries having the same semantic contents (an entity followed by an entity type), they exhibit different semantic structures.

Once a keyword query has been annotated, we need to represent possible concept search structures that can encode the underlying query intent. To model high level query structure, we follow the logical connectives from the knowledge representation language.

Definition 9 (Structured Query Template) A *structured query template* T is a schema-level description of the structure of a concept search query, expressed in the following grammar

$$T \quad ::= \quad node \ \mid \ edge(T) \ \mid \ T_1 \sqcap T_2$$

where $node \in \{ent, type, val\}$ and $edge \in \{rel, attr\}$. □

A structured query template describes the overall graph structure of the query as well as the node and edge types of the query predicates. For example, the structured query template for our example query is $type \sqcap rel(ent)$.

We want to estimate the probability of a query template given a semantically annotated keyword query, $\Pr(T|AQ)$. We assume access to an annotated query log containing both semantically annotated queries and their structured query templates, $L = \{(AQ_i, T_i)\}$.

Our structuring approach directly estimates the probability of a structured template given an annotated query by aggregating over all queries in the training log that share the same high-level summary of semantic annotations.

Definition 10 (Semantic Summary) Given an annotated query $AQ = \langle q_1{:}a_1, q_2{:}a_2, \ldots, q_n{:}a_n \rangle$, a *semantic summary* is an ordered list of semantic constructs occurring in AQ, and is given by the function $S : AQ \rightarrow \mathrm{C}^n$ s.t. $S(AQ) = \langle a_1, a_2, \ldots, a_n \rangle$ □

For example, $S($*"songs"*:type *"by"*:rel *"jimi hendrix"*:ent$) = \langle type, rel, ent \rangle$.

We directly estimate $\Pr(T|AQ)$ from labelled training examples in our query log from the definition of conditional probabilities, using the semantic summary as a high level representation of the annotated query.

$$\Pr(T|AQ) = \frac{|\{\langle AQ', T' \rangle \in L \ s.t. T = T', S(AQ) = S(AQ')\}|}{|\{\langle AQ', T' \rangle \in L \ s.t. S(AQ) = S(AQ')\}|}$$

The probability of a query template given an annotated query is estimated by the proportion of queries with the same semantic summary that are structured using that template, versus the total number of queries with the same semantic summary and any structuring.

Combining a structured query template with a semantically annotated query yields a structured representation of the keyword query. Following the example query we have *"songs"* $\sqcap$ *"by"*(*"jimi hendrix"*). In Chapter 4 we explore how to exploit these semantic and structural annotations to compute mappings to candidate concept search queries representing a possible intention of the query over the knowledge base. We also describe how semantic annotation and structuring fit into a probabilistic model of scoring query interpretations in Section 4.4.

3.3 Analysis of a Web Query Log

We analyzed a sample of keyword queries available from the Yahoo WebScope program [129]. This query log contains English queries issued to the U.S. Yahoo search engine in January 2009. The log is filtered to ensure user privacy, in particular all queries in the log have been issued by at least three different users and queries containing numeric values longer than four digits are removed. We inspected queries keeping only those that have a *semantic construct* as the primary query intention, following the classification proposed in [83]. We annotated 258 queries with the part-of-speech tags and semantic constructs. We did not consider misspelled, non-English, or other queries that were not clearly understandable. Among the annotated queries, 156 queries had some semantic construct as their primary intention (*entity-based queries*). That is approximately 60% of queries having a semantic construct as the primary intent of the query. This is consistent with the analysis done in [83] which was performed over a different Web query log and reported 58% of queries having a semantic construct as the primary intent. The sample of entity-based queries have an average length of 2.94 query terms.

Our part-of-speech tag set is based on the analysis in [7], which is a reduced tag set designed for annotating Web queries. Their tag set contains only 19 tags compared to around 90 used by taggers for natural language text. Their tag set combines many tags into one representative tag, e.g., one verb tag for all forms of verbs (past, present, etc...). We extend their tag set to include a tag for plural nouns rather than group plural nouns with all other forms of nouns. Figure 3.2 shows the distribution of part-of-speech tags over query tokens as well as the percentage of entity-based queries that contain that part-of-speech. The figure also illustrates the distribution of terms with the given part-of-speech over semantic constructs. This distribution captures the relationship between part-of-speech tags and semantic constructs, which plays a key role in our annotation methods described in Section 3.1. Not surprisingly, there is a very strong correlation between proper nouns and entities. Interestingly, plural nouns are a strong indicator of a term representing

Part-of-speech	Example	% of Queries	% of Tokens	Semantic construct	Frequency
Proper noun	waterloo	77.1%	28.7%	ent	99.1%
				type	0.9%
Noun	musician	42.6%	15.9%	type	49.5%
				ent	42.7%
				attr	6.8%
				rel	0.9%
Plural noun	songs	13.6%	5.1%	type	68.0%
				ent	20.0%
				attr	8.0%
				rel	4.0%
Adjective	big	11.6%	4.3%	ent	60.0%
				type	40.0%
Preposition	in	7.8%	2.9%	rel	55.6%
				ent	44.4%
Number	2008	6.6%	2.5%	ent	53.8%
				val	46.2%
URI	yahoo.ca	5.0%	1.9%	ent	100%
Verb	run	4.7%	1.7%	ent	100%
Determiner	the	3.1%	1.2%	ent	100%
Gerund	winning	2.7%	1.0%	rel	66.7%
				type	33.3%

Figure 3.2: The ten most frequently occurring part-of-speech tags among entity-based queries, with the distribution of how frequently that tag mapped to various semantic constructs.

a *type*, while regular nouns are split between denoting *type* and *entity* labels.

From the 156 entity-based queries, we also annotated the structured query template underlying our interpretation of the intent of the query. In the cases where there were multiple possible structurings, we annotated all of them and count them as individual queries when computing the frequencies for the equations described in Section 3.2.

Figure 3.3 shows the ten most frequent structured query templates in our training query log, along with the percentage of queries exhibiting that structure (over all entity-based queries). We see that many queries tend to repeat the same structures.

3.4 Evaluation of Keyword Query Understanding

In this section we evaluate the end-to-end system in interpreting keyword queries over a large real-world Web knowledge base.

3.4.1 System Implementations

We implemented all of the techniques described in Chapter 3 for keyword query annotation and structuring, and use the methods presented in Chapter 4 as a KB mapping tool and backend knowledge base engine. In this section we treat KB mapping as a black-box operation, and explore the impact of query annotation and structuring on overall quality of query interpretation and answering. The process of KB mapping and scoring of candidate mappings is explored in Chapter 4. In particular, the integration of annotation scoring and knowledge base mapping scores is presented in Section 4.4.2. We use CRF$_{++}$ [112] to learn the CRF models. We implement a custom knowledge graph engine to support evaluating the concept search queries produced by the query understanding approaches, as described in Section 4.6. Each resulting concept search query produced by the procedure is evaluated and only those with non-empty answers are kept as admissible query interpretations.

There are a number of threshold parameters that can be used to control the search space considered by the query understanding process, including the number of semantic annotations per keyword query, the number of structurings of annotated queries, the number of knowledge base mappings of each structured keyword query, and the number of candidate knowledge base items considered for each query term when mapping. We consider all annotations with non-zero probability and the top-10 structurings unless otherwise specified. For entity search, we union the results of the top-k KB mappings for

Template	Freq.	Example Query	Summary	Structured Keyword Query
ent	44.9%	*"jimi hendrix"*	$\langle ent \rangle$	*"jimi hendrix"*
$type \sqcap rel(ent)$	12.8%	*"restaurants in toronto"*	$\langle type, rel, ent \rangle$	*"restaurants"* $\sqcap$ *"in"*(*"toronto"*)
$ent_0 \sqcap rel(ent_1)$	7.7%	*"eiffel tower paris"*	$\langle ent_0, ent_1 \rangle$	*"eiffel tower"* $\sqcap *$(*"paris"*)
$ent \sqcap type$	5.8%	*"john smith dentist"*	$\langle ent, type \rangle$	*"john smith"* $\sqcap$ *"dentist"*
$type$	5.8%	*"salsa songs"*	$\langle type \rangle$	*"salsa songs"*
$attr(ent)$	3.8%	*"albert einsteins birthday"*	$\langle ent, attr \rangle$	*"birthday"*(*"albert einsteins"*)
$ent_1 \sqcap rel(ent_0)$	3.2%	*"barcelona parc guell"*	$\langle ent_0, ent_1 \rangle$	*"parc guell"* $\sqcap *$(*"barcelona"*)
$rel(ent)$	1.9%	*"author of moby dick"*	$\langle rel, ent \rangle$	*"author of"*(*"moby dick"*)
$ent_0 \sqcap rel(ent_1, rel(ent_2))$	1.3%	*"parthenon athens greece"*	$\langle ent_0, ent_1, ent_2 \rangle$	*"parthenon"* $\sqcap *$(*"athens"* $\sqcap *$(*"greece"*))
$type_1 \sqcap rel(type_0)$	1.3%	*"protest song musicians"*	$\langle type_0, type_1 \rangle$	*"musicians"* $\sqcap *$(*"protest song"*)

Figure 3.3: The ten most frequently occurring templates among entity-based queries in the training log, along with an example of a query following that template, the summary, and the structured keyword query.

each structure with $k = 1$ unless otherwise specified. We consider the top 30 knowledge base items as candidates for query terms unless otherwise specified. For interpretations resulting in unary structured keyword queries (which do not have multiple terms to perform disambiguation) we employ a heuristic requiring 0.95 syntactic similarity for a query interpretation to qualify.

We create two system configurations, the first using the naïve Bayes (NB) term classifier for query annotation and the second using the CRF approach (CRF). Both systems use the direct approach (Drct) to structuring (Section 3.2) and the same KB mapping tool to map the computed structured keyword queries into the knowledge base. We omit structures containing more than one unknown relation to avoid the exponential blow-up caused by matching over all possible pairs of relations. We have found this does not have a significant impact on the quality of results. We implement a simple POS-tagger that builds on the Brown and Wall Street Journal POS tagged lexicon. Our tagger assigns the most frequent POS tag for each known word (suitably mapped to our reduced tag set as described in Section 3.3), and assigns proper noun to unseen tokens (the most frequent tag in our query log). We find this to be sufficiently accurate for our purposes. Building and evaluating a more sophisticated POS tagger is an orthogonal problem beyond the scope of this work.

For comparison, we also implement two alternative ways of mapping keyword queries into knowledge bases. The first method implements traditional keyword search over graphs as described in Section 2.3, with a number of heuristics taken from the literature. The process of keyword search over graphs is to find all nodes and edges that match query terms, then search among these *seed* matches to see if they can be connected. A result graph is a subgraph of the knowledge graph that spans all query terms. This can be viewed as an instance of the Steiner tree problem. Our implementation uses a breadth-first search approach to finding Steiner trees. We use the same Lucene index and graph database used by the KB mapping tool. Seed nodes are processed in order of syntactic similarity to the query term they match. The graph search will generate a given parameter number of results and return the k highest scoring. The parameter is set to 10,000 in the effectiveness experiments and is set to 10 in the performance experiments. Scoring is based on the same syntactic scoring (3-gram similarity) used by the KB mapping tool, normalized by the answer graph size to promote more compact answers.

The heuristics used in our graph search implementation include traversal of outgoing edges only, incoming edges only, bi-directional edge traversal [51], search depth limiting as recommended in [98] (we use a depth of three), and a last heuristic that forces *type* nodes to be leaves in answer graphs. The motivation for this heuristic comes from an observation that type nodes form hubs in the graph that connect many nodes, allowing query results graphs to spuriously match all query terms. Recall as an example the query *"jimi hendrix birthday"* discussed in Section 2.3. Forcing *type* nodes to be leaves in answer graphs alleviates this problem in many cases.

Note that the graph search systems return subgraphs of the knowledge base, and cannot precisely interpret queries in order to find *specific answers*. In our evaluation, we give this system the benefit of considering any answer graph as correct if it contains the answer anywhere in the graph.

The second comparison approach builds a text representation of each node in the knowledge base, then performs traditional keyword search over the text representations. We use the labels of all edges and nodes in a one hop radius around each node to form the text representation, then index these text representations using Lucene.

3.4.2 Data and Workload

We use YAGO [96] as our knowledge base, a high quality fact collection with a rich type hierarchy over entities and a part of the Linked Data Web. YAGO contains approximately thirty million assertions over two and a half million entities. The knowledge graph contains around thirteen million nodes including constants. Our training data is the Yahoo query log described in Section 3.3. To evaluate our approaches, we use both queries from the query log and a hand-crafted keyword query workload designed with specific properties we wish to evaluate. The query workload consists of 96 queries, half of which are manually created to have an underlying query intention that *can* be modelled with the data in YAGO, and half taken from the Yahoo log that describe data *not* occurring in YAGO. The half that have YAGO answers, called *positive queries*, are created to exhibit the 12 most frequent structures found in our analysis, accounting for over 90% of entity-based queries. We create four examples of each structure. The second half are entity-based queries taken from the Yahoo log for which we have manually verified that no interpretation exists in YAGO. We refer to these as *negative queries*.

The gold standard result for positive queries is defined by manually constructing a structured query over YAGO, and the correct answer for negative queries is defined as the empty set. This query workload explicitly exercises both of the problems of interpreting queries when possible, and recognizing when an interpretation is not possible. The workload intentionally places equal importance on both problems. We manually create the queries to ensure all structures are represented, and to control for KB coverage. Finding examples of queries with all of the desired structures in a Web query log, such that they also have intentions existing in YAGO would require a huge manual effort, and possibly a larger query sample. Existing benchmarks also do not capture all of the structures we want to evaluate while controlling for KB coverage. The positive queries from the workload are available in Appendix A.1, the negative queries are available as part of Yahoo's WebScope program [129].

3.4.3 Effectiveness Results

To evaluate the effectiveness we conducted a direct evaluation of the query annotations, an evaluation of the KB interpretations produced, and an end-to-end evaluation of the full system in finding exact answers to keyword queries.

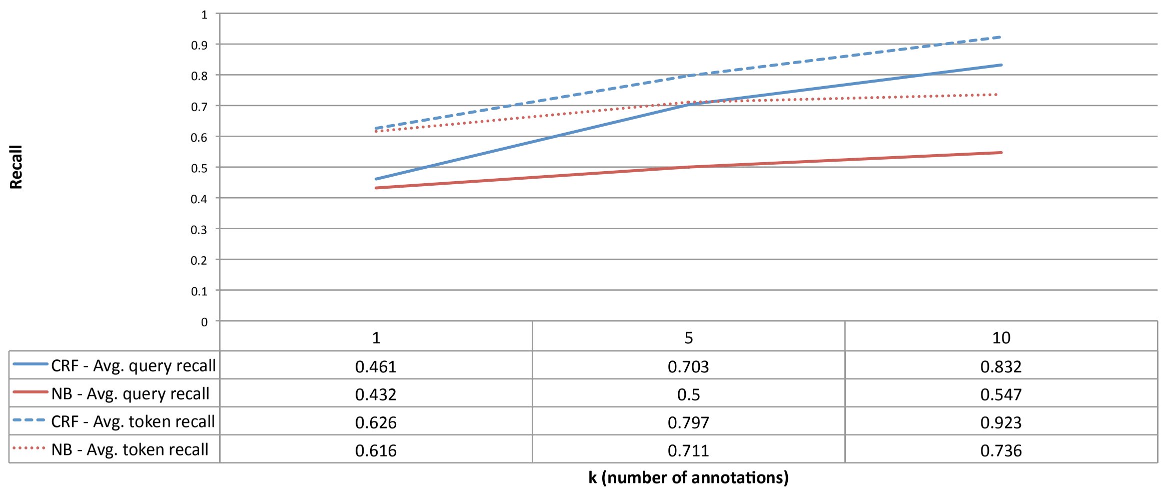
We employ a number of measures for our evaluation, based on the methodology presented in Chapter 6. Precision is defined as the fraction of returned results that are correct. Recall is defined as the fraction of all possible correct results that get returned. MRR measures the (reciprocal) average rank where the (first) correct answer occurs. Given a set of queries $\mathcal{Q}$ and a function $rank(i)$ that returns the rank of the first correct answer for query i, MRR is given by the following.

$$\text{MRR}(\mathcal{Q}) = \frac{1}{|\mathcal{Q}|} \sum_{i}^{|\mathcal{Q}|} \frac{1}{rank(i)}$$

For all measures, we count a value of 0 if a system does not return any result for a positive query or if any result is returned for a negative query. A system that always returns the correct answer at rank one would have an MRR of 1.0, always returning the correct answer at rank two would produce an MRR of 0.5, and so on.

Annotation Accuracy

Figure 3.4 shows the results of both the NB and CRF annotation approaches. Results are averages using 10-fold cross validation over the 156 entity-based queries from the Yahoo query log. We measure average query recall, the frequency in which the top-k annotations of a query generates the correct annotation, for varying values of k. The correct annotation means that all query terms are annotated correctly. The figure also shows average token annotation accuracy (token recall) for the best annotation in a query's top-k annotations. This is the fraction of query terms annotated correctly for the best annotation produced for each query. This is relevant because our structuring and KB mapping tools are tolerant to errors, meaning an incorrect segmentation or annotation can still produce a correct concept search query. While both approaches are comparable at k=1, the superiority of the CRF approach is clear as we consider the top-5 to top-10 annotations.



	1	5	10
CRF - Avg. query recall	0.461	0.703	0.832
NB - Avg. query recall	0.432	0.5	0.547
CRF - Avg. token recall	0.626	0.797	0.923
NB - Avg. token recall	0.616	0.711	0.736

Figure 3.4: Average number of queries with the correctly annotated query occurring in the top-k annotations (query recall), and average number of correctly annotated tokens for the best annotation occurring in the top-k (token recall).

CRF.Drct	Incorrect - "false"	Correct - "true"	Total +/−
Interpretation - "positive"	3 (3.1%)	44 (45.8%)	47 (49.0%)
No Interpretation - "negative"	4 (4.2%)	45 (46.9%)	49 (51.0%)
Total false/true	7 (7.3%)	89 (92.7%)	

NB.Drct	Incorrect - "false"	Correct - "true"	Total +/−
Interpretation - "positive"	3 (3.1%)	36 (37.5%)	39 (40.6%)
No Interpretation - "negative"	12 (12.5%)	45 (46.9%)	57 (59.4%)
Total false/true	15 (15.6%)	81 (84.4%)	

Figure 3.5: Confusion matrices for the query interpretation task. Raw counts and percentages are shown for the true/false positive/negatives, as well as the marginals. The 96 query workload contains 48 positive queries and 48 negative queries.

Query Interpretation

There are four possible outcomes when interpreting a query. The first is that the query is interpreted over the KB, and a correct interpretation is found (true positive); the second is that the query is interpreted but all interpretations are incorrect (false positive); the third is that the query does not get interpreted though an interpretation does exist in the data (false negative); and the last is that the query does not get interpreted because no interpretation exists in the data (true negative). Figure 3.5 shows the confusion matrices for these outcomes for both the CRF and NB approaches when considering the top-10 structures and no bound on the number of query mappings per structure. A result is taken to be correct if a correct interpretation is produced somewhere in the result list of query interpretations. This experiment gives a view the quality of the search space explored by our query interpretation algorithm with default parameter settings. The figure shows that the CRF approach is superior with respect to both types of error, and exhibits a higher overall true positive rate (correct interpretations) than the NB approach. Overall the CRF-based approach produces a correct interpretation or recognizes an uninterpretable query almost 93% of the time. We found the CRF's ranked list of query interpretations to have an MRR of 0.698 on the positive query set. This means that the correct interpretation is found generally in the first or second position on average. The NB approach had an MRR of 0.589, also slightly better than the second position on average. Figure 3.6 shows the confusion matrix for the CRF.Drct approach with no bounds on any parameters (KB item candidates per query term, number of structures per annotation, and number of KB

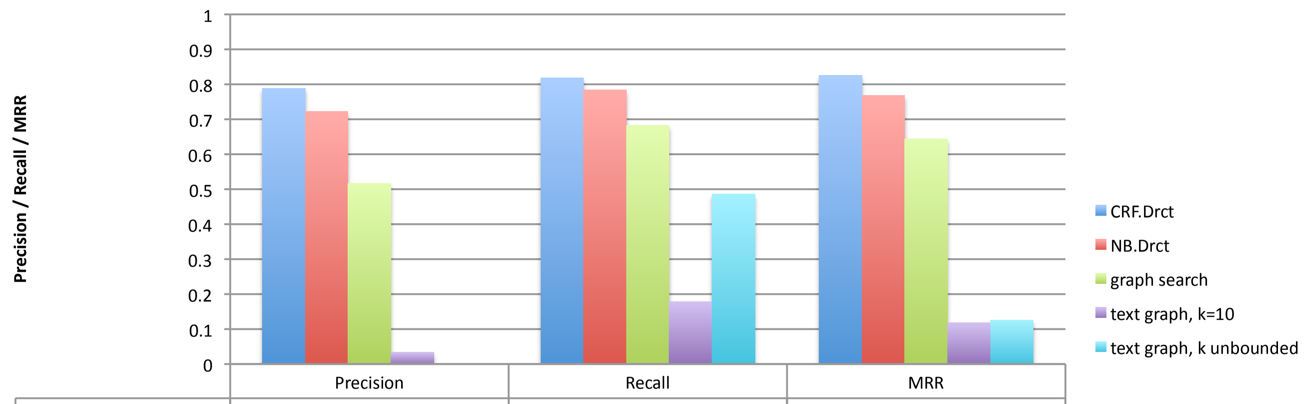
CRF.Drct	Incorrect - “false”	Correct - “true”	Total +/−
Interpretation - “positive”	11 (11.5%)	46 (47.9%)	57 (59.4%)
No Interpretation - “negative”	2 (2.1%)	37 (38.5%)	39 (40.6%)
Total false/true	13 (13.5%)	83 (86.5%)	

Figure 3.6: Confusion matrices for the query interpretation task with unbounded thresholds. The 96 query workload contains 48 positive queries and 48 negative queries.

mappings per structure). We can see that opening up the full search space reduces false negatives and increase true positives, meaning that correct interpretations are found more often for positive queries. However, this comes at the expense of finding more coincidental matches for negative queries. Exploring the complete search space is also much more expensive than computing top-k interpretations. This motivates the use of thresholds for pruning the search space of query interpretations as described in Section 3.4.1.

Keyword Query Answering

We used the workload outlined in Section 3.4.2 to evaluate the ability of the proposed approaches to interpret and answer keyword queries. Figure 3.7 shows the results of each of the systems, as well the best run for each measure by *any* of the graph search system's configurations. The superior CRF-based annotator paired with the the semantic summary-based structuring approach has the best performance across all metrics. This is a very promising result as this feature rich model is trained with a relatively small number of training examples. While the NB approach is comparable, the improved annotation accuracy of the CRF approach yields better results and returns correct answers at higher ranks. The graph search approach achieves reasonable recall and MRR, meaning it can find many correct answers and return good answers at decent ranks, however the answer set also becomes polluted with many non-relevant results hurting precision. The low performance is in part due to spurious matches for negative queries that do not actually have an answer existing in the knowledge base. The approach has no capacity to determine which results are proper answers and which are coincidental keyword matches. Also, the graph search is given the advantage of not having to find the precise answer to queries (any answer graph containing the query answer is considered correct). Not surprisingly, the text graph approach has very low precision as it is designed to find resources related to the query terms and does not attempt to interpret queries in order to find exact answers. Precision is never higher than 0.1 for any value of k. While the unbounded text graph



	Precision	Recall	MRR
🟦 CRF.Drct	0.789	0.819	0.825
🟥 NB.Drct	0.722	0.784	0.768
🟩 graph search	0.516	0.682	0.644
🟪 text graph, k=10	0.034	0.178	0.118
🟦 text graph, k unbounded	0	0.486	0.125

Figure 3.7: Average Precision, MRR, and Recall for the query answering task over the combined positive/negative query workload.

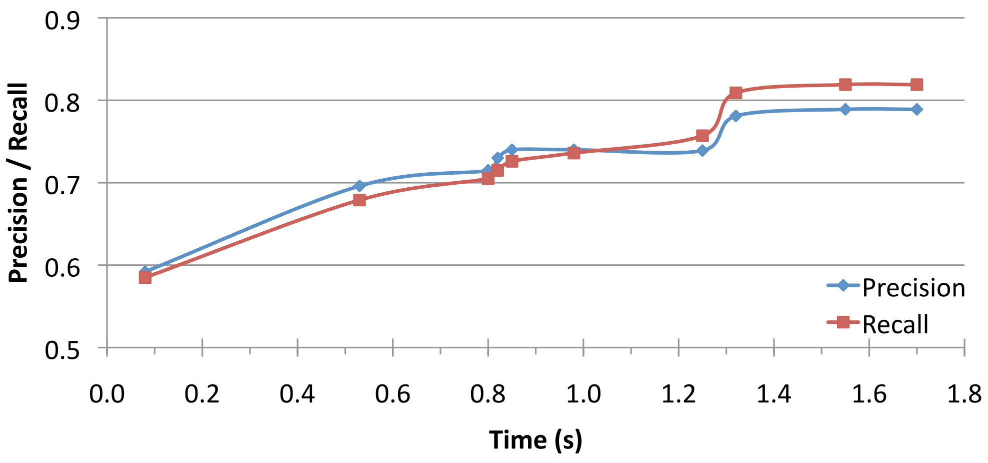


Figure 3.8: Average precision and recall vs. average run time for varying number of candidate query structures.

approach achieves 97% recall on the positive query set, it has no means of detecting the negative queries and answers each incorrectly, bringing down the average considerably.

3.4.4 Efficiency Results

Precision and recall can be traded off for run-time performance by varying the number of query interpretations that are considered for each query. Figure 3.8 shows the effect of varying the total number of query structures considered from top-1 to top-10 for the CRF.Drct approach, taking the best KB mapping for each structure. As the data points move from left to right, the total number of structures is increasing. This gives an increase in precision and recall as we explore more of the search space to find the correct query interpretation, but at the expense of greater run times. Occasionally an incorrect interpretation is found causing a dip in precision. It is important to note that the lower precision configurations of our system are most often due to queries *not being answered*, as opposed to being answered incorrectly. The highest precision (0.789) is obtained for a configuration that takes 1.5 seconds per query on average. An average precision of approximately 0.74 can be achieved in sub-second query processing time.

Figure 3.9 shows a breakdown of the average run-time for each component of the system. The actual annotation and structuring time accounts for only a small percent of the total run time, with the majority of time spent mapping candidate structures into the knowledge

System	Struct	Map	Exec	Total
CRF.Drct	0.002s	1.23s	0.32s	1.55s
NB.Drct	0.001s	0.94s	0.29s	1.24s
graph search	-	-	-	67.27s
text graph, $k = 10$	-	-	-	0.33s
text graph, k unbounded	-	-	-	17.33s

Figure 3.9: Average run-times for annotation and structuring (Struct), KB mapping (Map) and execution (Exec).

base. We further explore the performance of this component in Chapter 4. The graph search approach ($k = 10$) took around 67 seconds on average, often hitting an imposed two minute timeout. This is in part due to the graph search attempting to searchwalk the entire search space if it is unable to find k results. The text graph approach is very efficient at $k = 10$, but at the cost of effectiveness.

3.5 Related Work

The problem of interpreting keyword queries over knowledge bases bears relevance to many areas, including semantic search, keyword search over structured data, and question answering over knowledge bases. In this section we discuss the most relevant related research in these and other areas.

Semantic/KB-based Search

Guha et al. first proposed the "semantic search" problem, which aimed to augment document results with semantic Web data [41]. They proposed the TAP system, which scrapes and publishes RDF triples, then augments document search results by displaying entities that match query terms. Early work by Mayfield and Finin also proposed keyword search over RDF by using a text index over RDF nodes [75]. Bast et al, also proposed integrating knowledge into text search, but with a more explicit accounting for knowledge base semantics [8]. In their ESTER system, documents are annotated with entity occurrences and the semantic types of these entities. Query terms are then matched against knowledge base entities and types, allowing the retrieval of documents mentioning specific entities or entities of a given type. A dynamic user interface allows users to disambiguate among possible matches. However, keyword mappings are based solely on individual entity types and entities, and no attempts to compose complex concepts from sets of keywords are made. Rocha et al. also map keywords into knowledge base items and treat the knowledge base as a general graph to use a spreading activation algorithm to find additional relevant knowledge base items related to the search terms [89].

Tran et al. [98, 99] have explored mapping keyword queries into conjunctive queries by matching terms against knowledge base items and searching for connections among the matches. This approach is similar to our graph search approach used in the evaluation, though their top-k algorithm may explore less of the total search space than our BFS approach and thus be more efficient. Zhou et al. have also proposed a similar data driven approach [110]. These techniques are in contrast to our query-log driven approach which elicits structures from the information needs of real users.

Lei et al. use manually defined templates to map query terms into formal structured queries over a knowledge base, and employ heuristics to reduce the search space when the query contains more than two terms [62]. This is combined with an entity index over a text collection to provide a semantic search capability. Fagin et al. [38] use user specified grammars and vocabularies to annotate query terms with type and attribute labels. Our proposed approach could possibly be used to learn these types of grammars from a query log.

Farfan et al. used knowledge bases to enhance keyword search over XML encoded medical records that also encode knowledge base references [39]. Their work assume unambiguous annotation of XML records with knowledge base identifiers. Castells et al. proposed a semantic text search engine which takes structured RDQL queries and returns documents annotated with entities in the result of the query [20]. The work was later extended to process natural language questions [69]. The Swoogle search engine by Ding et al. takes a different view of the knowledge based search problem by indexing and searching RDF documents directly, returning RDF documents as results [32].

Blanco et al. have used relevance based retrieval to search for nodes in large knowledge bases [13]. A statistical language model based approach was presented by Elbassuoni and Blanco [35]. These approaches however do not attempt to interpret the semantics of queries, and thus do not necessarily provide exact answers to keyword queries. Instead the goal of these systems is to provide a ranked list of resources relevant to the query terms.

Term annotation

One form of query understanding is to try and annotate individual query terms with semantic meaning. From a linguistic query understanding point-of-view, there has been work on understanding the senses and parts-of-speech (POS) of query terms. In particular, Barr et al. showed that relatively accurate POS tagging of Web queries was possible, achieving 83% tagging accuracy over a large query log [7]. Work has also been done in deriving the word sense of ambiguous query terms, for example in [68]. Li proposed a method of determining the *intent head* of a query, i.e., the part of the query that represents what the user is looking for, leaving the remaining query terms as context or *intent modifiers* [64]. For example, the query *"Pet Cemetery author"* has *"author"* as the intent head, since the query is ultimately searching for an author. Our running example query would have *"songs"* as the intent head, since the query is seeking songs with particular qualifications specified by the context. While annotations of individual terms can provide insight into the query semantics, these works do not address how terms relate to form single coherent expressions of the underlying query intent. Guo et al., explored annotation of named entities in keyword queries [42], this is a subset of our annotation problem which aims to annotate types, relations, attributes, and attribute values in addition to named entities.

From a knowledge base point-of-view, Paparizos et al., have investigated using a large heterogenous product KB to answer Web queries [79]. As part of this initiative, Sarkas et al., proposed annotating query terms as being instances of attributes values using a probabilistic model based on value distributions in the data [91]. This work was then extended to produce more accurate annotations by comparing attribute value distributions to term distributions over a large query log [84]. A similar goal was achieved by Manshadi and Li, by using a probabilistic hybrid parser based on probabilistic context-free grammar rules [74]. In addition to attribute annotations of query terms, their proposal also allowed terms like *"cheap"* to be annotated with operational annotations like "SortOrder." While these approaches annotate query terms with schema labels, they do not address how individual terms fit together to form a single semantic representation of query intent.

A similar theme was explored by Agarwal et al., where query templates (like grammar rules) were mined from a query log [1]. Templates would generalize terms that match KB items, for example *"jobs in #location"*. Web-sites clicked for particular queries were then aggregated over all queries matching templates, allowing the computation of relevance of Web-sites to templates. Unseen queries that match templates can then be answered by returning the sites relevant to the template. Our template mining is similar in nature that of Agarwal, though our templates fully specify semantic structure and not just types occurring in keyword queries. Also, we do not rely on input query terms exactly matching a set of known types, but instead match against syntactically similar types and resolve ambiguity from the query context. Our Web knowledge base setting also yields over 280,000 possible types. Cheng et al. allowed user queries to specify a similar form of template, which explicitly specified the intended return type [23]. For example, the query *"#phone-number amazon customer support"* would specify a search for instances of the type "phone-number" in the context of the given keywords. Types are unambiguously specified and are simple unary type specifications, not complex expressions built up from primitives as in our case. Kandogan et al., proposed the AVATAR semantic search system which translates individual keywords to possible KB items based on matching keywords provided by domain experts for each KB item [53].

Search over Structured Data

Search over relational data can be viewed as a form of semantic query understanding. Query terms are mapped into structured data producing structured queries that can be used to retrieve data that contains the queried terms. These structured queries can be viewed as semantic interpretations of the original keyword query. There is a long line of research for search over structured data, starting with the DISCOVER [45] and BANKS systems [10] that aim to efficiently find tuples that span all query terms, as well as extensions that aim to integrate IR style ranking [44] or augment the graph traversal algorithm [51]. The heuristics used in our graph search approach are based on this line of research. Tata et al. extend keyword search over structured data to include aggregates and top-1 queries [97]. Queries are translated to SQL with the allowance of various aggregation keywords such as *num* or *avg*.

Question Answering

The problem of semantic query understanding shares a similar goal as the problem of natural language question answering (QA). The QA problem aims to find specific answers to questions, which inevitably implies constructing a model of the intention of the query. QA approaches generally depend on either having properly formatted natural language questions that can be linguistically parsed to take advantage of the grammatical structure when building the semantic interpretation, or they depend on having the relatively large amount of text (as compared to Web queries) from the question in order to match against reference knowledge. Katz et al. have built the OMNIBASE+START system that answers questions using a knowledge base extracted from the Web [56, 55]. OMNIBASE consists of an extracted knowledge base of object-property-value triples, which are queried after identifying a target data source, an entity and a desired property from the query (e.g., imdb, "Gone With the Wind", DIRECTOR). Similarly, the TextRunner project has provided a search service over a large set of triples extracted from Web text [5]. This project focuses more methods to extract knowledge bases, but does include a flexible demo query interface for TextRunner that parses natural language questions into triple patterns to match against data triples. TextRunner has been succeeded by the ReVerb project [37], which does not yet provide any special query capabilities.

The idea of exploiting existing Web knowledge bases for QA has also been explored by Fernandez et al. by using the PowerAqua question answering system [40, 69]. PowerAqua integrates a front-end QA system on a semantic search back-end to answer questions over Linked Data knowledge bases. Lopez et al. have recently conducted a survey on using semantic Web knowledge bases for question answering [70].

Yahya et al., have also recently proposed an approach to answering natural language questions over Web knowledge bases [106]. Their approach (published in parallel with our approach [80]) follows a similar methodology to our approach, where questions are segmented and mapped to knowledge base items followed by an evaluation of semantic coherence to retain only meaningful mappings. Their approach uses a dependency parser to determine parts of the structure, this does not directly apply to keyword queries which do not have the rich linguistic structure of natural language questions. They also rely on query terms mapping to each part of the underlying query interpretation and do not consider the possibility of latent relations existing in the query interpretation. Such considerations are necessary to apply interpretation techniques to keyword queries.

While QA has a similar high level goal to our task, QA approaches generally depend on having properly formatted questions that can be parsed to take advantage of the grammatical structure; having a large amount of query text (as compared to keyword queries) to match against answer candidates; or having large amounts of answer text to match query terms or exploit redundancy. Properly formatted questions allow a QA system to determine a great amount of information about the question intent and candidate answers, before even looking at the data. For example, many QA systems will perform dependency parsing, answer type identification, and head/key noun-phrase detection. Having large amounts of query text (as compared to three to four term keyword queries) is also often used match questions against answer candidates extracted from passage retrieval [109]. An abundance of text can also be used to exploit redundancy to discern among multiple answer candidates [25]. Keyword queries over knowledge bases on the other hand do not allow deep question analysis, and both keyword queries and knowledge bases contain very little text for frequency-based matching algorithms. Due to these core differences question answering approaches are generally not directly applicable to the problem of keyword query answering. We omit a comprehensive overview of question answering approaches, Lopez et al. have recently conducted a survey on using semantic Web knowledge bases for question answering [70] while Kolomiyets and Moens give an overview of general IR-based question answering approaches [57].

The problem of semantic parsing shares a similar high level goal with semantic query understanding. Both problems aim to construct a formal and unambiguous representation of the underlying meaning of an ambiguous input. Recent work has explored applying semantic parsing to natural language queries [107, 65]. These approaches parse natural language queries to lambda calculus representations of the query intent. While these approaches do produce rich semantic interpretations of the input queries, they greatly depend on having fully specified natural language inputs with proper grammatical structure. These queries contain large amounts of text in comparison to Web keyword queries. Also, these approaches are applied to small controlled domains with heterogeneous knowledge bases, as opposed to the massive heterogeneous knowledge bases found on the Web.

3.5.1 Query Categorization

Query categorization (or classification) is another form of query understanding, where queries are classified into categories indicating the target result types. For example, queries may be classified as product search queries, political news queries, or sports queries. While the categorizations do not represent semantic understanding of the query intent, they do aid in judging the relevance of a candidate result or in selecting a data source for search. Knowledge bases have been used to drive query categorization, Hu et al. showed that query terms can be matched against Wikipedia categories to derive query intentions [47]. Wang et al. showed that loosely structured data in HTML lists could be exploited to learn semantic classes for Web queries [103].

Chapter 4

Graph Queries with Keyword Predicates

Structured graph queries specify patterns to be matched against a knowledge graph. Formalisms like the concept search queries defined in Section 1.3 also exploit semantics encoded in an underlying schema. In this chapter we explore extending graph queries based on concept search queries to contain keyword predicates instead of proper well-formed knowledge base predicates. Such a decision introduces ambiguity into the query, which must be efficiently resolved by reasoning over possible mappings of the keyword terms into the knowledge base predicates. This chapter presents work originally published in [81] and [82].

Graph queries with keyword predicates may be explicitly issued to a system by an expert user. Such a user is capable of formulating concept search queries, but is unable to use proper graph predicates due the massive size of the knowledge base schema typical of Web knowledge bases. Alternatively, these queries may serve as an internal representation language for modelling the semantics of keyword queries as described in Chapter 3. Combining an annotated query with a structured query template yields a structured version of the keyword query. Mapping this structured graph query into the knowledge base, while satisfying the keyword constraints corresponds to step 3 of our query understanding process illustrated in Figures 2.1 and 2.2. Combining the annotated running example query with its template gives the following query

$$\textit{“songs”} \sqcap \textit{“by”}(\textit{“jimi hendrix”})$$

by swapping the annotated constructs into their respective positions in the template. (we use numbering to avoid ambiguity in templates that contain multiple instance of the same type of construct, such as a query template with two entities). We refer to this type of expression as a *structured keyword query*, which we define in the following chapter. We also present an efficient algorithm for mapping structured keyword queries into a knowledge base, producing a concept search query in the vocabulary of the knowledge base.

4.1 Motivation

Knowledge bases such as those discussed in Section 1.2 are heterogenous and contain large numbers of resources. As an example, both ExDB [19] and YAGO [96] have schema items numbering in the millions, and fact collections numbering in the hundreds and tens of millions respectfully. WebTables has over five million unique attribute names in over two million unique relational schemas [18]. At this scale, writing structured queries can be a daunting task as the sheer magnitude of the information available to express the query is overwhelming. We call this the *information overload* problem.

To deal with this problem, and also open a new form of exploratory search, recent work has brought keyword query processing to structured data models, such as keyword search over relational databases. While this alleviates the information overload problem caused by massive schemas, it comes at a loss of expressivity. Users can no longer express desired structure in the query, and can no longer explicitly take advantage of schema information.

As an example, consider a user who wants to find *all guitarists who have won a Grammy.* The user may pose the following concept search query to an information system.

$$\texttt{GUITARIST} \sqcap \exists \mathit{hasWonPrize}\,(\texttt{GRAMMY\_AWARD}) \tag{4.1}$$

If the user is not familiar with the schema of the underlying information system (i.e., they do not know the labels of the knowledge base items in order to formulate a well formed query), then they may alternatively issue the following keyword query.

$$\textit{``guitarist has won grammy''} \tag{4.2}$$

This query searches for all data items with a syntactic occurrence of the given keywords. There are two important differences between the structured query and the keyword variant. First, the structured query will be processed by making use of explicit semantics encoded as an internal schema. In particular, the query will find data items that are a type of guitarist, rather than those which contain the keyword *"guitarist"*. Second, the keyword query will look for data items with any syntactic occurrence of *"grammy"*, while the structured query uses this as a selection condition over the qualifying entities to find those that have won a grammy, exploiting the structure of the query to find qualifying results.

This example illustrates how query languages can vary in their expressiveness and ease-of-use. On one end of the spectrum, structured query languages, such as SQL or SPARQL, provide a mechanism to express complex information needs over a schema. A user of such a language must have intimate knowledge of the underlying schema in order to formulate well formed queries for arbitrary information needs. On the other end of the spectrum are free form query languages such as keyword queries. These languages allow users to express information needs with little to no knowledge of any schematic constructs in the underlying information system, giving ease-of-use as well as useful exploratory functionality to the user.

Consider the design of a query language that falls somewhere in the middle of this spectrum. We would ideally like to retain the expressive structure of structured queries, while incorporating the flexibility of keyword queries. One way to achieve this is to embed ambiguity into the structured query language by allowing keywords to take the place of entities or relations. In this setting the query from the previous example may be written using structure as in Equation 4.1, but with keywords as in 4.2.

$$\textit{``guitarist''} \sqcap \textit{``has won''}(\textit{``grammy''})$$

This approach keeps the flexibility of keywords, but allows structure using conjunctions and nesting with relations. In this model each keyword phrase will be replaced, by the query processor, with a set of candidate schema items based on some metric of syntactic matching, such as keyword occurrence or edit distance as depicted in Figure 4.1. Then, a mapping of each query term to a knowledge base item will be chosen to find a semantically coherent interpretation of the query. The advantage of this approach is in the flexibility of how queries can be expressed. Users need not have intimate knowledge of the underlying schema to formulate queries as is necessary with traditional structured query languages. At the same time, the query retains explicit structure that gives greatly increased expressiveness over flat keyword queries.

The problem with this approach is that the number of possible (disambiguated) queries is exponential in the length of the query. This is problematic as many of the possible matchings may be meaningless with respect to the underlying schema, or may not represent the users actual intention. Processing every possible match is not only inefficient, as many of the possible query interpretations may have empty result sets, it can overload the user with many unwanted results from unintended interpretations. For example, the query for all `GUITARIST` who have won an `AWARD` (the super class of all awards) would produce

$$\begin{array}{ccccc} \textit{“guitarist”} & \sqcap & \textit{“has won”} & & (\textit{“grammy”}) \\ \left\{\begin{array}{c}\texttt{GUITARIST}\\ \texttt{Guitar}\\ \texttt{STEEL\_GUITARIST}\end{array}\right\} & \sqcap & \left\{\begin{array}{c}\textit{hasWeight}\\ \textit{hasWonPrize}\end{array}\right\} & & \left(\left\{\begin{array}{c}\texttt{GMM\_Grammy}\\ \texttt{GRAMMY\_AWARD}\\ \texttt{The\_Grammy\_Family}\end{array}\right\}\right). \end{array}$$

Figure 4.1: An ambiguous concept search query and the possible mappings of each keyword to a knowledge base item.

many results that obstruct the user from finding those of interest (those who have won a GRAMMY_AWARD in particular). Note that syntactic matching alone could not possibly be sufficient as a disambiguation solution in all cases. For example, the term *“grammy”* could describe the award, the name of a company, or the name of a family among other possibilities. Also a term such as *“spanish”* is a natural choice of word to express the both a nationality and a language (e.g., consider the query *“spanish speaking countries”* vs. the query *“spanish football players”*). This phenomena, known as *polysemy*, is just one of the challenges of disambiguation. Similar problems arise with synonymy, when multiple different terms can be used to describe the same conceptual thing.

In this chapter, we investigate a solution to the problem of querying large heterogeneous schemas by introducing a structured query language that builds upon keywords as its most basic operator, while taking disambiguation steps before query evaluation in order to avoid the exponential blow-up caused by keyword ambiguity. Our query language allows a natural keyword-based description of entity-relationship queries over large knowledge bases without intimate background knowledge of the underlying schema, while allowing complex query structure to be explicitly specified.

4.2 Structured Keyword Queries

A structured variation of keyword queries may serve as an end-user query language to users with the expertise to formulate structured queries, or as an intermediate internal representation language for modelling the semantics of keyword queries as presented in Chapter 3. Combing an annotated query with a query template yields a structured query with keywords as predicates. For example, combining the semantically annotated query from our running example $\langle$ *“songs”*:type *“by”*:rel *“jimi hendrix”*:ent$\rangle$ with the template structure $type \sqcap rel(ent)$, gives the following query *“songs”* $\sqcap$ *“by”*(*“jimi hendrix”*) by swapping the annotated constructs into their respective positions in the template We call this type of expression a *structured keyword query*, which we define as follows.

Definition 11 (Structured Keyword Query) Let $k \in \mathbb{S}$ be a keyword phrase (one or more keywords), then a *structured keyword query* SKQ is defined by the following grammar.

$$\begin{array}{rcl} SKQ & ::= & k \\ & | & k(SKQ) \\ & | & SKQ_1 \sqcap SKQ_2 \end{array}$$

$\square$

The first construct allows an entity or entity type in a query to be described by a set of one or more keywords (e.g., "*grammy award*"). The second construct allows one to describe an entity in terms of the relationships it has to other entities or types. (e.g., "*born in (liverpool)*"). The third construct allows a set of queries to describe a single class of entities in conjunction (e.g., "*harmonica player* $\sqcap$ *songwriter*").

Note that the relation query constructor allows an arbitrary query to describe the entity to which the relation extends, e.g., "*born in(country* $\sqcap$ *has official language(spanish))*" could be used to query all entities born in spanish speaking countries.

4.3 Matching Subgraphs with Keyword Predicates

Because our structured keyword query language is built on keywords as primitives, there is an inherent ambiguity in the semantics of the query. The user's intentions are unknown since each keyword in the query could map to one of many items from the knowledge base, and there is an exponential number of ways (in the length of the query) in which we could select an item for each keyword to create a general concept. Our approach to disambiguating the structured keyword query is to compute mappings of the query expression into the knowledge base. Such mappings aim to maximize the similarity of query terms to knowledge base item labels while at the same time composing a coherent concept with respect to the underlying knowledge base semantics.

4.3.1 The Disambiguation Model

Consider a structured keyword query SKQ. Each keyword in SKQ could have many possible KB item matches based on some measure of syntactic similarity. Let $M(k)$ denote the set of possible KB item (entity/type/relation/attribute/constant) mappings of keyword k.

The goal is to map each query term to one KB item, such that the resulting mapping produces a meaningful disambiguated representation of the query with respect to the knowledge base. There is a tradeoff that must be considered when mappings for each keyword: we want mappings that stay true to the user's intention as best as possible in terms of syntactic similarity, but which also have a meaningful interpretation in terms of the underlying knowledge base by means of semantic coherence.

We encode our disambiguation problem as a graph which represents the space of all possible interpretations that bind one knowledge base item to each keyword phrase. In the discussion that follows, we refer to a concept or the subgraph that denotes a concept interchangeably.

Definition 12 (Disambiguation Graph) Let SKQ be a structured keyword query. Then a *disambiguation graph* $G = \langle V, E \rangle$ with vertex set V and edge set E, and P a set of groupings of the vertices in V, are given by the following.

$$\begin{aligned} V &= \bigcup_{k \in SKQ} M(k) \\ P &= \bigcup_{k \in SKQ} \{M(k)\} \\ E &= edges(SKQ) \end{aligned}$$

where the edge generating function $edges(SKQ)$ is defined as follows.

$$\begin{aligned} edges(k) &= \{\} \\ edges(k(SKQ)) &= \left(\bigcup_{\substack{n_1 \in M(k), \\ n_2 \in root(SKQ)}} \langle n_1, n_2 \rangle \right) \cup edges(SKQ) \\ edges(SKQ_1, SKQ_2) &= \left(\bigcup_{\substack{n_1 \in root(SKQ_1), \\ n_2 \in root(SKQ_2)}} \langle n_1, n_2 \rangle \right) \cup edges(SKQ_1) \cup edges(SKQ_2) \end{aligned}$$

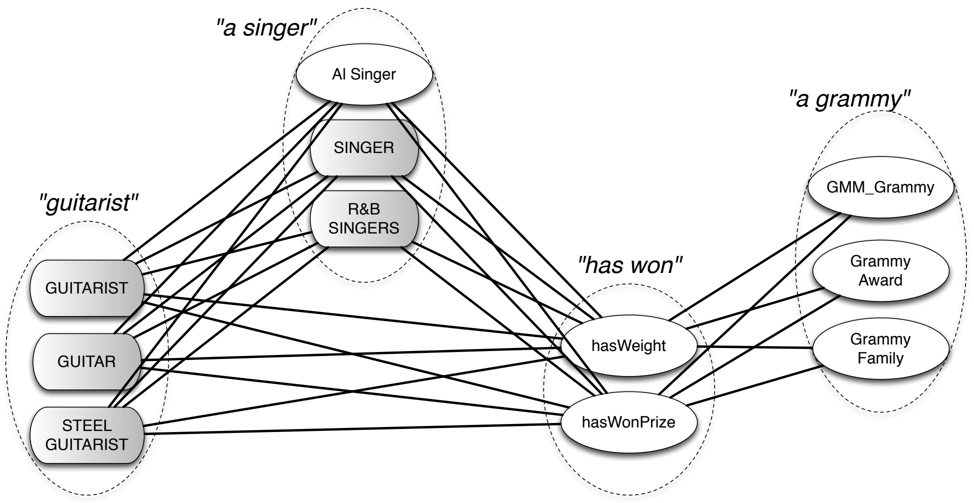


Figure 4.2: An example query disambiguation graph.

where $root(SKQ)$ denotes the vertices in the root level query of SKQ as follows.

$$\begin{aligned} root(k) &= M(k) \\ root(k(SKQ)) &= M(k) \\ root(SKQ_1, SKQ_2) &= root(SKQ_1) \cup root(SKQ_2) \end{aligned}$$

□

Figure 4.2 depicts an example of a disambiguation graph for the structured keyword query *"a singer ⊓ guitarist ⊓ has won (a grammy)"*, which could be input by a user or generated by the means described in Chapter 3 from some keyword query. The candidate KB items are represented as vertices, the dotted circles denote vertex groups P with groups for relation terms in a double lined dotted circle, the quoted italic font labels denote the keywords for which the group was generated (k_i), and the interconnecting lines denote edges which represent semantic similarity. Observe that each level of any subquery with t conjuncts is fully connected t-partite, while edges do not span query nesting boundaries. This is because nested concepts do not occur in conjunction with concepts at other nested levels. They are related only through the vertices that denote the explicit knowledge base relations.

For a disambiguation graph G generated from a structured keyword query SKQ, any induced subgraph of G that spans all vertex groups in P corresponds to a concept interpretation of SKQ. For example, the induced subgraph spanning the vertices

$$\{\texttt{SINGER}, \texttt{GUITARIST}, \mathit{hasWonPrize}, \texttt{Grammy\_Award}\}$$

corresponds to the concept

$$\texttt{SINGER} \sqcap \texttt{GUITARIST} \sqcap \exists \mathit{hasWonPrize}(\{\texttt{Grammy\_Award}\})$$

It is evident that the space of possible query interpretations is exponential in the size of the query. Finding the "best" or top-k interpretations will depend on how we compute a score for a candidate subgraph corresponding to a query interpretation.

4.3.2 The Scoring Model

Now that we have established a model for generating a space of candidate query interpretations, we need to define the notion of a score in order to compute the top-k interpretations. We start by reviewing notions of semantic and syntactic similarity which will form the basis of our scoring function.

Semantic Similarity

The first factor of our scoring model is *semantic similarity*, which we denote generically using $semanticSim(A, B)$ for the similarity between knowledge base items A and B. For the purpose of similarity, we consider every knowledge base item to be a set. An entity is a singleton set containing the entity, an entity type denotes the set of entities of that type, a relation denotes all entities with an assertion containing that incoming relation, and an inverse relation denotes the set of all entities with that outgoing relation (attributes follow the same convention as relations).

The problem of computing the similarity between two concepts has a long history in the computational linguistics and artificial intelligence fields. Early works focused on similarities over taxonomies, often including the graph distance of concepts [50, 61, 105], and probabilistic notions of information content of classes in the taxonomy [50, 66, 88]. More recent works have looked at generalizing semantic similarity metrics from taxonomies to general ontologies (i.e., from trees to graphs) [27, 72, 90].

There are also traditional notions of set similarity which can be applied, since concepts in a knowledge base ultimately denote sets of entities. One commonly used metric is the Jaccard index which is defined below.

$$Sim_{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

One common trait to all of these similarity measures is that they are binary; they express similarity over pairs of concepts. We will see that efficiently accommodating n-ary similarities using preprocessed statistics poses challenges, though our model allows a natural approximation based on aggregates of binary metrics.

We now introduce the notion of knowledge base support, which characterizes the semantic coherence of an arbitrary concept. In the definition of knowledge base support, we appeal to n-ary similarity, with our approximation technique based on binary similarity to be introduced in Section 4.3.3.

Definition 13 (Knowledge Base Support) Let C be a general concept expressed using the entity types, relations, and entities in the knowledge base $\mathcal{K}$. Then the *support* of C by $\mathcal{K}$ is given by the following.

$$support(C, \mathcal{K}) = \begin{cases} 0 \text{ if } C \text{ is primitive,} \\ semanticSim(C_1, C_2, \cdots, C_n) + \\ \quad \sum_i support(C_i, \mathcal{K}) \qquad \text{otherwise.} \end{cases}$$

where each C_i is the i^{th} concept expression in a conjunction occurring in C. □

Intuitively, this is the similarity according to the edges in the disambiguation graph, which mirrors the structure of the query.

Figure 4.3 illustrates a disambiguation graph for a more complex query with multiple nesting branches and multiple levels of nesting. For this example, consider the following concept C as a possible disambiguated concept search query produced by mapping the query terms into the given knowledge base items.

$$\begin{array}{l} \texttt{SINGER} \sqcap \texttt{GUITARIST} \sqcap \exists \mathit{hasWonPrize}(\texttt{Grammy\_Award}) \\ \quad \sqcap \exists \mathit{livesIn}(\texttt{CITY} \sqcap \exists \mathit{locatedIn}(\texttt{England})) \end{array}$$

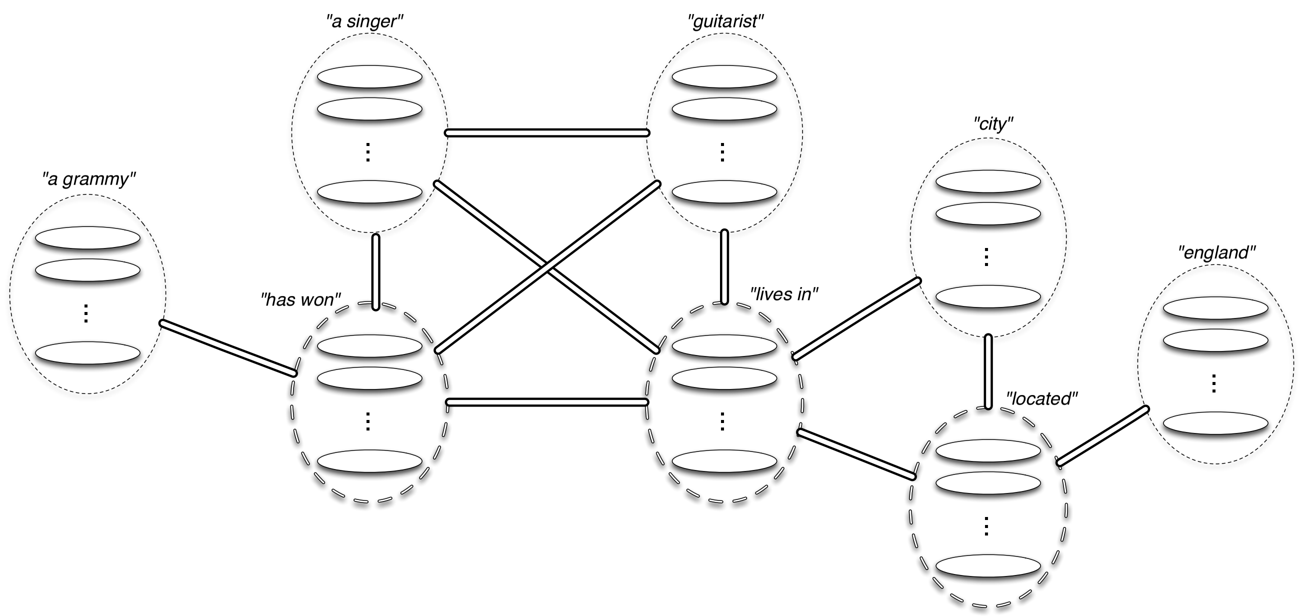


Figure 4.3: An example of a query disambiguation graph for the query ”a singer ⊓ guitarist ⊓ has won (a grammy)⊓ lives in (city⊓ located in (england))” involving multiple query branches and multiple levels of nesting. Each edge represents complete bipartite connectivity between all vertices in the groups.

Let C_1, C_2, and C_3 be defined as follows:

$$\begin{aligned} C_1 &= \exists \mathit{hasWonPrize}(\texttt{Grammy\_Award}) \\ C_2 &= \exists \mathit{livesIn}(\texttt{CITY} \sqcap C_3) \\ C_3 &= \exists \mathit{locatedIn}(\texttt{England}) \end{aligned}$$

Now C can be expressed as the following.

$$\texttt{SINGER} \sqcap \texttt{GUITARIST} \sqcap C_1 \sqcap C_2$$

The support of C corresponds to the following expression.

$$\begin{aligned} &\mathit{support}(C, \mathcal{K}) = \mathit{semanticSim}(\texttt{SINGER}, \texttt{GUITARIST}, C_1, C_2) \\ &+ \mathit{semanticSim}(\mathit{hasWonPrize}, \texttt{Grammy\_Award}) \\ &+ \mathit{semanticSim}(\mathit{livesIn}, \texttt{CITY}, C_3) \\ &+ \mathit{semanticSim}(\mathit{locatedIn}, \texttt{England}) \end{aligned}$$

Syntactic Similarity

The second component to our scoring model is the syntactic matching of the KB item label (or one of its synonyms) to the keyword. Indeed, the user entering queries has an idea of the entities they are trying to describe. It is simply the terminology of the user and the knowledge base that may differ, and the massive scale of the knowledge base impedes learning on the user's part. The closer we can match a query syntactically while retaining knowledge base support, the better our query interpretation will be with respect to both the user's intention and the knowledge encoded in our knowledge base. We use the function label $\mathit{syntaxSim}(a, b)$ to denote a measure of syntactic similarity. This could be simple keyword occurrence in the knowledge base item's label, or a more relaxed measure such as edit distance or q-gram similarity.

While syntactic similarity serves as a proxy for the intended meaning of individual query terms by a user, alternative signals of user intention could be integrated. For example, the personalized entity ranking approach presented in Chapter 5 could be used to prioritize knowledge base items of interest to a particular user. This would result in personalized query interpretations.

Score Aggregation

Our final scoring model is thus some combination of knowledge base support, which quantifies semantic coherence for the candidate disambiguated concept, and syntactic similarity, which reflects how closely the candidate concept interpretation matches the user's initial description of their query intention. We can tune how important each factor is by how we combine and weight the two similarity metrics. In general, we allow any aggregation function to be plugged into our system. However, as we will see in later sections, we can improve efficiency if the aggregation function is monotonic by making use of a rank-join algorithm for top-k search. We denote such an aggregation function as $\oplus$.

Definition 14 (Concept Score) Let SKQ be a structured keyword query, C a concept, $\oplus$ a binary aggregation function, and $\mathcal{K}$ a knowledge base. Then the *score* of concept C, with respect to SKQ, $\mathcal{K}$, and $\oplus$ is given by the following.

$$score(C, SKQ, \mathcal{K}) = support(C, \mathcal{K}) \oplus syntaxSim(C, SKQ)$$

□

Given a disambiguation graph G with vertex groups P and a parameter integer k, the goal of disambiguation is to find the top-k maximum scoring subgraphs of G that span all groups in P. Intuitively, we want to find the best k interpretations of the query in terms of some balance between knowledge base support and syntactic similarity. Each interpretation is formed by mapping each query term into an associated knowledge base item, and corresponds to a concept search query over the vocabulary of the underlying knowledge base.

The difficulty in solving the disambiguation problem lies in the nature of the scoring function. Looking back to our disambiguation graph model, the score of a general concept representing a candidate query interpretation depends on *all* components of the concept (all conjunctions of entity types and relations). The volume of statistics that must be (pre)computed in order to support queries with n terms would be infeasibly large to compute. One would need to have access to the semantic similarity of all n-way compositions of knowledge base items occurring in the knowledge base.

4.3.3 Approximating the Scoring Model

In most cases, having access to n-way semantic similarity would require pre-computing an infeasible number of statistics, or incurring the expensive cost of computing similarity at query time. Because the number of candidate query interpretations is exponential in the length of the query, computing similarity at query time could add unacceptable costs to performance. For example, Jaccard similarity requires the computation of intersections of (possibly very large) sets. Thus, we move to an approximation model for any binary metric of semantic similarity.

We approximate the score of a candidate query interpretation by aggregating the pairwise support of all components of the candidate concept. In terms of the disambiguation graph, this means that each edge weight in the graph can represent the support of having the two knowledge base items denoted by the edge's vertices in conjunction.

We extend a disambiguation graph G to include a weight function w that assigns weights to the vertices (v) and edges ($\langle v_1, v_2 \rangle$) of G as follows. The weights are computed with respect to some knowledge base $\mathcal{K}$ and the keywords k from a structured keyword query SKQ.

$$\begin{aligned} w(v) &= \mathit{syntaxSim}(\mathit{label}(v), k), \ \ \textit{where } v \in M(k) \\ w(\langle v_1, v_2 \rangle) &= \mathit{support}((\mathit{item}(v_1) \ \sqcap \ \mathit{item}(v_2)), \ \mathcal{K}) \end{aligned}$$

where $\mathit{item}(v)$ denotes the knowledge base represented by vertex v and $\mathit{label}(v)$ denotes the string representation of the KB item represented by v.

The approximate score of an induced subgraph denoting a candidate concept interpretation is given by the following.

Definition 15 (Approximate Score) Let SKQ be a structured keyword query, $\mathcal{K}$ a knowledge base, $\oplus$ a binary aggregation function, and $G = \langle V, E, w \rangle$ a subgraph of the disambiguation graph of SKQ representing a concept search query C. Then the *approximate score* of C represented by G with respect to SKQ, $\mathcal{K}$, and $\oplus$ is given by the following.

$$\widehat{score}(G, SKQ, \mathcal{K}) = \sum_{\langle v_1, v_2 \rangle \in E} w(\langle v_1, v_2 \rangle) \ \oplus \ \sum_{v \in V} w(v)$$

□

4.3.4 Computing Knowledge Base Mappings

The goal of the disambiguation problem is to find the top-k maximally scoring subgraphs (corresponding to concept interpretations of the original query) which are single connected components spanning all vertex groups. A subgraph corresponds to a mapping of each query term to a knowledge base item. Note that simply taking the maximum scoring subgraph for each subquery will not necessarily result in a connected graph, since one level of a query may bind a relation keyword to one relation, while the nested level of the query may bind the relation keyword to a different relation. For example, a the top level query may have a maximal scoring interpretation that binds *"has won"* to the relation *hasWeight*, while the nested query may be maximal in binding *"has won"* to *hasWonPrize*. Interpretations must therefore ensure equality on the vertex that relation keywords are bound to, and not only consider the disambiguation graph edges.

If our scoring function is monotonic, we can implement a rank-join algorithm [48] to efficiently find the top scoring subgraphs, so long as we have access to vertices and edges in sorted order. A global ordering over all edges can be precomputed, and vertices can be sorted on the fly since these sets fit in memory and can be kept relatively small in practice. Alternatively, we can implement a rank-join if we have access to only the vertices in sorted order and allow random access to edges.

With this view of the problem, our disambiguation graph is a join graph, with each keyword in the query providing an input to a rank-join operation based on the vertex group corresponding to the keyword. Each group is ordered by syntactic similarity which corresponds to vertex weight in the disambiguation graph. Consider each possible KB item match for a keyword phrase (the set $M(k)$) ordered by syntactic similarity as an input. We "join" two vertices if they have non-zero knowledge base support, and rank the join based on the value of the approximate score. Simultaneously, we join across nested subqueries based on equivalence of the vertex representing the relation keyword. This ensures the resulting joins form connected graphs in terms of the underlying disambiguation graph.

In practice we can prune zero-weight edges from the disambiguation graph since they do not contribute to semantically coherent interpretations and joins are only considered across non-zero weight edges. Figure 4.4 illustrates the example query disambiguation graph with the zero-weight edges pruned. As can be seen in the example, zero-weight edge pruning can significantly reduce the search space considered by the disambiguation algorithm.

The rank-join algorithm proceeds as follows. Each vertex group provides an input to the join. Each input is consumed one vertex at a time according to some policy for iterating over inputs. In our case we use simple round-robin iteration over each vertex group. When a vertex is consumed from an input list, we check (using random access to edges in the disambiguation graph) if it joins with any other previously consumed candidates. If it does a new (partial) mapping is created by adding the new vertex to the existing mapping. We also create a new mapping consisting of only the vertex. After each iteration, we can recompute the highest and lowest possible scores each mapping can have. The lowest score, or *lower bound*, corresponds to the current score of the mapping. This would be the final score of the mapping if no other vertices in the inputs join to the vertices currently in the mapping. The highest score, or *upper bound*, corresponds to the current score of the

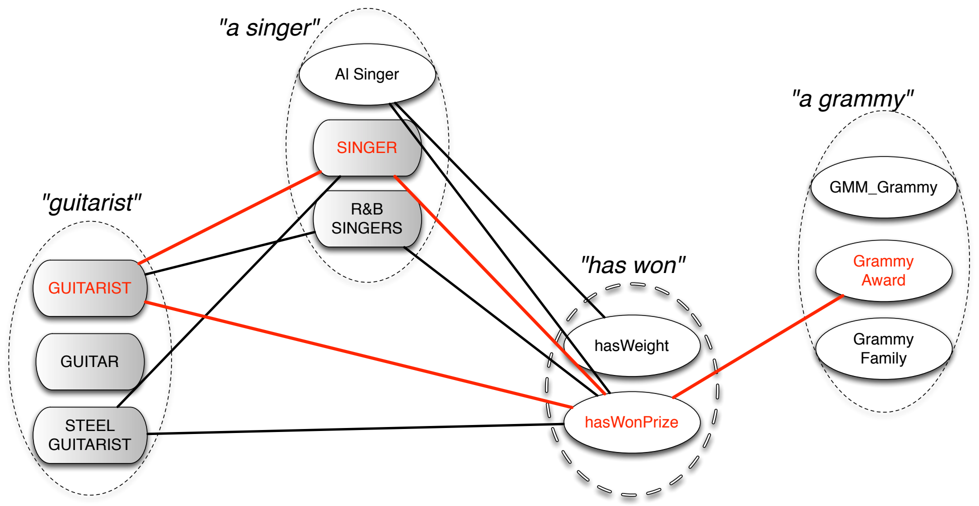


Figure 4.4: The example query disambiguation graph with zero-weight edges pruned. The correct mapping is highlighted in red.

mapping aggregated with the last seen score from each input that the mapping is currently missing (i.e., each vertex group for which the mapping has not yet bound a vertex to the query term). Because the inputs are sorted by score, we know that other vertices that remain in the input have equal or lower score than the last vertex consumed from the input. Assuming our score aggregation function is monotonic, we also know that the total score of the mapping will not decrease as more vertices are added to the mapping (bound to query terms). The computation of these bounds is what allows early termination of the algorithm during top-k query processing. As the algorithm iterates, we maintain a list of the current top-k candidates in addition to the pool of all other candidates. If the lower bound of the lowest scoring candidate in the top-k list is higher than the largest upper bound among all other candidates the algorithm can terminate. This is because the highest possible score that the best candidate can have will not be high enough to qualify it as a top-k result. Similarly, if the upper bound of any other candidate is lower than the lower bound of the lowest scoring top-k mapping, we can remove that candidate from the candidate pool. This is because, independent of what other vertices join to the candidate, it cannot possibly have a score high enough to qualify it as a top-k scoring mapping since we already have k mappings with a higher score. The early termination based on scoring bounds is the fundamental principle behind threshold algorithms. A more detailed overview of the rank-join algorithm in particular can be found in Ilyas et. al. [48].

As an example, consider the disambiguation graph from Figure 4.2, we illustrate a rank-join approach to disambiguation of the query in Figure 4.5. The illustration shows each vertex group as an input to the rank join. Conjunctive components are joined based on the existence of non-zero weight edges in the disambiguation graph. The actual weight of the edge contributes to the final score of the concept search query produced by the mapping. Nested sub-queries are joined to the outer query based on binding of the relation keyword to the same KB relation by the outer query and sub-query.

When interpreting keyword queries we allow relations to be unspecified, however we do not consider mappings in which entities or types are unspecified. This is denoted by replacing k with the symbol "$*$" in the relation construct and represents an arbitrary, unspecified relation with no syntactic constraints. The "$*$" thus matches any relation in the knowledge base, as seen in the example query *"songs"* $\sqcap$ $*$(*"jimi hendrix"*) illustrated in Figure 2.1.

While solving the disambiguation problem involves possibly exploring an exponential search space in the worst case, our experiments show that we can still find meaningful solutions quickly in practice (see Section 4.7).

4.4 Concept Search Queries as Keyword Query Interpretations

4.4.1 Admissibility of Concept Search Queries as Interpretations

Applying the techniques outlined in Chapters 3 and 4 to keyword queries produces admissible concept search queries as defined in Definition 6. Keyword query interpretations are ultimately composed of the nodes of a disambiguation graph. Disambiguation graph nodes are constructed from the sets $M(k)$ where k is a segmented keyword expression from a keyword query Q. The set $M(k)$ is defined as a set of knowledge base items. Thus, produced concept search queries are safe and the first property is satisfied by definition.

Since query interpretations map each segmented query phrase to an item in $M(k)$, the second property is also satisfied at the level of each query phrase $\langle q_1 q_2 \ldots q_n \rangle \in Q$. At the level of individual query terms however, the strict definition of the second property is not satisfied. We adopt a relaxed version of the constraint for keyword query coverage which allows partial and fuzzy matching of segmented keyword phrases to knowledge base items in order to satisfy this property.

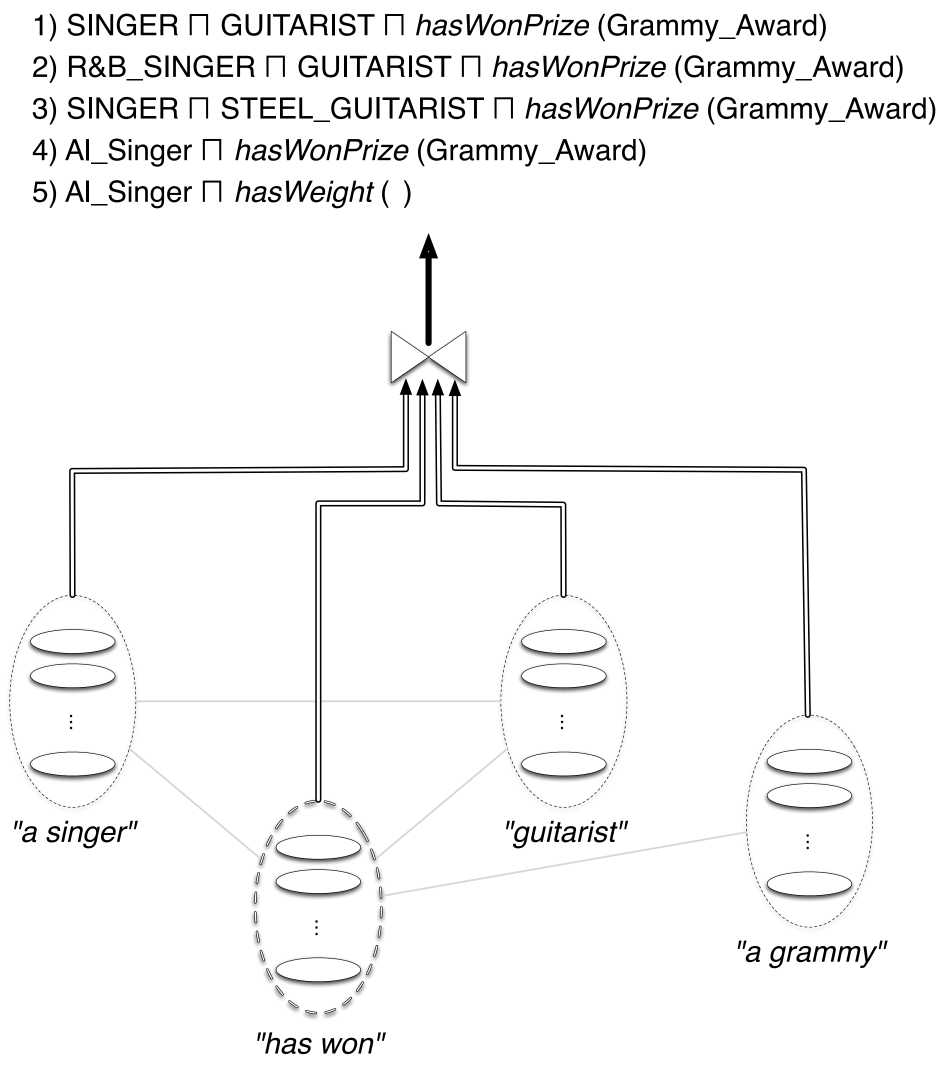

1) SINGER ⊓ GUITARIST ⊓ *hasWonPrize* (Grammy_Award)
2) R&B_SINGER ⊓ GUITARIST ⊓ *hasWonPrize* (Grammy_Award)
3) SINGER ⊓ STEEL_GUITARIST ⊓ *hasWonPrize* (Grammy_Award)
4) Al_Singer ⊓ *hasWonPrize* (Grammy_Award)
5) Al_Singer ⊓ *hasWeight* ( )
"a singer"
"guitarist"
"a grammy"
"has won"

Figure 4.5: Encoding disambiguation as a rank-join with random access to edges. Each vertex group forms an ordered input sequence to the join operator. The join conditions are on the existence of (non-zero) edges in the disambiguation graph, and on the equality of relation bindings to ensure sub-queries are connected.

The third property is satisfied by the procedure described in Chapter 3 executing each candidate concept search query, retaining only those with a non-empty result.

4.4.2 Scoring Keyword Query Interpretations

Among the space of admissible concept search queries, we want to compute the most probable concept search query C as an interpretation for the given keyword query Q over the knowledge base $\mathcal{K}$, as specified in the query understanding problem definition (Definition 7). Recall Equation 2.6 from the definition.

$$C = \text{argmax}_{C'}\ \Pr(C'|Q, \mathcal{K})$$

While directly estimating this distribution is not possible in our situation (we do not have enough examples of keyword queries mapping to concept search queries in order to directly estimate the distribution), we can view the heuristic scoring functions presented in Chapters 3 and 4 as an intuitive approximation of this quantity. We start by introducing structured keyword queries as an intermediary between keyword queries and concept search queries. We do this by expressing $\Pr(C|Q, \mathcal{K})$ (the conditional distribution over concept search queries C given a keyword query Q and knowledge base $\mathcal{K}$) as the joint distribution of concept search queries and structured keyword queries given a keyword query, with the structured keyword query SKQ marginalized out.

$$= \text{argmax}_{C'}\ \textstyle\sum_{SKQ} \Pr(C', SKQ|Q, \mathcal{K})$$

Applying the law of total probability we can rewrite the above equation as follows.

$$= \text{argmax}_{C'}\ \textstyle\sum_{SKQ} \Pr(C'|SKQ, Q, \mathcal{K}) \cdot \Pr(SKQ|Q, \mathcal{K})$$

We assume a concept search query to be conditionally independent of a keyword query given a structured keyword query. This allows us to drop the dependence on Q in the first term on the assumption that the structured keyword query faithfully represents the intention of the keyword query Q. (Such assumptions are common practice, see for example [58]).

$$= \text{argmax}_{C'}\ \textstyle\sum_{SKQ} \Pr(C'|SKQ, \mathcal{K}) \cdot \Pr(SKQ|Q, \mathcal{K})$$

The mapping of queries to structured keyword queries is independent of any particular knowledge base, thus we can drop the dependance on $\mathcal{K}$ in the second term which yields the following.

$$= \text{argmax}_{C'}\ \textstyle\sum_{SKQ} \Pr(C'|SKQ, \mathcal{K}) \cdot \Pr(SKQ|Q)$$

We assume a structured keyword query to be conditionally independent of a keyword query given an annotated query. This is intuitive since the annotated query captures the keyword query in its entirety. The quantity $\Pr(SKQ|Q)$ can then be written as $\sum_{AQ} \Pr(SKQ|AQ) \cdot \Pr(AQ|Q)$ which yields the following equation.

$$= \text{argmax}_{C'} \sum_{SKQ} \Pr(C'|SKQ, \mathcal{K}) \cdot \sum_{AQ} \Pr(SKQ|AQ) \cdot \Pr(AQ|Q)$$

We use the methods based on query log analysis presented in Sections 3.1 and 3.2 to estimate the latter two quantities directly where $\Pr(SKQ|AQ) = \Pr(T|AQ)$ with T being the structured query template for SKQ. In practice we cannot directly estimate the quantity $\Pr(C'|SKQ, \mathcal{K})$. We simply do not have a sufficiently large amount of labelled training data to estimate such a large distribution. We resort to scoring concept search queries using a heuristic scoring function. We use the scoring function $\mathit{score}(C, SKQ, \mathcal{K})$ presented in Section 4.3 as a heuristic to quantify the relevance of a concept search query given a structured keyword query and knowledge base. Our final computation of a concept search query C given a keyword query Q and knowledge base $\mathcal{K}$ is then given by the following heuristic scoring function.

$$C \approx \text{argmax}_{C'} \sum_{SKQ} \mathit{score}(C', SKQ, \mathcal{K}) \cdot \sum_{AQ} \Pr(SKQ|AQ) \cdot \Pr(AQ|Q)$$

While *score* is not a proper estimation of $\Pr(C'|SKQ, \mathcal{K})$, we rely on empirical evidence that it effectively quantifies the quality of a concept search query given a structured keyword query and knowledge base (see Section 4.7). Beacause the scoring function is ultimately used to determine the relative order of concept search queries as interpretations of a given keyword query, the absolute value of the function is not important. We are only interested in the relative order of values over candidate query interpretations.

4.5 Computing All-Pairs Similarity over Knowledge Base Items

A key computational challenge to supporting the disambiguation model described in this chapter is in pre-computing semantic similarities over all pairs of knowledge base items in order to support the efficient calculation of knowledge base support described in Definition 13. One approach is to actually enumerate and compute the similarity of all pairs. This naïve approach is prohibitively expensive, we estimate it would have taken roughly six months of compute time on our server to compute all pairs of similarities for YAGO.

However, by exploiting a particular property of our chosen similarity measure, a much more efficient formulation of the computation can be achieved.

Set-based similarity metrics that are based on intersections can benefit from a more efficient all-pairs similarity algorithm if they have the property that a zero sized intersection implies a similarity of zero. The algorithm gains efficiency in only considering pairs that have non-zero similarities. If we conceptually think of the all-pairs similarity space as a matrix with all items along the row and column, and each cell representing the similarity of the corresponding row/column item, then we expect this matrix to be very sparse for real world knowledge bases. Most items are not similar to each other (their entity sets have an empty intersection). Any algorithm that explicitly considers all pairs is doomed to waste time computing zero entries in such a matrix.

Our approach to all-pairs similarity for the Jaccard similarity measure works from the bottom up, rather than the top down approach of considering all pairs. We iterate over all entities, for each entity we enumerate all of its types, outgoing relations, incoming relations (as inverses), and a singleton set of the entity itself. We then output all pairs from this enumeration, as this entity will contribute a count of one to the size of the intersection of each of these output pairs. A large file of pairs is generated after a linear database scan is completed, we then sort this file so that the same pairs (generated by different entities) will be sequential in the file. We then perform a linear scan over the file, counting the number of times each pair occurs. This count is equal to the size of the intersection of the two items constituting the pair. Since we also know the size of the entity set denoted by each item, we can compute the Jaccard measure as follows.

$$\begin{aligned} Sim_{Jaccard}(A,B) &= \frac{|A \cap B|}{(|A|+|B|)-|A \cap B|} \\ &= \frac{|A \cap B|}{|A \cup B|} \end{aligned}$$

This bottom up approach runs for our complete knowledge base in roughly one week, an estimated ~26x reduction in run-time as compared to the naïve approach.

4.6 Knowledge Graph Management

4.6.1 Knowledge Graph Indexing

We index the knowledge graph using an adjacency list representation. For each node in the graph (entity, entity type, constant), we create an index entry. For a given node, an index entry consists of all incoming and outgoing edges in the graph, grouped by edge label. We store edge labels ordered as a binary search tree. For each edge label, all adjacent nodes are stored in an ordered set (represented physically as an array) ordered by a global ordering (e.g., lexicographic ordering or a static global scoring function). Storing nodes using a static global ordering will aid in efficiently computing intersections and unions of node lists. An example index is illustrated in Figure 4.6.

An advantage to this representation of the knowledge graph is that it can be easily implemented using a key/value store. With the recent explosion in interest around key/value stores (e.g., [22, 60, 115]), our approach can directly benefit from the properties provided by these systems. In particular, this indexing approach can be easily distributed, scaled, and provide fault tolerance by building on existing key/value store technology.

A knowledge graph index is accessed by a single interface function $index(\mathcal{K}, node, edge)$ where $\mathcal{K}$ is a knowledge graph, *node* is a valid entity, entity type, or constant appearing in $\mathcal{K}$, and *edge* is a valid relation, attribute, or one the special labels *type* or *subClassOf* which represent entity type assertions and inclusion dependencies respectively. We also use the function $closure(\mathcal{K}, A)$ to denote the set of entity types in the subclass hierarchy below the entity type A. The closure function can be computed by recursively following the *subClassOf* relation using the index function. We now describe some desirable properties of a knowledge graph index that will be useful in proving the correctness of a concept search evaluation procedure.

Propostition 1 (Knowledge Graph Index Construction) Given a knowledge graph $\mathcal{K} = \langle \mathcal{O}, \mathcal{A} \rangle$, the following properties hold by construction.

$$A(e) \in \mathcal{A} \quad \textit{iff} \quad e \in index(\mathcal{K}, A, type^-) \tag{4.1}$$

$$R(e_1, e_2) \in \mathcal{A} \quad \textit{iff} \quad e_2 \in index(\mathcal{K}, e_1, R) \tag{4.2}$$

$$R(e_1, e_2) \in \mathcal{A} \quad \textit{iff} \quad e_1 \in index(\mathcal{K}, e_2, R^-) \tag{4.3}$$

$$f(e_1, k) \in \mathcal{A} \quad \textit{iff} \quad k \in index(\mathcal{K}, e_1, f) \tag{4.4}$$

$$f(e_1, k) \in \mathcal{A} \quad \textit{iff} \quad e_1 \in index(\mathcal{K}, k, f^-) \tag{4.5}$$

$$A_1 \sqsubseteq A_2 \in \mathcal{O} \quad \textit{iff} \quad A_1 \in index(\mathcal{K}, A_2, subClassOf^-) \tag{4.6}$$

$$A_1 \sqsubseteq A_2 \ldots A_n \sqsubseteq A \subseteq \mathcal{O} \quad \textit{iff} \quad \{A_1, A_2, \ldots, A_n\} \subseteq closure(\mathcal{K}, A) \tag{4.7}$$

□

PERSON	:	{*subClassOf*$^{-}$⟨MUSICIAN, NON-VIOLENCE_ADVOCATE⟩}
MUSICIAN	:	{*subclassOf*$^{-}${GUITARIST⟩, *subClassOf*⟨PERSON⟩}
GUITARIST	:	{*type*$^{-}$⟨John_Lennon, Jimi_Hendrix⟩, *subClassOf*⟨MUSICIAN⟩}
NONVIOLENCE_ADVOCATE	:	{*type*$^{-}$⟨John_Lennon⟩, *subClassOf*⟨PERSON⟩}
AWARD	:	{*subclassOf*$^{-}$⟨GRAMMY_AWARD⟩}
GRAMMY_AWARD	:	{*type*$^{-}$⟨Grammy_Lifetime_Achievement_Award⟩, *subclassOf*⟨AWARD⟩}
CREATIVE_WORK	:	{*subClassOf*$^{-}$⟨SONG⟩}
SONG	:	{*type*$^{-}$⟨Castles_Made_of_Sand, Little_Wing⟩,
	:	*subclassOf*$^{-}$⟨ANTI-WAR_SONG⟩, *subclassOf*⟨CREATIVE_WORK⟩}
ANTI-WAR_SONG	:	{*type*$^{-}$⟨Happy_Xmas_(War_Is_Over)⟩, *subclassOf*⟨SONG⟩}
LOCATION	:	{*subClassOf*$^{-}$⟨CITY⟩}
CITY	:	{*subclassOf*$^{-}$⟨COASTAL_CITY⟩, *subclassOf*⟨LOCATION⟩}
COASTAL_CITY	:	{*type*$^{-}$⟨Athens, Liverpool}⟩, *subclassOf*⟨CITY⟩}
Grammy_Lifetime_Achievement_Award	:	{*hasWonPrize*$^{-}$⟨Jimi_Hendrix, John_Lennon⟩, *type*⟨GRAMMY_AWARD⟩}
Happy_Xmas_(War_Is_Over)	:	{*created*$^{-}$⟨John_Lennon⟩, *type*⟨ANTI-WAR_SONG⟩}
Little_Wing	:	{*created*$^{-}$⟨Jimi_Hendrix⟩, *type*⟨SONG⟩}
Castles_Made_of_Sand	:	{*created*$^{-}$⟨Jimi_Hendrix⟩, *type*⟨SONG⟩}
"1940-10-09"	:	{*dateOfBirth*$^{-}$⟨John_Lennon⟩}
Liverpool	:	{*birthplace*$^{-}$⟨*John_Lennon*⟩, *type*⟨COASTAL_CITY⟩}
Athens	:	{*type*⟨COASTAL_CITY⟩}
Jimi_Hendrix	:	{*created*⟨Castles_Made_of_Sand, Little_Wing⟩,
	:	*hasWonPrize*⟨Grammy_Lifetime_Achievement_Award⟩, *type*⟨GUITARIST⟩}
John_Lennon	:	{*birthplace*⟨Liverpool⟩, *created*⟨Happy_Xmas_(War_Is_Over)⟩,
	:	*hasWonPrize*⟨Grammy_Lifetime_Achievement_Award⟩, *dateOfBirth*⟨*"1940-10-09"*⟩,
	:	*type*⟨GUITARIST, NON-VIOLENCE_ADVOCATE⟩}

Figure 4.6: A knowledge graph index for the example knowledge graph.

```
eval($C$, $\mathcal{K}$, $index$):
    case $C$ = "$\{e\}$":
        return $\{e\}$

    case $C$ = "$A$":
        result = $index(\mathcal{K}, A, type^{-})$
        for each $A'$ in $index(\mathcal{K}, A, subClassOf^{-})$:
            result = result $\cup$ eval($A'$, $\mathcal{K}$, $index$)
        return result

    case $C$ = "$C_1 \sqcap C_2$":
        return eval($C_1$, $\mathcal{K}$, $index$) $\cap$ eval($C_2$, $\mathcal{K}$, $index$)

    case C = "$\exists R(C_1)$":
        result = $\emptyset$
        for each $e$ in eval($C_1$, $\mathcal{K}$, $index$):
            result = result $\cup$ $index(\mathcal{K}$, e, $R^{-})$
        return result

    case C = "$\exists R^{-}(C_1)$":
        result = $\emptyset$
        for each $e$ in eval($C_1$, $\mathcal{K}$, $index$):
            result = result $\cup$ $index(\mathcal{K}$, e, $R)$
        return result

    case C = "$f(k)$":
        return $index(\mathcal{K}, k, f^{-})$

    case C = "$f^{-}(C_1)$":
        result = $\emptyset$
        for each $e$ in eval($C_1$, $\mathcal{K}$, $index$):
            result = result $\cup$ $index(\mathcal{K}$, e, $f)$
        return result
```

Figure 4.7: The knowledge graph concept search query evaluation procedure.

4.6.2 Concept Search Query Evaluation

Definition 16 (Concept Search Query Evaluation) Given a knowledge graph $\mathcal{K}$, a concept search query C, and an knowledge graph index *index*, we write $eval(C, \mathcal{K}, index)$ to denote the result of the procedure presented in Figure 4.7.

□

Figure 4.7 details the query evaluation algorithm for query processing over the knowledge graph index. The evaluation mirrors the concept grammar (Definition 1) used to define concept search queries. Subclass hierarchies over entity types are recursively traversed as part of query evaluation. The benefit of this approach is that knowledge graph constraints can be efficiently inserted or deleted without affecting many index entries. If type hierarchies were "flattened" (ie., all implied assertions were explicitly materialized) then updates could potentially be very expensive.

As an example of query processing, consider a search for the following concept search query: `MUSICIAN` $\sqcap$ *birthplace*(`COASTAL_CITY`). We can find all entities explicitly asserted to have type `MUSICIAN` by retrieving the corresponding index entry, and extracting the nodes (denoting entities) in the $type^{-}$ relation (i.e., the nodes denoting entities that have an outgoing *type* edge to the `MUSICIAN` node). In our example index, there are no entities explicitly asserted to have the type `MUSICIAN`. Then we recursively repeat this process for all nodes in the $subclassOf^{-}$ relation, computing an ordered union (i.e., merge operation) for all entities found. In the example, this produces the ordered set ⟨`Jimi_Hendrix, John_Lennon`⟩. Next (or in parallel in a distributed scenario), we evaluate the relation part of the query: *birthplace*(`COASTAL_CITY`). We evaluate the subquery `COASTAL_CITY` producing the set ⟨`Liverpool`⟩, and extract all nodes for the $birthplace^{-}$ relation for each result. In the example, this corresponds to the set ⟨`John_Lennon`⟩ The last step of query evaluation is to compute the intersection of the two subqueries. Note that the global entity order can be preserved by using a merge style intersection algorithm since the input entity lists stored in the index are sorted. For our example, this produces the final result ⟨`John_Lennon`⟩. We use a galloping search implementation of set intersection [9] based on the analysis done in Barbay et al [6]. An interesting observation with regards to this intersection approach is that the behaviour can resemble a nested loop join, where items from one side of the intersection are searched for in the other side. This gives significant performance advantages over a linear intersection algorithm. For our example, the intersection would consist of a single logarithmic search for `John_Lennon` in the intersecting set.

We now show the correctness of the evaluation procedure defined in Definition 16 with respect to the concept search query semantics defined in Definition 4.

Theorem 1 Given a knowledge graph $\mathcal{K} = \langle \mathcal{O}, \mathcal{A} \rangle$, a concept search query C, and the knowledge graph index function $index$:

$$eval(C, \mathcal{K}, index) = answer(C, \mathcal{K})$$

Proof: (See Appendix B.2.) □

4.7 Evaluation of Graph Queries with Keyword Predicates

Since computing mappings from candidate structures to knowledge base concept interpretations in a key part of our system, we isolate that component and explore its capabilities for a series of more complex structures than those produced by keyword queries. These longer, more complex queries represent information needs that could be issued by technically inclined expert users capable of formulating their queries directly as structured keyword queries.

We have designed a number of tasks to evaluate the effectiveness and efficiency of our KB mapping procedure. We directly evaluate the concept search queries produced by our system against the ideal structured concept query which is constructed manually. We also use an entity search task to evaluate the precision and recall of entities returned by our system as the number of computed query interpretations is varied.

We compare our system to two baseline approaches, one using syntax-only mappings, and one being an adaptation of a traditional IR engine for the entity search task. We use a manually defined concept search query as the definition of a correct answer in all cases.

4.7.1 System Descriptions

QUICK is built on a variety of open-source software. We create a Lucene [117] text index over all KB items and any synonyms encoded in YAGO. We use this index to generate the vertex groups of candidate items (the sets $M(k)$) based on keyword occurrence. The knowledge base engine for concept query processing is an implementation of the architecture described in Section 4.6 and was written in Java. The system uses Berkeley DB [111] as a back-end data store for index entries. We also use a Berkeley DB instance to maintain the knowledge base semantic similarity measures. The pair-wise semantic similarity values used for disambiguation are pre-computed as described in Section 4.5. We experiment with three types of syntactic similarity measures. The first is keyword occurrence, where the syntactic similarity is the number of query keywords occurring in the label of the schema item. While this measure is simplistic, it is very efficient since it is computed as part of generating the candidate sets. The second measure is Levenshtein edit-distance. The last measure is q-gram similarity for varying values of q. We use the Jaccard measure for semantic similarity.

IR is an information retrieval based system that uses only the keywords in the query for search, and is thus unable to exploit the structure encoded in the query. This system retrieves entities for the entity search task (see Section 4.7.3) by searching their Wikipedia text. We make use of a commercial Web search engine for document retrieval over Wikipedia by building on Yahoo BOSS[1].

Syntax-Only is a simple disambiguation approach that uses only syntactic similarity to perform mappings. The closest syntactic match occurring in the structured data source for each relation and entity/concept are chosen for disambiguation to a formal query, and semantic similarities are unused.

4.7.2 Data and Workload

Data

We start with the YAGO knowledge base described in Section 1.2.5. In order to measure the effectiveness of our system in various situations, we also created an extended version of YAGO. In the extended version, any information used in any of the benchmark queries that was missing in the original YAGO is added. This allows us to evaluate knowledge base mappings in the case where the knowledge base is incomplete as well as the case when full disambiguations are possible. The extended information is encoded using the existing entities and relations in YAGO where possible, or by using the Wikipedia identifier of the entities if they do not exist in YAGO. We introduce a new relation when needed using the terms from the benchmark. This means that the extended YAGO contains all of the information used by the queries, but the specific keywords used in the queries may not necessarily correspond to the labels in YAGO.

TREC Target 216: *"Paul Krugman"*		
Question / Answers	For which newspaper does Krugman write? At which university does Krugman teach? From which university did he receive his doctorate? What prize originating in Spain has Krugman won? ...	new york times princeton university MIT john bates clark medal
Structured Keyword Query	*"writes for (new york times)* ⊓ *teaches at (princeton university)* ⊓ *received doctorate (MIT)* ⊓ *won prize(john bates clark medal)"*	
Concept Query	*worksAt* (`The_New_York_Times`) ⊓ *worksAt*(`Princeton_University`) ⊓ *graduatedFrom* (`Massachusetts_Institute_of_Technology`) ⊓ *hasWonPrize*(`John_Bates_Clark_Medal`)	

Figure 4.8: An example encoding of a TREC QA task item to a structured keyword query.

Effectiveness Workload

For the quality evaluation, we wanted a set of information needs which, when expressed as structured keyword queries, form queries that extend beyond the relatively simple structures extracted from keyword queries. It was important when designing this benchmark that the keywords in the queries were chosen without bias, as this could greatly simplify disambiguation. We constructed a benchmark from the TREC 2007 Question Answering task [29]. While the individual TREC questions where too simple to form challenging disambiguation problems (i.e., they are too short), we were able to use the benchmark to design more complex queries in the following way. Each TREC task consists of a target entity and a set of questions and answers about the target. We inverted the benchmark, taking question and answer pairs in conjunction to form a query which has the target entity as the desired answer. Figure 4.8 shows an example of how this translation was done. In all cases, we use only keywords appearing in the TREC benchmark (or morphological variants to preserve the semantics of the questions) to ensure there is no bias in how keywords are chosen. We then structured the queries to represent the query intent. This coincides with the intent of structured keyword queries as an end-user query language, in which users are able to express structure as desired, while the specific keywords needed to describe concepts, entities, and relations are unknown. We encoded a total of 22 queries which can be found in Appendix A.2. Half of the queries can be interpreted over the raw YAGO KB, while half need the extended KB for interpretations.

Efficiency Workload

Because our TREC derived workload produces a set of queries with very similar shape and size, we designed a synthetic workload to exercise a wider range of query shapes and sizes. These queries vary in length from one to eight terms, and vary in generality of the terms. We characterize the overall generality of a query by the total number of entities that are processed at intermediate stages in order for our system to evaluate the query. Our workload varies in generality from 1 to 100,000 intermediate entity results. Queries also vary in their shape including flat queries (conjunctions with no relations), chain queries (a nesting of relations), and star queries (a conjunction of relations) of each length and generality where possible (it is not always possible to create one or two term queries with high generality). This workload consists of 62 queries and can be found in Appendix A.3.

[1]http://developer.yahoo.com/search/boss/

	Mapped (+)	Empty (−)	Total (true/false)
Correct (true)	%45.5	%50.0	%95.5
Incorrect (false)	%4.5	%0.0	%4.5
Total (+/−)	%50.0	%50.0	

(a) Raw KB

	Mapped (+)	Empty (-)	Total (true/false)
Correct (true)	%90.9	%00.0	%90.9
Incorrect (false)	%9.1	%0.0	%09.1
Total (+/−)	%100.0	%0.0	

(b) Extended KB

Figure 4.9: Confusion matrices for mappings over the Raw and Extended knowledge bases. Proportion of queries that get disambiguated (Mapped) or do not get disambiguated (Empty) vs. whether the disambiguation is correct.

4.7.3 Experimental Results

Disambiguation Task

In the disambiguation task, we isolate the disambiguation algorithm from the rest of the system and explore its behaviour on our effectiveness workload for both the raw and extended versions of the YAGO KB.

We consider the following question for each query. Was the computed (possibly partial) disambiguation correct, or if no disambiguation is possible due to KB incompleteness, was no disambiguation computed? This property allows us to see the proportion of queries for which we obtain a correct formalization of some part of the information need. We consider a “failed” (or “empty”) disambiguation to be the correct behaviour in the case where no disambiguation is possible due to a lack of coverage in the underlying KB. This corresponds to the true-positives, true-negatives, false-positives, and false-negatives respectively.

Figure 4.9 shows the confusion matrices after running our disambiguation algorithm with the raw (incomplete) and extended KB. This demonstrates disambiguation behaviour in the case of an incomplete KB and a complete KB. We take the top ranking knowledge base mapping using 3-gram syntactic similarity and Jaccard similarity for semantic similarity.

The results demonstrate that even in the presence of incomplete knowledge (Figure 4.9

table (a)), our system can still compute a correct partial mapping around 95% of the time. Interestingly, our approach proves to be very accurate in determining when a mapping does not exist, and returns no mapping rather than an incorrect one. It is this property that allows us to accurately determine when keyword queries cannot be interpreted as investigated in Section 3.4. This is particularly promising since in the scenario of applying this technology to a general search system, this situation corresponds to falling back to regular keyword search. Correctly identifying when to fall back to keyword search means we do not process incorrect concept search queries which would produce irrelevant results. When a complete KB is introduced (table (b)), the task becomes more difficult since there always exists a correct disambiguation for every query. Our algorithm sees a slight drop in overall correct mappings, but still stays around 90% correct.

Entity Search Task

In the entity search task, we used our effectiveness workload to evaluate the accuracy of retrieving entities described by the queries. The results are compared against a manually created concept search query which returns correct results by definition. We follow the evaluation methodology outlined in Chapter 6 with correct results having a relevance of 1.0 and all other results 0.0. The evaluation in the raw KB scenario is not well defined for this task, since partial interpretations may describe a broader class of entities than the actual query target. How to use these partial interpretations is system dependant and more of a user interface design issue than a search relevance issue. We therefore consider only the extended KB for the entity search task, meaning complete interpretations are always possible.

Figure 4.10 shows the average precision, recall, and F_1-score for candidate sets ($M(k)$) of size 30 as a function of k, the number of knowledge base mappings considered by the *QUICK* systems. In all cases, we take the union of the results of the top-k disambiguated concept search queries. The three variants of *QUICK* correspond to the configurations of syntactic similarity (keyword occurrence (KW), edit distance (ED), and q-gram similarity with $q = 3$ (QD3)). We see for *QUICK*-ED and *QUICK*-QD3 that precision degrades as k increases. This is expected as more results are added from different interpretations, causing more irrelevant entities in the result. *QUICK*-KW on the other hand, generally does not find relevant entities until $k = 5$, causing the spike in precision. In the recall graph, we see that *QUICK*-QD3 generally finds all of the relevant entities it is capable of finding within the top ($k = 1$) mapping, while *QUICK*-ED peaks at $k = 3$ and *QUICK*-KW at $k = 5$. Overall, we get maximum precision and F_1-score using q-gram similarity at $k = 1$, while recall is best with edit distance at $k = 3$. We also varied the size of candidate sets

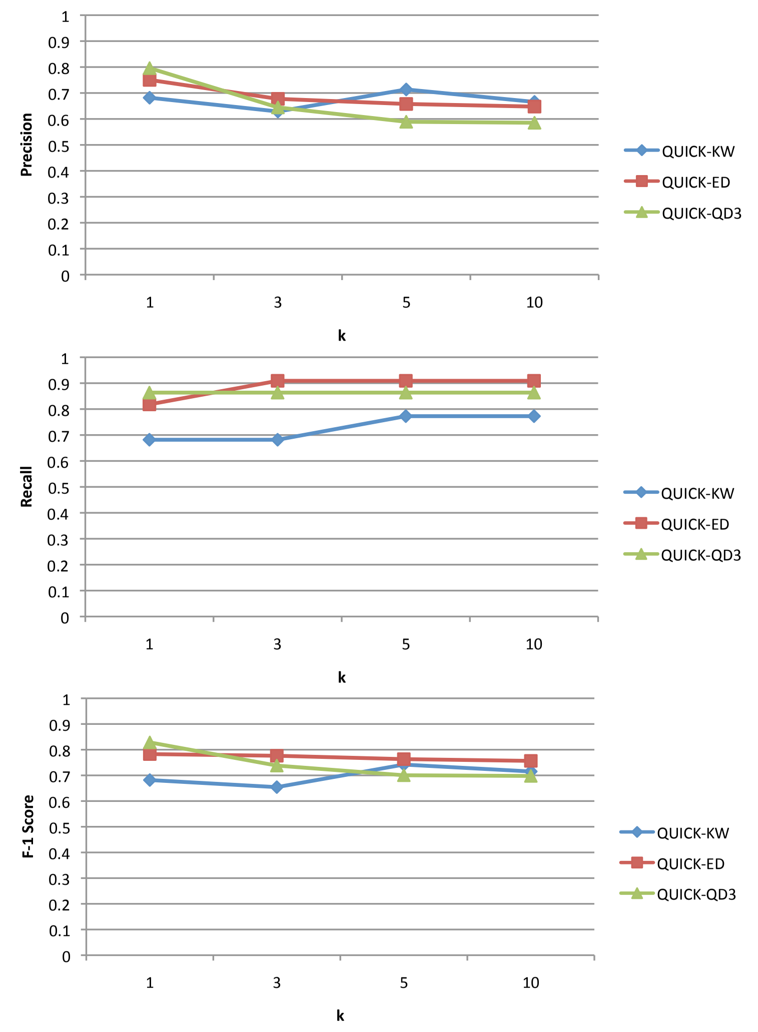


Figure 4.10: Average precision (a), recall (b), and F_1-score (c) vs. the number of knowledge base mappings (k) considered for the entity retrieval task.

from 10 to 100, but found this had little to no impact on result quality. This means that the correct schema items generally appear near the top of the candidate list, or not at all within the first 100.

Figure 4.11 shows the results for all systems, including three variations of *QUICK* with fixed k values. The IR and Syntax-Only disambiguation approaches do similarly, scoring around 0.1 for precision, recall, and F_1-score. All three variations of *QUICK* produce substantially better results, even when considering the poorest performing configurations which can be seen in Figure 4.10. We also varied the value of q from 1 to 5 for the q-gram approach, and found no difference in quality. The simple position invariant property of q-gram similarity seems to be what contributes to its high quality.

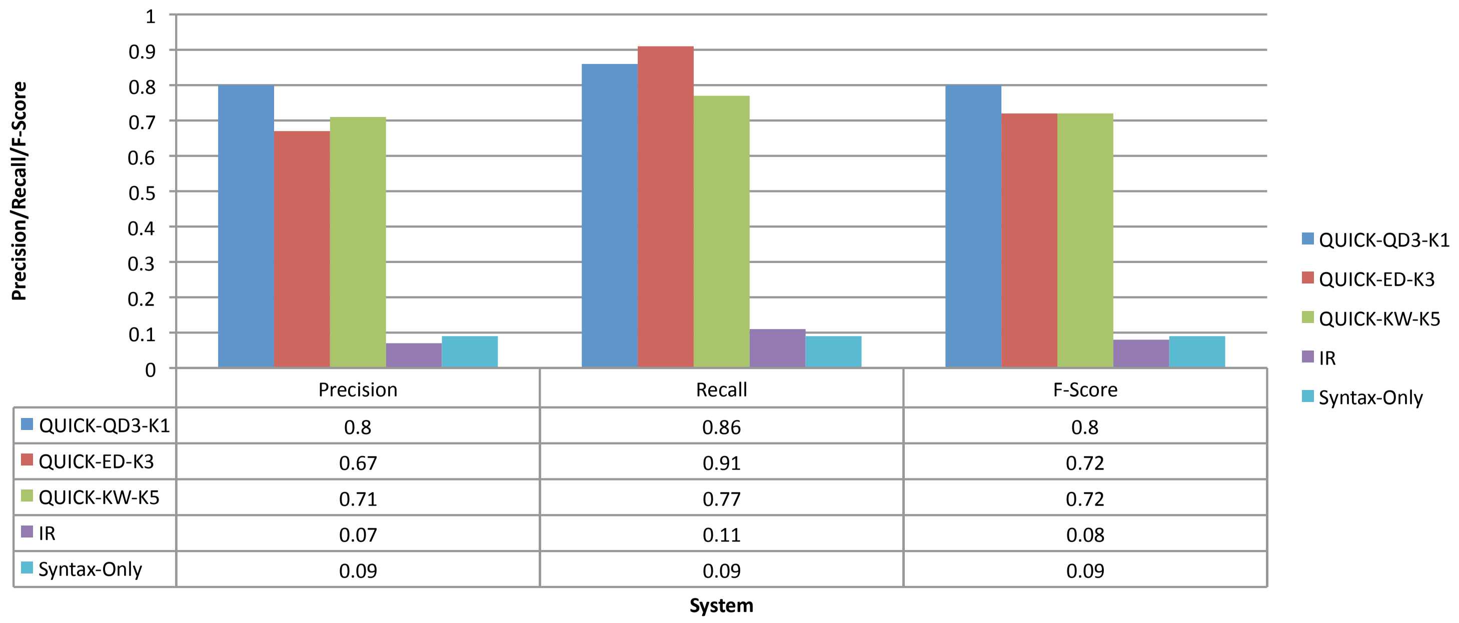
We find that all of the configurations of *QUICK* greatly outperform the simple IR and Syntax-Only baselines. This is because *QUICK* is able to exploit the structure expressed in the query, while the IR system is not designed to incorporate this form of structure. The poor performance of the Syntax-Only system emphasizes the need for knowledge base mappings that consider the semantic coherence of the mapping of the query into the underlying structured data. Simply taking the best syntactic match for each keyword does not produce good interpretations.

System Performance

The full system performance on our TREC derived workload averaged 0.32 seconds (0.06 seconds with the omission of one long running outlier) due to the relative simplicity in the structure and generality of the queries. The synthetic workload on the other hand exercises a much broader class of query shapes and generalities. Figure 4.12 shows the breakdown of performance over the synthetic workload for both phases of query processing (query disambiguation and KB search). KB search is generally the most expensive part of the process. In general the disambiguation component accounts for only a small fraction of the total run-time, however for some queries produce highly connected disambiguation graphs causing disambiguation to dominate run-time.

4.8 Related Work

The problem of mapping keywords into structured knowledge graphs has been studied in the context of keyword search over relational databases. In this problem, candidate network graphs are generated based on the location of keyword matches [45]. The challenge in this



	Precision	Recall	F-Score
🟦 QUICK-QD3-K1	0.8	0.86	0.8
🟥 QUICK-ED-K3	0.67	0.91	0.72
🟩 QUICK-KW-K5	0.71	0.77	0.72
🟪 IR	0.07	0.11	0.08
🟦 Syntax-Only	0.09	0.09	0.09

Figure 4.11: Average precision, recall, and F_1-score for the entity retrieval task.

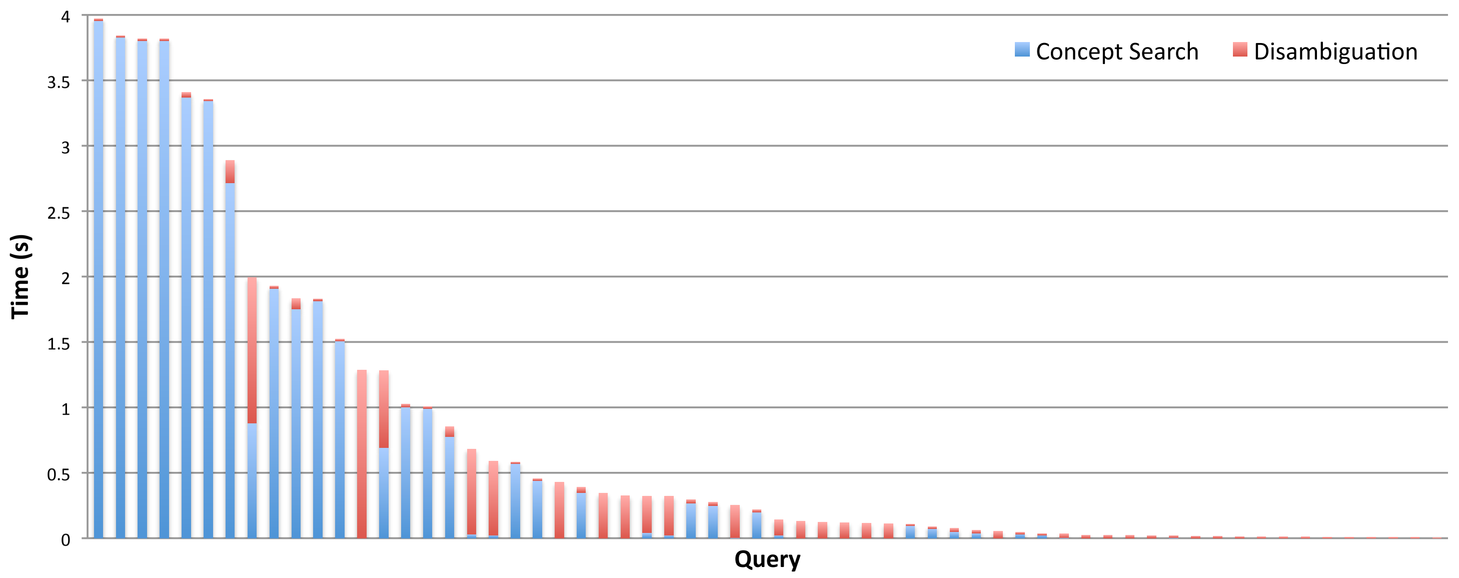


Figure 4.12: Performance of structured keyword query disambiguation and evaluation for the synthetic workload.

work is in efficiently generating and ranking the candidate networks, in contrast to our work, where the shape of the disambiguation graph is fixed by the structure of the query. Our approach also considers mapping against schema constructs, which is not considered by candidate networks. This difference is important as it allows users to create abstract descriptions of information needs at the schematic level, as opposed to simply expressing keywords that may appear in the text of desired results.

The WebTables project [18] also processes keyword queries over large Web-extracted data sets. While structured queries are not considered, their notion of probabilistic schema co-occurrence could be seen as a semantic similarity metric and used to support a disambiguation model like ours.

Our proposal differs from structured query languages over Web-extracted data, such as [20, 23, 49, 54], in the integration of ambiguous terms into the structured part of queries. NAGA [54] does support some ambiguity in queries via regular expression and keyword matching against strings in a special *"means"* relation that is part of YAGO. Query processing is done by finding all matches and evaluating the constraints in the query against candidates. This is in contrast to our heuristic-based disambiguation as a preprocessing step to query evaluation. Zhang et al. proposed Semplore, a semantic search engine that processes hybrid queries: structured queries that allow keyword expressions to optionally appear in the place of entities or types [108]. All entities or types that syntactically match the given keywords are then interpreted as being instances of the keyword expression. This is in contrast to considering different matches as distinct interpretations and ranking query interpretations separately. The EntityRank system [23] incorporates keywords into structured queries as so called context keywords. However, EntityRank does not attempt to construct complex queries from the input query, and entity types are explicitly given by the user with no ambiguity. Elbassuoni et al., considered annotating variables in a SPARQL query with keywords, and then used keyword matching and ranking against predicates that bind to the variable [36]. This is in contrast to our structured keyword query language which is founded entirely on keywords.

Chapter 5

Personalized User Interest Models for Entity Ranking

Many keyword queries issued to search engines are ambiguous and underspecified descriptions of a user's information need. Without some representation of the user's mental context, it is often impossible for a search engine to reliably interpret many of the vague keyword queries faced by search engines every day. For example, the query *"southwest"* could be interpreted in many ways. There is the `Southwest_Research_Institute`, a music album named `Southwest` by a famous hip-hop artist, as well as the airlines `Southwest_Airlines` and `Air_Southwest` among other possible interpretations. Without any context, an entity-based search engine may rank the music album highest, since its label is an exact match to the user query. However, given the knowledge that the user is interested in airlines, the search engine may prefer a significantly different ranking, one that prioritizes the two airlines. Furthermore, if the search engine was aware that the user had a preference for US based airlines, it could rank `Southwest_Airlines` (an airline based in the US) over `Air_Southwest` (a British airline[1]).

Search engines aiming to personalize the ranking of results must infer a user's search context based on the previous queries issued during the user's *search session*, and their *click behaviour* when presented with candidate results to a query. Mining a user's search and click history is a rich area of research and is heavily used in commercial search engines.

In this chapter we explore the possibility of cross-referencing Web search query logs with large Web-extracted entity databases in order to build session-level profiles of the

[1]Although the company Air Southwest is no longer in existence, it was an active business at the time our query log was collected, and is thus a relevant entity for our example.

Query	Baseline	Context-aware
"american airlines"	**American_Airlines** Pan_American_Airlines American_Airlines_191	**American_Airlines** Pan_American_Airlines American_Airlines_191
"southwest"	Southwest_(album) Air_Southwest **Southwest_Airlines**	**Southwest_Airlines** Air_Southwest Southwest_(Album)

"continental"	Continental_(magazine)
	Continental_(album)
	The_Continental_(TV)
		Continental_Airlines
		Continental_(magazine)
		Continental_(album)

Figure 5.1: Baseline and context-aware top-3 result lists for queries in an example session. The context-aware approach learns user search context from previous result clicks (shown in bold).

types of entities a user is interested in. Entity types can be general such as `Person` and `Organization`, or they may be very specific types such as `ETH_Zurich_alumni` and `Airlines_of_the_United_States`. We base our approach on the hypothesis that entities of the same type tend to co-occur in search sessions. We analyse a large click log from a commercial search engine to show support for our hypothesis. We then show how an entity-based search system can exploit this knowledge to produce a more effective ranking of entity results. As an example, consider the three-query session illustrated in Figure 5.1. A baseline entity ranking function simply orders entities based on their syntactic similarity to the entity labels. A context-aware reranking learns the user is interested in airlines based on the results clicked. Results of subsequent queries are reranked to give higher ranking to entities that have an entity type of interest.

5.1 Data & Problem Definitions

5.1.1 Search Engine Click Logs

A *search engine click log* is a recorded collection of user interactions with a search engine. The main information recorded in a click log are a user identifier, a query string, and the URL clicked by the user (if any result is clicked) in response to the search engine presenting candidate results for the given query string. If a user clicks multiple results for a query, each click forms a separate entry in the log. A click log generally also records additional data such the rank of the clicked result in the list of presented candidate results and a timestamp indicating when the event occurred. Formally, we can model a click log $\mathcal{L}$ as a set of four-tuples

$$\mathcal{L} = \langle \text{user\_id, timestamp, query, [url]} \rangle$$

where parameters in square parenthesis are optional, depending on whether or not a URL was clicked for the given query. A keyword query Q is expressed as an ordered set of terms $q_1 q_2 \ldots q_n$ over a concrete domain $\mathbb{S}$ as defined in Definition 5.

In query log mining, a user click on a URL in response to a query can be seen as an indication of the intent of the query. Essentially, if the user clicked on a result URL, it is assumed that the URL somehow satisfies the information need of the user. In the context of query log mining, the clicked URL forms a proxy of the user intention. For example, if the user types the query *"southwest"* and then clicks on the URL `http://www.southwestair.com`, it is assumed that the intention of the query is to find the website of the airline Southwest Airlines. Of course this is a very noisy signal, sometimes users click on a URL to find it is not relevant, and subsequently go back to the result list to find a different result, or re-issue a modified version of their query. For example, after issuing the query *"southwest"* and being presented with airline results, a user may re-issue their query as *"southwest research"* to find the homepage of the Southwest Research Institute. We leave the integration of negative feedback for future work.

5.1.2 Entity Databases

An *entity database* encodes information about real-world entities. We use a broad definition of an entity database as the techniques studied in this chapter can be applied to a variety of data sources, independent of their particular encoding. An entity database can be encoded as a knowledge base (as in Section 1.2.5), though any format that captures some particular information about entities could be substituted. There are three core types of information we require of an entity database. The first is that the database encodes the relationships between entities and their types. For example, the association between the entity `American_Airlines` and the type `Airlines_of_the_United_States`. The second requirement is that the database encodes some relationship between entities and URLs that are used to describe those entities on the Web. For example, relating the entity `American_Airlines` to the URL `http://aa.com` or the entity `Nikola_Tesla` to a URL that directs to an image of Nikola Tesla. The third type of information we require is a mapping of entities to some textual representation of the entity, such as a label.

Formally, an entity database is a set of tuples $\mathcal{D}$ of the form $\langle e\_id, type, t\_id\rangle$, $\langle e\_id, label, s\rangle$, or $\langle e\_id, homepage, URL\rangle$ where e_id is an entity identifier, t_id is a type identifier, s is an arbitrary string, URL is an identifier, $type$ is a relationship asserting that the entity denoted by e_id is of the type denoted by t_id, and $label$ denotes a relationship asserting that the entity denoted by e_id can be textually described by the string s.

Our entity database starts with a subset of the knowledge base described in Section 1.2.5 extended with other public knowledge sources. The entity database is described in detail in Section 5.3.1.

5.1.3 The Entity Retrieval Problem

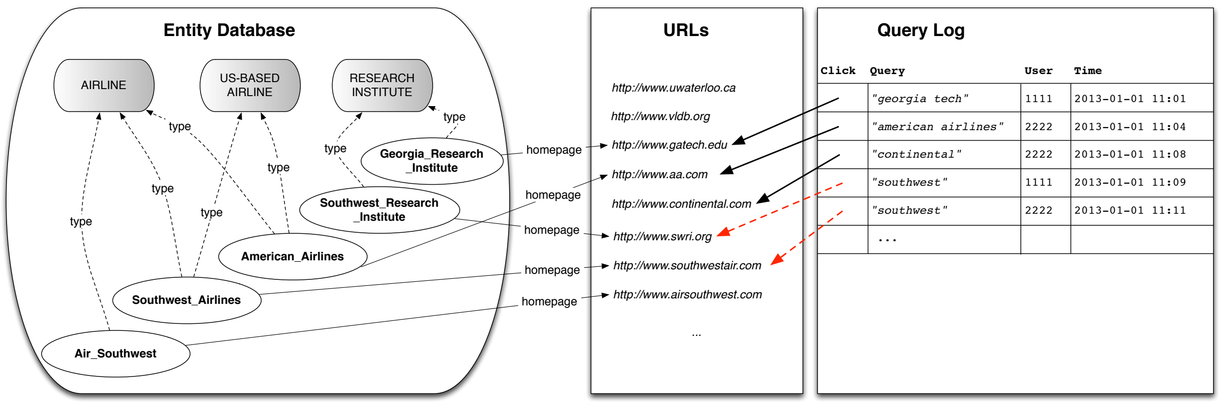
The problem of keyword search over entity databases (also known as entity retrieval, entity search, or object retrieval) is concerned with finding relevant entities given arbitrary keyword queries in an unconstrained vocabulary. The first proposal for an entity-based search task over a large Web entity database was given in [83]. The proposed task was the entity database variation of ad-hoc information retrieval, the task performed by document-based Web search engines such as Google, Yahoo, and Bing. The core difference of course being the structured entity database as a target data source, rather than a document collection.

While at first glance entity retrieval appears quite similar to document retrieval, there is a core difference: entity databases generally contain very little text. As such, the long history of techniques developed for document retrieval do not directly apply to entity retrieval. It has been the focus of recent research to develop efficient and effective techniques for entity retrieval (see for example [13, 80, 99]), as well as the focus of a series of search competitions [4, 31, 123, 124].

Formally, given a query Q, an entity search function $S : \mathbb{S}^{|Q|} \rightarrow e\_id^n$ takes a keyword query Q along with a number of desired results n and produces a set of entities determined to be the result of the query using some heuristic. A search function is generally paired with a ranking function $R : e\_id^n \rightarrow e\_id^n$ which takes the result of a search function and orders the elements according to some relevance heuristic.

5.1.4 Solution Overview

Our approach to identifying the entity of interest in response to a user's keyword query is based on learning a search context from the user's search history. We use the entity types from the entity database as a vocabulary to express a user's search context, and build a



Click	Query	User	Time
	"georgia tech"	1111	2013-01-01 11:01
	"american airlines"	2222	2013-01-01 11:04
	"continental"	2222	2013-01-01 11:08
	"southwest"	1111	2013-01-01 11:09
	"southwest"	2222	2013-01-01 11:11
	...		

Figure 5.2: An example entity database and query click log. Clicks on URLs are mapped to entities via their known homepages.

statistical model of the types of entities a user is interested based on how frequently they interact with entities of a given type during their search session. This model is learned by analyzing the URLs clicked by a user and then cross-referencing those URLs to the entities they represent.

The approach depends on an assumption that multiple entities of the same type are likely to queried (and clicked on) in the same search session. We explore this hypothesis empirically in Section 5.3.

Figure 5.2 shows an example entity database and query click log. The log shows a sequence of queries for two different users, one searching for research institutions and the other for airlines. We want to predict the clicked URL for a given user based on that user's past search history (i.e., the dotted red arrows). This will aid in answering ambiguous queries with user specific ranking, such as the query *"southwest"*.

Our methodology is based on the assumption that entities of the same type tend to co-occur in a user session with a probability higher than that of random chance. In Section 5.1.5 we analyze a real user click log in order to explore this assumption. We exploit user clicks on Web search results as a form of relevance feedback, and map user clicks on document URLs to entities in an entity database.

The validation of this assumption depends on how we define a user session and what entity types we consider to be relevant. A session may be only the most recent queries, queries issued over the past hour, over the past day, or even all queries ever issued by a user. In terms of relevant entity types, it is not always the case that all types are useful. For example, a hypothetical type `Thing` does not convey useful information since every entity is considered to be a type of `Thing` and thus every entity would be considered as co-occurring with an entity of the same type. We will consider various granularities of entity types in our analysis.

5.1.5 Analysis of an Entity Click Log

We conduct our analysis over the widely available AOL query log. This log contains 38 million search interactions issued by over 650 thousand different users over a three month period in 2006. While the log does not provide the complete URL clicked by users, it does provide the clicked domain which is sufficient for our purposes.

Our analysis starts with the key insight that we can link a Web query click log to an entity database by exploiting known home pages for entities. Since many real world entities have known home pages, we can link a query resulting in a click on an entities home page to the entity itself. For example, a query resulting in a click on `http://www.aa.com` allows us to infer that the query was seeking the entity `American_Airlines`. After cross-referencing all click log interactions that have a clicked URL corresponding to a known entity, we have a log of about 3.8 million queries. These queries are segmented into 372,611 separate user sessions, with 278,756 non-unary sessions. The non-unary sessions have an average session length of 11.71 unique queries (duplicate query/click pairs are removed) with a standard deviation of 43.98. The shortest non-unary sessions consist of a two queries while the longest session has 21,039 queries. Single query sessions provide no information for our type co-occurrence analysis, and while small sessions are less ideal, we include them as to not make arbitrary decisions about which sessions are large enough to include.

The integration of a click log with an entity database however, does produce a considerable amount of noisy data. Users may click on a URL which they subsequently decide is irrelevant, they may click on a URL unrelated to their query in an exploratory manner, or they may click on a URL that describes a different entity that is related to the intended entity (e.g., searching for one airline and clicking on a different airline, or searching for a music album and clicking on the homepage of the artist). We expect many of these phenomena to be infrequent enough as to not affect a statistical model learned over millions of queries. Another potential problem with this form of linking clicks to entities is that there are websites that encode information about many entities such as `http://en.wikipedia.org` (an encyclopedia) and `http://www.imdb.com` (a movie database). In many cases, a query leading to a click on `http://en.wikipedia.org` is likely not seeking the entity `Wikipedia`, but rather some other entity described by that site. Such sites can accumulate many clicks since they encode information about many different entities. We explore threshold-based filtering methods to deal with theses scenarios.

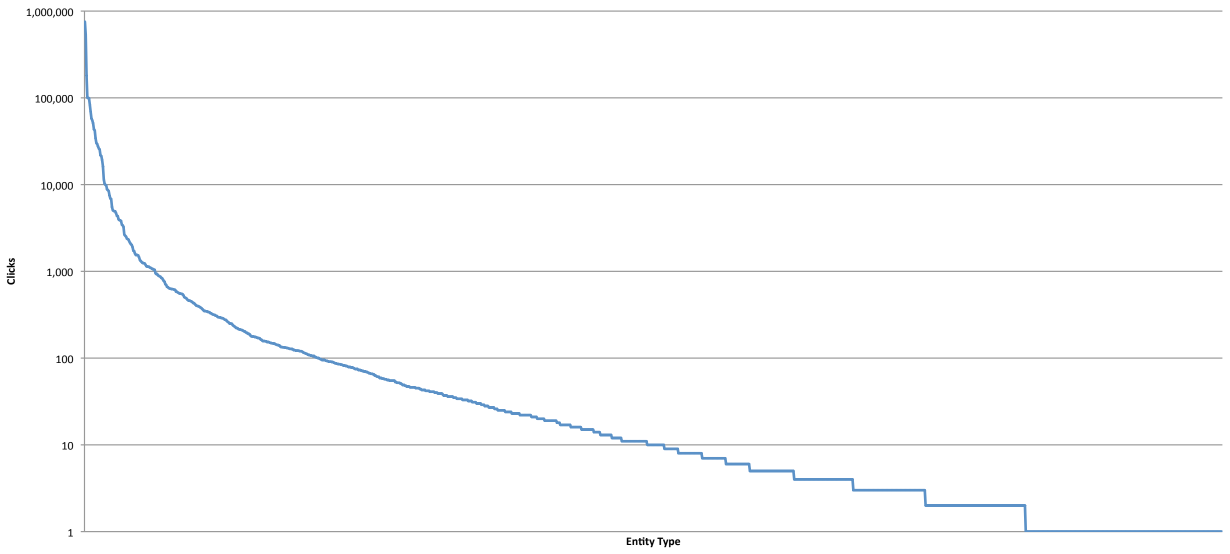


Figure 5.3: Distribution of click frequency of entity types clicked in the query log.

After linking entries in the click log to entities in the database, our next step is to analyze the distribution of entity types that are clicked in response to queries. Figure 5.3 shows the distribution of clicked entity types over the query log. This figure illustrates the exponential distribution exhibited by clicks on entities with given types. We see that some very popular entity types are clicked nearly one million times, while others are clicked much less frequently. There are two possible reasons why an entity type may be clicked frequently: the entity type is very popular; or the entity type is very general. An example of the first scenario is the entity type `Company`, a very popular entity type since many queries result on clicks on homepages of companies. An example of the second scenario is the entity type `Artifact`, a very general type that describes many entities in the database. Figure 5.4 shows the distribution of entities over entity types in the entity database. We see again an exponential distribution, showing that some types are very general (describe many entities) and that many others on the tail of the distribution are much more specific, describing relatively few entities. Distinguishing between these two scenarios is difficult and not always possible. We may want to preclude very general entity types from a learned user profile since they likely do not convey much useful information. For example, knowing a user clicked on an entity type `Thing`, for which all entities are a `Thing`, does not convey useful information. Knowing a user clicked on the type `Airlines_of_the_United_States` is much more meaningful. We explore various ways of filtering entity types in Section 5.3.

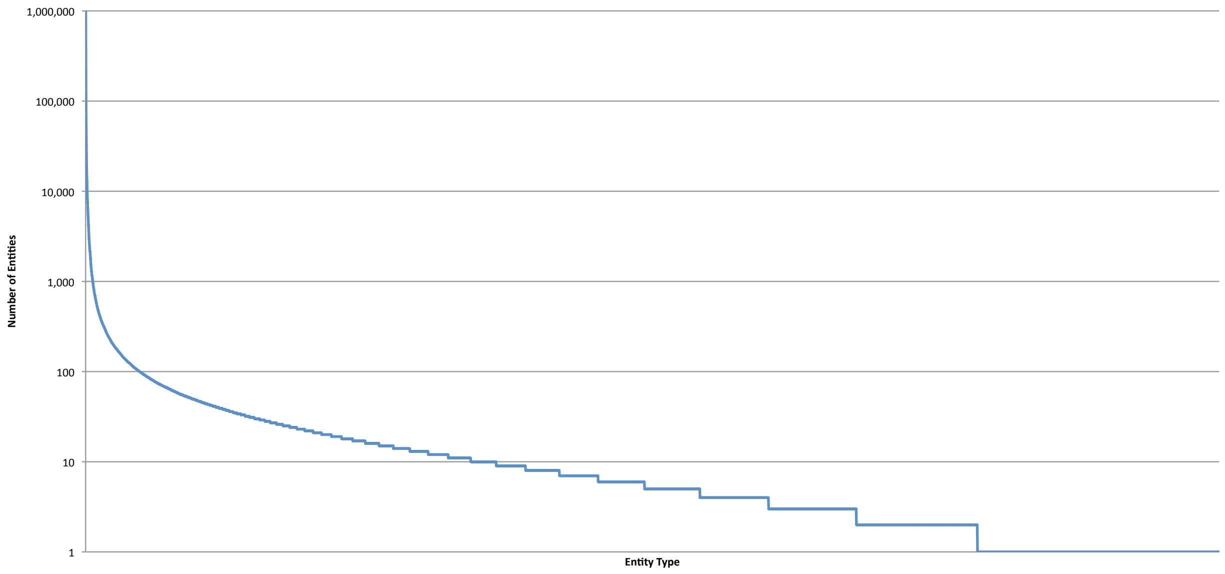


Figure 5.4: Distribution of entity count for entity types occurring in the entity database.

5.1.6 Entity Type Co-occurrence In User Sessions

The user interest model explored in this chapter is based on the premise that users searching for entities of a particular type will favour entities of the same type in future searches. This implies that entities of the same type should tend to co-occur more often than random chance. Quantifying precisely the co-occurrence of entities types in a user session is a difficult task, as it depends directly on what types are considered relevant. If we consider all types in our entity database, we find that %99 of non-unary sessions contain clicks on at least two entities of the same type. We also find that %72.8 of all queries result in a click on an entity of that has the same type as a previously clicked entity in the same session. However with extremely general types such as `Artifact` and `Physical_Entity` it is not clear how useful all of these type co-occurrences will be for learning a personalized user interest model. We explore this empirically in Section 5.3.

5.2 Personalized Ranking of Entity Queries

In this section we demonstrate how to exploit knowledge of a user's search context to improve the effectiveness of entity ranking. A per user profile is maintained that describes entity types of interest to the user along with the frequency that an entity of each type is clicked. We aim to exploit this information along with a baseline ranking algorithm in order to produce a more effective reranking of the results than the baseline approach.

5.2.1 Representing User Interests

Our model of user interest is based on annotating entity types from the database with the number of interactions the user has with entities of that type in during the a particular session. These interaction frequencies can then be used in various ways for entity ranking (see Section 5.2.3). We maintain a set of annotated entity types for each user.

Formally, a user model $\mathcal{M}$ is a set of tuples $\langle u\_id, t, count\rangle$ indicating that the user denoted by u_id clicked on an entity of type t $count$ number of times.

The simplicity of this user interest model allows for very efficient storage and maintenance. For each user, we need only store the entity types which have a non-zero count, and can reset a user's session by simply deleting the type annotations associated with that user. Such a design lends itself well to standard data structures with key-value interfaces.

This representation also allows on-line learning, so that user models can be built and modified in real-time as queries are issued and do not require any offline training.

5.2.2 Baseline Ranking Approaches

We employ a simple search function with disjoint boolean retrieval semantics. Our search function employs a standard stop-word list, breaks words on punctuation characters, and accounts for lemmatized forms of words in searches.

$$S(k) = \{e\_id \mid \exists\langle e\_id, label, s\rangle \in \mathcal{D}, \exists t_l \in s, t_k \in k, t_l \approx t_k\}$$

where $\approx$ computes equality of terms in a vocabulary accounting for lemmatized forms (e.g., *"airline"* $\approx$ *"airlines"*).

The search function is always paired with a ranking function that uses one of the following four baseline ranking heuristics for keyword search over entities.

1. **Label Similarity (labsim)** The first approach based on searching known text labels of the entities using a standard inverted indexing approach, followed by a ranking of candidate entities based on their syntactic similarity to the query text. We use q-gram similarity with a q value of 3 as the syntactic similarity function.

2. **Label Search (lab)** The second approach is inspired by the YBCN approach[14] that won the SemSearch 2010 competition [123]. This approach forms a virtual document from all text properties of an entity then performs classical document retrieval using an inverted index. In our case the text fields of most entities correspond to various labels of the entity, and the ranking algorithm is Lucene's tf·idf.

3. **Summary Search (abs)** The third approach is inspired by the NTNU approach from Balog et al., [3] that had the best combined score at the SemSearch 2011 competition [124]. This approach pulls a text description of each entity from Wikipedia and uses classical document retrieval over the Wikipedia text. In our case we are using Wikipedia abstracts for each entity.

4. **Summary Similarity (abssim)** The last approach performs an initial search using the abstract search described above, followed by a reranking based on similarity of the entity's label to the query as described in the first approach.

5.2.3 Personalized Entity Ranking

Given the output of an entity search using some baseline ranking function, the goal of personalized entity ranking is to rerank the entity list based on the user interest model of the user issuing the query. A personalized entity ranking function is then a ranking function which considers personalized context in the scoring heuristic. We consider five different scoring heuristics that make use of the user model and in some cases also the rank of the entity as determined by the baseline ranking algorithm. All of the reranking heuristics are based on the learned user preference model presented in Section 5.4, and the hypothesis that a user will be more interested in entities that have an entity type the user has previously interacted with. This is achieved by defining a ranking function to rerank the entity list while exploiting a user's interest model.

In all cases, we learn the user's interest model on-line. As queries are issued and results are clicked, the interactions are recorded in the model and those interactions are used to rerank subsequent queries. There is no offline training of the interest models required. The reranking functions are defined as a sort of one of the following scoring functions, with the underlying baseline ranking function being preserved in the case of ties (the rerankings are stable sorts). We start with a definition of type generality, a measure of how many entities in an entity database are asserted to have a given type. Formally, given an entity database $\mathcal{D}$, and an entity type t, the generality of t is given by the following.

$$generality(t) = |\{e \text{ s.t. } \langle e, type, t\rangle \in \mathcal{D}\}|$$

The scoring functions are defined as follows.

1. **Type Count (typecount)** This reranking approach directly ranks entities by the number of times the entity's types have been clicked by the user. For each candidate entity produced by the baseline ranking function, we consider all types the entity is known to have from the database. The total clicks for all of the entity's types is summed and the candidates are scored based on these type counts.

   Formally, given a query log $\mathcal{L}$, a user interest model $\mathcal{M}$, an entity e_id and a user ID u_id, the score of the entity represented by e_id for user u_id is given by the following.

   $$type\text{-}count(e\_id, u\_id) = \sum_{\langle e\_id, type, t\rangle \in \mathcal{D}} \sum_{\langle u\_id, t, c\rangle \in \mathcal{M}} c$$

   This reranking method can be viewed as an application of the proposal of Sperreta and Gauch [95] to entity databases, where the relevance of an entity to one of its asserted types is defined to be 1.0 (there is no probabilistic classification of entities to their types) and reranking is done purely based on the user model and not the baseline ranking ($\alpha = 1$) which was found to be the best performing configuration in their work. See Section 5.4 for more discussion of related work.

2. **Rank Discount (rankdiscount)** This reranking approach modifies the type count approach by giving more weight to the initial baseline ranking. In this approach, the type counts are discounted by the rank of the candidate entity in the baseline ranking. This makes it less likely an entity that is initially very lowly ranked will be reranked at high positions, dampening the effect of the type count reranking.

   Formally, given a query log $\mathcal{L}$, a user interest model $\mathcal{M}$, an entity e_id, a user ID u_id, and a function $rank(e)$ that gives the rank position of entity e in the baseline rank order, the score of the entity represented by e_id for user u_id is given by the following.

   $$rank\text{-}discount(e\_id, u\_id) = \frac{type\text{-}count(e\_id, u\_id)}{rank(e\_id)}$$

3. **Linear Discount (lineardiscount)** This reranking approach takes into account the *generality* of a type, discounting the score given by a type based on how many entities in the database share that type. The goal is that overly general types (those shared by many entities in the database) will have less effect in the ranking than more specific types (e.g., the type `Artifact` will have little effect on the score, while the type `Airlines_of_the_United_States` will have a more pronounced effect).

   Formally, given a query log $\mathcal{L}$, a user interest model $\mathcal{M}$, an entity database $\mathcal{D}$, an entity *e_id* and a user ID *u_id*, the score of the entity represented by *e_id* for user *u_id* is then given by the following.

$$\textit{linear-discount}(e\_id, u\_id) = \sum_{\langle e\_id, type, t\rangle \in \mathcal{D}} \left(\sum_{\langle u\_id, t, c\rangle \in \mathcal{M}} \frac{c}{\textit{generality}(t)} \right)$$

4. **Logarithmic Discount (logdiscount)** This reranking approach discounts the value a type contributes to a score based on the logarithm of the type's generality. This smoothed form of discounting is less drastic than the linear discount.

   Formally, given a query log $\mathcal{L}$, a user interest model $\mathcal{M}$, an entity database $\mathcal{D}$, an entity *e_id* and a user ID *u_id*, the score of the entity represented by *e_id* for user *u_id* is then given by the following.

$$\textit{log-discount}(e\_id, u\_id) = \sum_{\langle e\_id, type, t\rangle \in \mathcal{D}} \left(\sum_{\langle u\_id, t, c\rangle \in \mathcal{M}} \frac{c}{\textit{log}(\textit{generality}(t))} \right)$$

5. **Logarithmic *idf* discount (logidfdiscount)** This reranking approach discounts the value a type contributes to a score based on a normalized logarithm of the type's generality, inspired by tf·idf scoring in document retrieval. The type's generality is normalized by the size of the database, meaning types influence is proportional to the fraction of entities in the database asserted to have that type.

   Formally, given a query log $\mathcal{L}$, a user interest model $\mathcal{M}$, an entity database $\mathcal{D}$, an entity *e_id* and a user ID *u_id*, the score of the entity represented by *e_id* for user *u_id* is then given by the following.

$$\textit{log-idf-discount}(e\_id, u\_id) = \sum_{\langle e\_id, type, t\rangle \in \mathcal{D}} \left(\sum_{\langle u\_id, t, c\rangle \in \mathcal{M}} c \cdot log(\frac{|\mathcal{D}|}{generality(t)}) \right)$$

5.2.4 Filtering Overly General Types

While the various reranking approaches described in Section 5.2.3 provide various mechanisms for dealing with overly general types, we may benefit from directly filtering types deemed not to be useful. We investigate filtering overly general types using three different filtering thresholds. The first is to filter types with generality over a given threshold value. We define generality as the number of entities asserted to have the given entity type in the database (see Section 5.2.3). For example, the generality of the type `Musical_Work` is 110,086 since there are 110,086 entities known to be a type of `Musical_Work` in our entity database. Conceptually this filtering approach will remove overly general types, such as `Artifact` or `Physical_Entity`, that provide little information about the interests of a user. The second is to filter types that are clicked with very high frequency. This approach aims to filter noise in the data, such as the type `American_websites` becoming a highly relevant type because `http://www.google.com` is clicked frequently and is a type of `American_websites`. This causes any other entity that is asserted to be a type of `American_websites` to be reranked high in the ranking. The possible downside to this filtering approach is that legitimately popular types may be filtered. The third approach is to filter types with very low entropy in their click distribution. The motivation for this approach is that very low entropy in the click distribution of a type may be a sign of noise in the data. Returning to the previous example, the type `American_websites` has a click distribution consisting of many clicks on very few entities, and most clicks on the entity `http://www.google.com`. This click distribution has low entropy, i.e. it is very predictable. This predictability could be an indicator that a type is not useful in characterizing a user's interest. We explore these thresholding methods empirically in the following section.

5.3 Evaluation of Personalized Entity Ranking

We evaluate the effectiveness of personalized entity ranking by measuring the impact of reranking the results of a number of intuitive baseline entity ranking approaches. Our

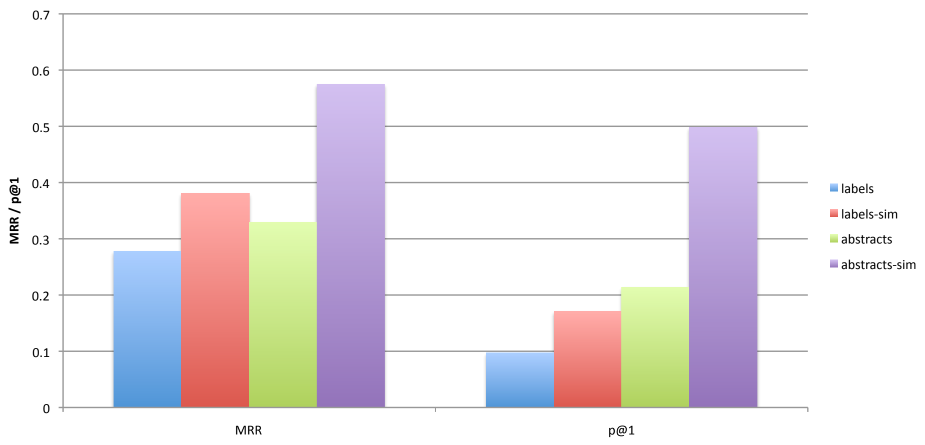


Figure 5.5: MRR and p@1 for the baseline ranking functions.

evaluation follows the entity retrieval evaluation methodology outlined in Chapter 6 with each query having only one relevant result (the clicked URL) and all other results considered irrelevant.

5.3.1 Data & Implementation Descriptions

We use the previously described AOL query log for the evaluation (see Section 5.1.5). This log contains 3.8 million queries that contain a click on a known entity. Because sessions in this subset of the query log can be rather sparse, we use all queries from a user as a definition of their session. Future work with larger click logs should explore varying the definition of a session as a parameter to the user model building algorithm.

To evaluate our reranking methods, we have compiled an entity database from various public sources, including Wikipedia [128], DBpedia [11], and YAGO [96]. Our entity database consists of 2,816,177 searchable entities of which 606,906 entities have known home pages and all have at least one textual label. The database is aware of 189,107 types over these entities with 22,356,866 assertions of entities having some type. The types in the database range from very general types such as `Person` to very specific types such as `Airlines_of_the_United_States`.

We implement all described reranking techniques in Java, with the baseline search and ranking functions implemented using Lucene[117]. Figure 5.5 shows the mean reciprocal rank (MRR) and precision at 1 (p@1) for the baseline methods. Note that p@1 can be

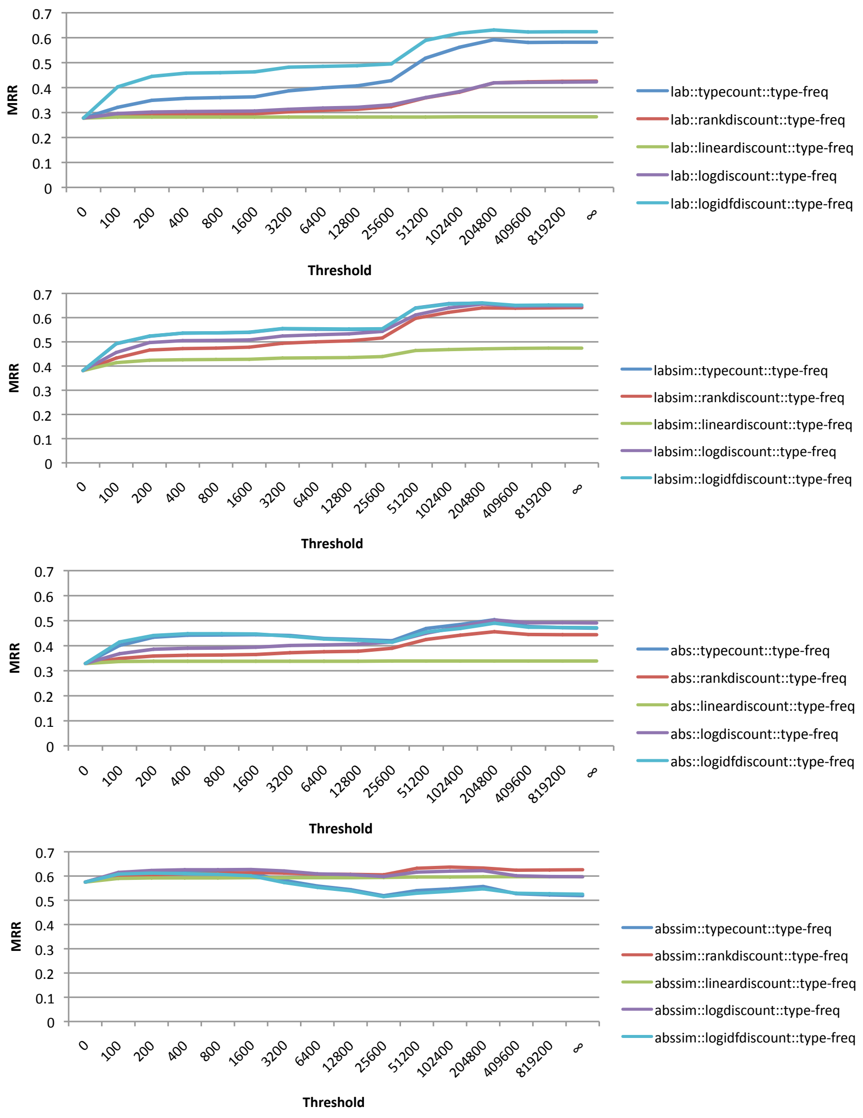


Figure 5.6: MRR values of each method vs. generality filtering threshold value. Graphs are plotted individually for each baseline entity ranking algorithm.

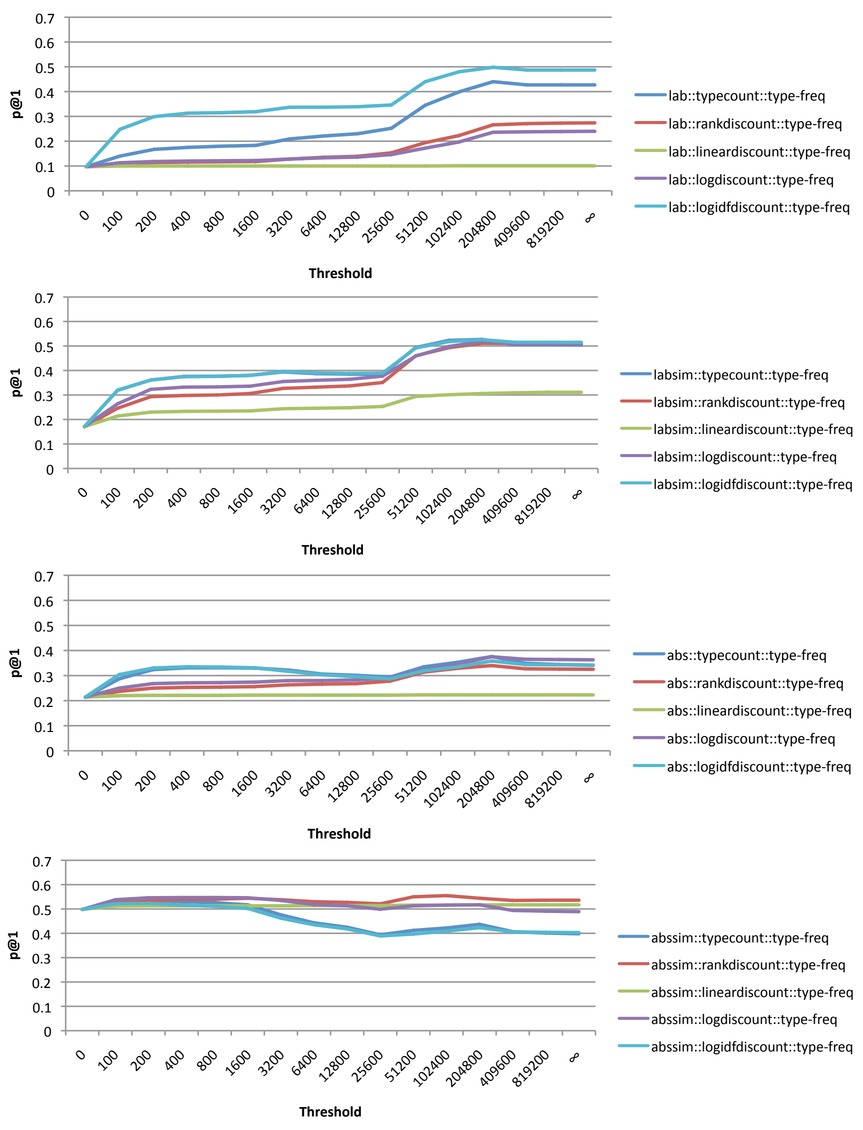


Figure 5.7: Precision at rank 1 for each reranking method vs. generality filtering threshold value. Graphs are plotted individually for each baseline entity ranking algorithm.

interpreted as the proportion of queries for which we can perfectly predict the entity the user will click for a query. We see that the abssim approach (text search over entity abstracts followed by similarity ordering of entity labels to query text) performs best for both metrics.

5.3.2 Personalized Entity Reranking

Figures 5.6 and 5.7 show the MRR and p@1 performance for each reranking approach over each baseline. Each figure shows one graph per baseline, with each of the five reranking approaches applied to each baseline. The graphs also show how the performance varies as a parameter of the type generality filtering threshold. A threshold value of 0 corresponds to the baseline ranking (no reranking) while a threshold value of ∞ corresponds to reranking using all types (no filtering of any types). The threshold filters any types with a generality greater than the threshold value (i.e., the higher the value, the fewer the number of entity types being filtered out). We see most baselines and reranking approaches have a general trend to higher performance with less filtering (i.e., including more types in the personalization model), though maxima are reached for most approaches with some level of filtering. Optimal approaches tend to occur at threshold values that filter the 12 (threshold=204,800) to 25 (threshold=102,400) most general types in the database. These overly general types, such as `Artifact` and `Physical_Entity`, carry little information for a user interest model and can cause spurious rerankings in the reranking methods. We also see that some reranking methods give little benefit or actually degrade performance for some baselines (see the absim graph in Figures 5.6 and 5.7). This appears to be particularly true for baselines that are already strong ranking methods. In contrast, we see that weaker baselines (lab and labsim) receive great benefit from reranking. These trends appear for both the MRR and p@1 measures.

Click frequency and entropy were also explored as filtering measures, however we found that any filtering using these measures degraded overall performance, and optimal performance was achieved when all entity types were included (threshold=∞). We therefore omit graphs of these approaches.

Figure 5.8 shows the best performing reranking approach for each baseline. It is interesting to see that the labsim baseline performs as well as the abssim baseline after reranking despite abssim being a much higher performing baseline. This shows that the reranking approaches can pick out entities of interest and rerank them effectively even in the presence of a less effective baseline ranking function. Overall, the reranking approaches are able to improve the effectiveness of all of the baseline ranking functions. Figure 5.9 shows the

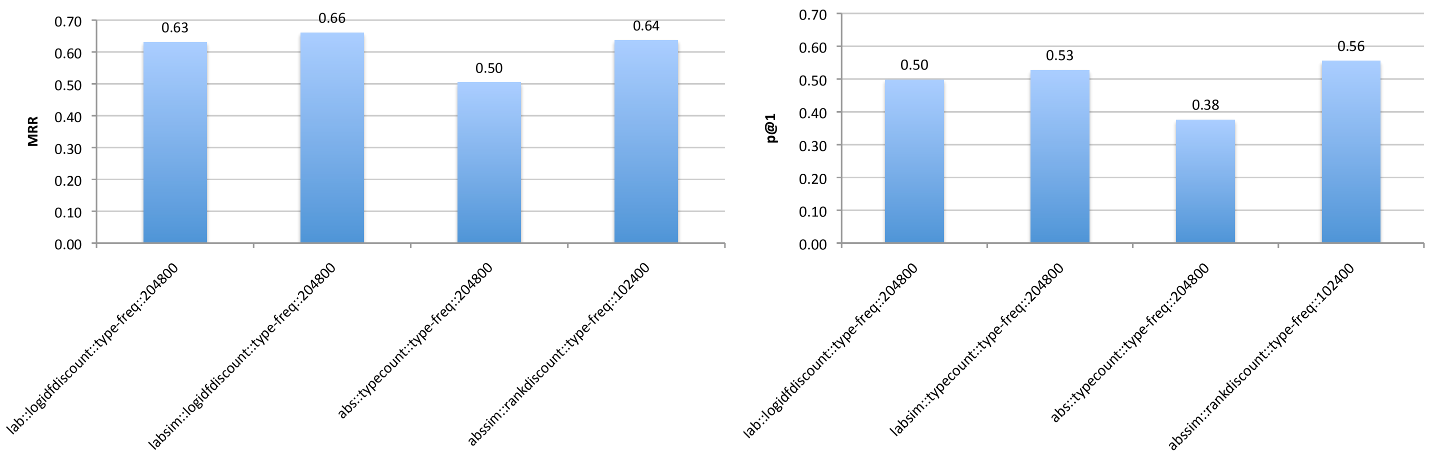


Figure 5.8: Best reranking approaches for each baseline.

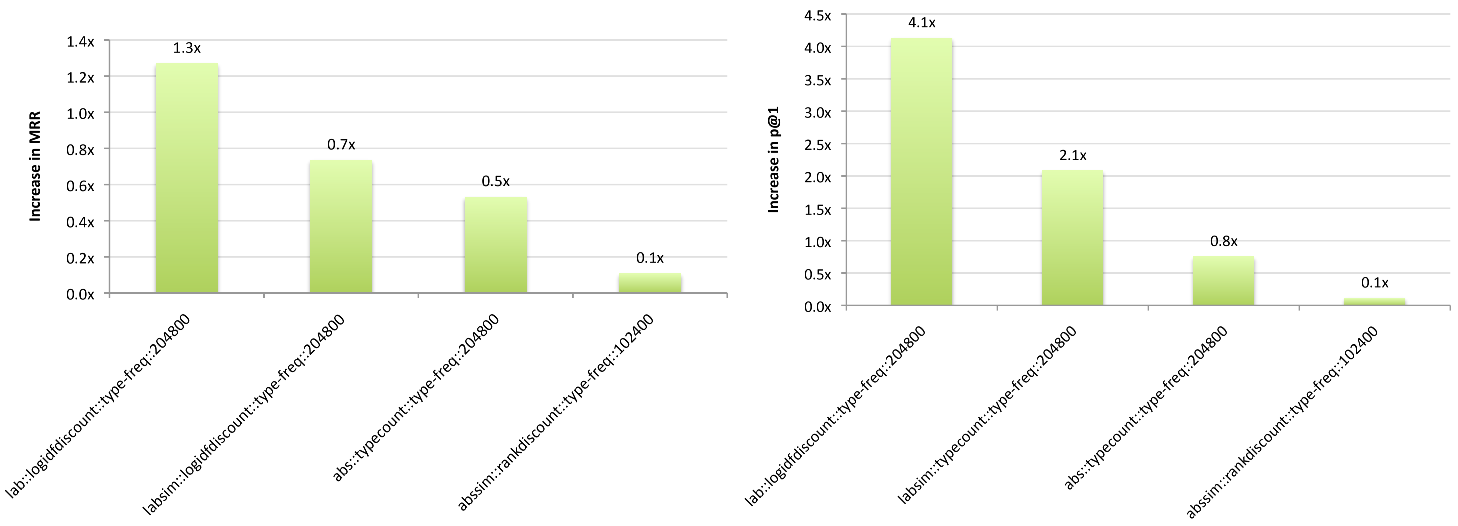


Figure 5.9: Increase in MRR and p@1 for best reranking approach for each baseline.

increase in MRR and p@1 for the best reranking approach for each baseline. Even the best baseline (abssim) gains an 11% increase in MRR and p@1, while the worst baseline (lab) sees a four-fold increase in p@1 again illustrating the ability of the reranking approaches to improve upon mediocre baseline ranking functions.

5.4 Related Work

There is a vast body of literature surrounding the topic of personalized search, with the majority focusing on keyword search over Web documents. Dou et al., give an empirical evaluation of some core methods for personalized search over (textual) Web documents. There has also been a considerable amount of work on entity ranking, however there has been very little focus on combining personalized search with entity ranking. Hristidis et al. have looked at scalability issues around personalizing authority flow computations over object graphs [46], as has Chakrabarti [21]. Their work focuses on computational challenges with (re)computing edge weights in an authority flow graph so that each user can have custom "page rank" style weights. These approaches require a base set of authority nodes for each user to be defined.

Previous research has also considered using a concept hierarchy to represent user interests. Pretschner and Gauch explored annotating a small 4,400 node concept hierarchy named Magellan with weights representing interest based on the documents the user has browsed [86]. Such an approach requires detailed knowledge of a user's browsing behaviour and cannot be extracted from a query log. If we substitute clicks on entities with given types for document classification into a small number of categories, their approach can be viewed as similar to our "typecount" approach, though they also integrate a notion of document relevance to a category which does not exist in our setting (every entity is by definition relevant to all of the types it has). Daud et al., took a different approach in annotating the concept hierarchy from the opened directory project (ODP), a small hierarchy of 1,896 classes, with term vectors extracted from documents classified into ODP categories [30]. Cosine similarities are then used to compute the relevance of a candidate document to a the ODP categories. While such an approach is feasible for a small number of categories, it does not scale to user profiles with thousands of categories. Chirita et al., also used the ODP hierarchy to model user interests, but required users to explicitly state which categories they were interested in [24]. This is in contrast to our approach of learning preferences from click logs.

The most relevant area of personalized search to our problem is that which aims to learn user interests from a query log using some form of concept hierarchy or ontology for representation. Speretta and Gauch have considered learning user interest models over a concept hierarchy from a user's search history [95]. They considered classifying document summaries from a search result page into the ODP classification scheme with a pre-trained classifier. Document reranking was then performed as a linear combination of the number of clicks on a class weighted by the importance of that class to the candidate result, and the original ranking given by the baseline search engine. Their results show best performance when using only the concept weight for reranking, and ignoring the baseline ranking ($\alpha = 1$). To this extent, their approach is equivalent to our "typecount" reranking approach since by definition each entity is relevant to the types it is asserted to have (there is no probabilistic classification of entities to types, as in scenario of text documents to categories). Liu et al., explored mapping query text into ODP categories [67]. In their work a user model associated the strength of a particular query term to a category, so that subsequent queries using the same terms can be personalized based on the learned weight for that term. These weights were learned by analyzing a user's search history. Such a model requires a large search history in order to learn the associations of the many possible keywords a user may use to describe their information need to each of the possible categories that the term may describe.

Chapter 6

A Methodology for Evaluating Entity Retrieval

Evaluating the quality of a ranked list of entities is a problem that has been considered throughout this thesis. In the following chapter, we describe and validate the methodology used for entity-based evaluation. This chapter presents work originally published in [83].

Despite a growing interest in entity-based search, there has been little work focusing on principled evaluation techniques for assessing the effectiveness of entity-based search systems. In order to evaluate the effectiveness of entity-based search technologies, as well as compare one system to another, a common evaluation methodology is needed.

Most current approaches to evaluating entity-based search systems are adaptations of document evaluation techniques from the information retrieval community (e.g., as in [40]). In this setting, search systems ultimately perform document retrieval, and the quality of documents returned is used as a metric of the quality of the entire system. These results make it difficult to interpret how well a search system functions internally, since a number of different factors can play a role in which documents are returned.

6.1 The Ad-hoc Entity Retrieval Task

In this work we try to map an entity ranking problem to the well known problem of *ad-hoc* document retrieval (ADR). Formalization of the ADR task and of its evaluation opened up the way for research in ranking models in Information Retrieval. By providing a clear problem setting, it allowed the creation of standard metrics and collections, allowing researchers in IR to communicate and collaborate. This in turn yielded many fundamental developments in ranking models, evaluation methods, efficiency and user models. A detailed overview of the ADR task can be found in Chapter 1 of [73], we summarize the task as follows:

- INPUT: a user keyword query Q and a collection of documents $\mathcal{D}$. The query has a query intent Z specified as a natural language description, which is not made available to the search engine. It is the job of the search engine to resolve among possible intentions represented by Q.
- OUTPUT: an ordered list of document identifiers $R = (d_1, d_2, ..., d_k)$, where $d_i \in \mathcal{D}$.
- EVALUATION: each document d_i is labelled with a score (independent of the rest) by a judge with respect to the information need Z describing the query Q.

Although ADR was initially developed for research in information science and library search, it had a natural application to Web search. ADR models well the typical situation for a user searching the Web: the user types a keyword query Q, and expects a ranked list of document identifiers R which represent Web pages relevant to their intention Z. For this reason, Web search engines from their infancy built on all the advances in ADR research, and adopted ADR metrics.

However, the Web is no longer simply a collection of Web pages. As the Web of Linked Data grows, we see entities of different types surfacing in Web search results. For example, for queries with a strong "local" interpretation (such as *"pizza in new york"*), the first results of commercial Web search engines are typically constructed from structured entity databases and not necessarily from Web pages. For such applications, Web search engines need to be able to effectively rank candidate entities.

For the analysis of such tasks, we propose to adapt the traditional ADR task. Intuitively we would like to replace documents by entities, leaving everything else the same. However, this brings up several non-trivial issues that need to be dealt with. These issues in turn will restrict our definition of the entity ranking task.

The ad-hoc entity retrieval task (AER) is defined as follows.

- INPUT: a user keyword query Q and a knowledge graph $\mathcal{K}$. The query has a query type T and query intent Z which are not explicitly defined. It is the job of the search engine to resolve among possible intentions represented by Q.

- OUTPUT: a ranked list of knowledge base item identifiers $R = (e_1, e_2, ..., e_k)$ such that each e_i occurs in $\mathcal{K}$. An identifier corresponds to a node in the knowledge graph.

- EVALUATION: each resource e_i is labelled with a score (independent of the rest) by a judge with access to all the assertions containing e_i, with respect to the query Q, query type T, and the query intent Z.

In the following sections, we highlight two of the key differences in evaluating AER and ADR: the dependence between the type of the query and the type of result returned, and the influential role played by result presentation. We then discuss metrics applicable to AER and demonstrate their applicability.

6.1.1 Query Types

In ADR, a human judge must evaluate the relevance of a document to a query. Could a human judge evaluate the relevance of an entity to a query? It would seem so intuitively. For example, if one queries for the name of a person, then any hCard (a micro-format encoding information similar to a business card) exactly matching the query would be judged relevant, and other hCard matching the name partially would be judged related but not as relevant. However, as we push our investigation of entity relevance further, we quickly run into several problems. In particular, results may be connected to relevant entities, but not directly relevant on their own. For example, a query for a movie could return an entity representing an actor that starred in the movie. Also, the expectation of what a result is may vary depending on the query. For example, a result may be a node, a set of nodes, a connected subgraph, or even a set of connected subgraphs.

Our first step towards studying this problem was to manually analyse a real Web search query log to try to understand how entity retrieval in the Web of Data could improve upon these answers, if at all. For each query in the query log, we manually annotated any entities, entity types, attributes, or relations occurring among the query terms (see Figure 6.1). Furthermore we annotated which was the primary intent of the query.

In our Web search log analysis we identified several query categories that would require different treatment in an AER engine. We established five such categories (see examples in Figure 6.1):

- Entity query: the intention of the query is to find a particular entity. Correct results would be entities corresponding to some interpretation of the query entity.

Query	Entities (Types)	Intent	Query Type
1978 cj5 jeep	cj5 jeep	cj5 jeep	Entity
applewood golf in windham nh	applewood golf, windham, nh	applewood golf	Entity
north texas eye doctors eye surgery	north texas (eye doctors, eye surgery)	eye doctors	Type
akita dog	akita	akita	Entity
best cold medication	(cold medication)	cold medication	Type
botanicals for hair	botanicals	botanicals	Entity
CARS FOR SALE IN AUSTIN	austin (cars)	cars	Type
cello players	(cello players)	cello players	Type
employment agencies w. 14th street nyc	w. 14th street, nyc (employment agencies)	employment agencies	Type
zip code waterville Maine	waterville, Maine	zip code	Attr.

Figure 6.1: A sample of queries with their entities, types, query intent, and query type.

- Type query: the intention of the query is to find entities of a particular type (or class). Correct results would be entities that are instances of the specified type.
- Attribute query: the intention of the query is to find values of a particular attribute of and entity or type. Correct results would be the values of an attribute specified in the query.
- Relation query: the intention of the query is to find how two or more entities or types are related. Correct results would describe the relationship among the query entities or types.
- Other keyword query: the intention of the query is described by some keywords that do not fit into any of the above categories. Correct results would be resources providing relevant information.

The existence of these query types has great importance to our problem, since each type requires a different type of result, and would thus have to be evaluated differently by the human judge. Any attempt to map the entity retrieval task to the ADR task will must consider the existence of these query types, and the differences required in relevance judgements. We believe this to be one of the significant distinctions between AER and ADR, and thus one of the most complicating factors in mapping the ADR task to entity ranking. We further investigate this issue in a quantitative analysis in Section 6.2.

6.1.2 Result Presentation

There is another important difference between entity search and document retrieval: what constitutes a result? In document retrieval it is clear that a single document is a result. Documents were invented by humans to be read, and therefore it is no challenge for a human judge to read a document result and decide its relevance. However, entity graphs on their own are not necessarily intended for raw human consumption. Resources are complex, structured, and heavily interconnected. For example, many properties may be needed to "define" an entity, and blank nodes are often used to hold information together. The correct unit of information for retrieval is unclear.

The ADR community has also encountered such problems when retrieving very large structured documents, for example books and XML documents. The main problem there is to define the right level of *granularity* of the results: sometimes relevance will be in a single sentence, sometimes in an entire section of XML subtree. Several approaches have been developed to tackle these issues, but they always lead to quite cumbersome task formalizations which demand a great amount of effort to the human evaluators (e.g., TREC sentence retrieval [94], and INEX entity search [52]).

We note that similar to the way a text retrieval engine provides an abstract of the search result, a typical entity-based search engine will return more than the identifier of the resulting resource. The search engine will provide more complete information to allow the user to inspect the result without retrieving its definition from the Web. The search engine may also highlight parts of the resulting graph, e.g. to show where the query terms have matched or which triples are most representative in defining the entity.

However, treating the resulting (possibly decorated) RDF graphs as a result would break our commitment to component-wise evaluation, because the relevance of the results would be tied to particular strategies for presentation. This would be similar to judging ADR methods by the quality of the snippet generator, or the rendering of the document. Take the following example as an illustration. Consider three entity-based search systems (S1, S2, and S3) that all return the entity `Jimi_Hendrix` (identified by `URI-1`) as a search result for the keyword query *"hendrix"*. The results are returned as follows:

S1: `URI-1`

S2: ⟨`URI-1`, *label*, *"Jimi Hendrix"*⟩

S3: ⟨`URI-1`, *label*, *"Jimi Hendrix"*⟩
⟨`URI-1`, *birthplace*, `Seattle`⟩
⟨`URI-1`, *hasColleague*, `URI-2`⟩
⟨`URI-2`, *label*, *"Mitch Mitchell"*⟩
⟨`URI-2`, *birthplace*, `London`⟩
⟨`URI-2`, *actedIn*, `Live_it_up!`⟩

It is evident from the example that, despite all three search systems having essentially returned the same result, there is an inherent difference in the quality of how each item is returned. A human evaluator would likely have difficulty judging the relevance of the result returned by S1, while the verbosity of the result returned by S3 seems to make it of lesser quality than that of S2. It is this difficulty that leads us to the separation of retrieval from presentation. This allows us to define an explicit evaluation method for the ranked retrieval of knowledge base resources with a fixed explanation strategy (and presentation strategy for that matter), leaving presentation as a separate task from retrieval.

For this reason, in this work we use resources as the unit of retrieval. We show to the evaluator a fixed presentation strategy consisting of all of the resource's contents and structure including the connections to other resources. This allows us to reuse human judgments of resources to evaluate different ranking strategies independent of how systems present results. Section 6.2 will discuss a particular evaluation tool and the guidelines used for the human evaluators, and will present results obtained from the evaluation of a baseline entity ranking system.

Judging a single resource, while making use of any information linked to that resource, seemed to us the best possible compromise for evaluation. It has the advantage of decoupling the evaluation of entities from its presentation. Furthermore, it makes no limiting assumptions on the uses an application may make of the entity; on the contrary, it assumes that the application will know how to utilize a resource identifier. It is this decoupling of relevance from presentation that allows the reuse of human judgments. We consider this to be of paramount importance for an AER task.

6.1.3 Evaluation and Performance Measures

One of the advantages of mapping the entity ranking problem to the ADR task is that we can reuse performance measures designed for ADR. Defining new performance measures is very problematic, because it requires studying many aspects of the measure, such as its generalization ability, its stability, appropriate statistical significance measures, etc. In fact, being able to use well established performance measures for the entity ranking problem was one of our main motivations in trying to map the problem to ADR.

Under the AER task definition, for a given query q with query type t, a retrieval system will return a list of entities $\{e_1, e_2, \cdots, e_n\}$, which are evaluated and given relevance values $R(e_i, q, t)$. We consider three popular metrics from the IR community, normalized discounted cumulative gain (NDCG), mean average precision (MAP), and precision at k (p@k) for a k value of 5. Details of these metrics can be found in Chapter 8 of [73].

6.2 An Empirical Study of Entity-based Queries

In order to evaluate a retrieval task such as ADR or AER, one needs to set up a search task including data collection and human evaluators. As we saw in the previous section, in the case of entity ranking, we also need to fix the resource presentation method. In this section we set up a realistic evaluation of an AER task on the Web. We use real query logs from a commercial search engine and a real large-scale collection of resources from a crawl of the Web of Data.

For resources, we focused on metadata publicly available on the Web. We used a subset of 240 million Web pages which have been crawled by Yahoo! and contain some form of metadata (RDFa and various types of microformats). This produced approximately 8 billion triples when normalized as RDF, or about a 1.1 terabyte knowledge graph (uncompressed).

In the evaluation we used the annotated query set described in Section 6.2.2. The advantage of this approach is the diversity of the information needs that query logs capture, i.e. a sample of the aggregate information needs of Web users in the US geography. As opposed to a manually created or synthetic benchmark these queries also provide real information needs. Lastly, since our data set has been gathered from the Web it is natural to rely on Web queries for our evaluation. The disadvantage of our method is that the original intent of the query is not directly available. We deal with this situation simply by allowing evaluators to skip the queries that they do not understand. We do not instruct evaluators to skip the queries where multiple interpretations are possible but rather to consider all possible interpretations as valid. For example, a query for a person name could be correctly answered by returning an entity corresponding to *any* person having that name. In practice, the majority of queries can be easily interpreted by the evaluators.

As an entity retrieval engine we used the baseline system described in Section 6.2.1. The retrieval implementation uses an inverted index over the text fields occurring in the underlying triples. This index was used to retrieve a subset of resources, from which a subset of the RDF graph could be constructed to produce and rank results. We use MG4J [118] for our index.

With a fixed retrieval and presentation strategy, we conducted an evaluation of ranking semantic resources as follows. We computed the top-5 results of a baseline ranking strategy. We then did a human evaluation of relevance of the results, and also established the ideal ranking based on the ideal re-ordering using the human evaluations. For a stability comparison, we also use a random re-ranking of the top-5 results.

6.2.1 Baseline Ranking Approach

The baseline ranking strategy used is an adaptation of TF-IDF to RDF graphs. In this setting, we compute an IDF score for each term in the vocabulary with respect to each RDF property occurring in the graph. This means, for example, that the term “John” can have different IDF values for the properties “vcard:name” and “vcard:address”, a desirable property to distinguish among common terms in particular properties. In the example, the term “John” may be very distinguishing as a street name, while very common as a person name. We also blacklist a set of RDF properties that contain a lot of diverse text causing noise in the results. These properties were chosen manually based on an inspection of a sample of query results. The blacklisting includes properties such as review text and blog summaries, which tend to match a diverse set of terms while providing little insight into the meaning of a resource. While this is certainly not an ideal ranking approach in general, it provides sufficient quality as a baseline metric to evaluate our methodology.

6.2.2 Query Analysis

Our query log analysis was conducted as follows. First, the queries themselves were annotated by human judges. Judges segmented the query into entities, types, relations, attributes and remaining relevant keywords. Furthermore, judges annotated the intent of the query, as discussed in 6.1.1. The intent of the query is defined as the component of the query for which the query is primarily seeking information. Besides the primary focus, a query may be further tagged with additional information such as secondary entities that appear in the query to disambiguate the primary intent or to provide additional information. For example, the keyword query *"doctors in barcelona"* is primarily seeking information about *"doctors"*, which is an entity type. The entity *"barcelona"* which also appears in the query would be referred to as a *context entity.*

Table 6.1 shows the results of this study. We found a large bias towards entity centric queries, or queries that do not fit easily into a semantic classification ("Other keyword queries", for example, the query "nightlife in Barcelona" has the primary intention of finding information about "nightlife" within the context of the entity "Barcelona"). We found that more than half of the queries focus on an entity or entity type. Interestingly, among all of the "Other keyword queries" (those which do not have any semantic resource as the primary intention), 14.3% of them also contain a context entity or type. This means that over 70% of all queries contain a semantic resource (entity, type, relation, or attribute), with almost 60% having a semantic resource as the primary intent of the query.

This analysis shows that type and entity queries constitute the vast majority of entity-based queries and thus we restrict ourselves to these types of queries in our analysis. For systems where these query types have a particular significance, separate evaluations can be devised in the future.

Query type	Percent of queries
Entity query	40.60%
Type query	12.13%
Attribute query	4.63%
Relation query	0.68%
Other keyword query	36.10%
Uninterpretable	5.89%

Table 6.1: Distribution of query types for a sample of Web queries.

6.2.3 Entity Relevance Analysis

For each of the queries in the benchmark, we run the baseline entity retrieval engine (discussed in Section 6.2.1) and asked human judges to evaluate the results. For this, we developed an evaluation tool which allowed judges to see all the query information, and the resources being evaluated.

The explanation strategy we used was a form of *concise-bounded description*[1], where all properties directly connected to a qualifying resource are included in the explanation, and blank nodes are recursively expanded to provide explanations of how blank nodes connect to any qualifying resource. We limit the expansion to a maximum of ten properties to avoid overwhelming the user with high degree nodes (which are often large collections of tags, generally unrelated to the resource and can thus be considered as spam). We do not enforce a limit on how many blank nodes can be expanded, meaning the size of the RDF subgraph shown to the user is unbounded and can vary considerably.

The tool displays a query along with a single entry of a search result (the URI along with its expansion). A user can then rate the relevance of the result to the query on a four-point zero to three scale. We measure two dimensions of relevance, the relevance of the result to the main intention of the query; and relevance of the result to the full query text. For example, given the query *"john smith barcelona"* with entity set {*"john smith", "barcelona"*} and *intent* = *"john smith"*, we would evaluate the relevance of each query result to both the entity *"john smith"* as well as the full query. In this setting, any "John Smith" is relevant in the first case, while only the "John Smiths" who are associated with "Barcelona" are relevant in the second case. We design our experiment in this manner because we are interested in both ranking for entity based retrieval and for full query answering.

[1]`http://www.w3.org/Submission/CBD/`

Human judges were instructed to rate the relevance of the main entity, and only use the query explanation as aid in understanding what the entity is. Thus over-explanation or any irrelevant information in the explanation does not affect the relevance rating, ensuring we only measure result quality and not explanation quality. The judges were given the following two questions and a four-point scale for each.

- Is the *resource* in the result relevant to ⟨`the query intent resource`⟩?
- Is the *resource* in the result relevant to ⟨`the full text query`⟩?

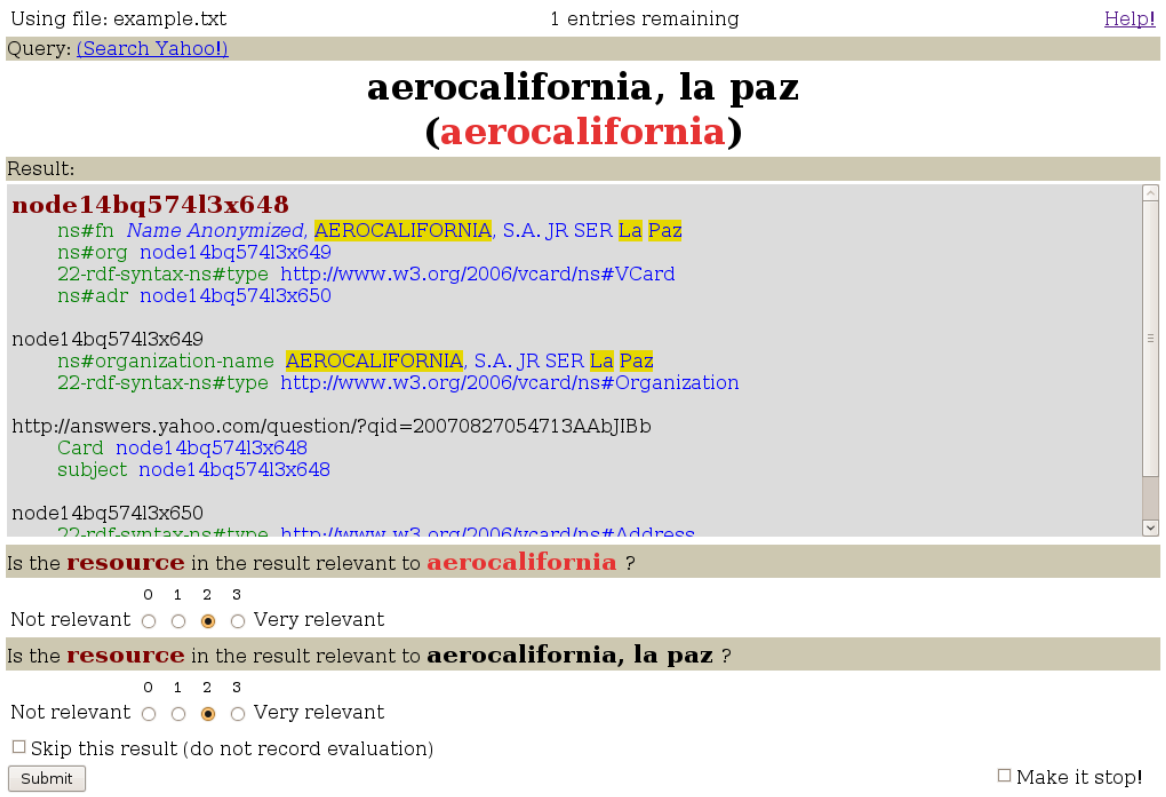
In each case, the words "resource" and the resource or query were highlighted and colour coded to coincide with the rendering of the query, query intent, and result resource. This ensured that the judge could easily distinguish what was being evaluated for each question. The description of the scale given to the judges was as follows.

- 0: Not relevant - the resource in the result is not at all relevant to the query resource.
- 1: Somewhat relevant - the resource in the result is moderately relevant to the query resource. For example, the result has the entity contained in a text property (review text, summary, etc...) or tag.
- 2: Relevant - the resource in the result is related to the resource in the query.
- 3: Perfect match - the resource in the result exactly describes the resource in the query. For example, the result is a VCard (address book entry) for a person entity in the query. In the cases of a type query, a result which is an instance of the type is also a perfect match.

A similar description was given for the task of rating the relevance of the result resource to the full query.

The tool is shown in Figure 6.2. The query, *"aerocalifornia, la paz"* is rendered in black, with the main intent of the query, *"aerocalifornia"* below in parenthesis. The result being shown is a VCard for a person who works for Aerocalifornia in La Paz (the name is anonymized for privacy reasons). Because this person entity is related to the intent entity of the query, it would be given a score of two by human evaluators.

With this mechanism we judged 1162 results from 264 queries, corresponding to the (at most) top-5 results produced by our baseline ranking algorithm. Such a *shallow* evaluation scheme is typical of Web search engine evaluation, where greater number of queries is preferred to large number of results evaluated per query.



Using file: example.txt 1 entries remaining Help!

Query: (Search Yahoo!)

aerocalifornia, la paz
(aerocalifornia)

Result:

node14bq574l3x648
ns#fn *Name Anonymized*, AEROCALIFORNIA, S.A. JR SER La Paz
ns#org node14bq574l3x649
22-rdf-syntax-ns#type http://www.w3.org/2006/vcard/ns#VCard
ns#adr node14bq574l3x650

node14bq574l3x649
ns#organization-name AEROCALIFORNIA, S.A. JR SER La Paz
22-rdf-syntax-ns#type http://www.w3.org/2006/vcard/ns#Organization

http://answers.yahoo.com/question/?qid=20070827054713AAbJIBb
Card node14bq574l3x648
subject node14bq574l3x648

node14bq574l3x650
22-rdf-syntax-ns#type http://www.w3.org/2006/vcard/ns#Address

Is the **resource** in the result relevant to **aerocalifornia** ?

Not relevant 0 1 [✓ 2] 3 Very relevant

Is the **resource** in the result relevant to **aerocalifornia, la paz** ?

Not relevant 0 1 [✓ 2] 3 Very relevant

☐ Skip this result (do not record evaluation)

Submit

☐ Make it stop!

Figure 6.2: A screen capture of the evaluation tool showing a query for the entity “Aerocalifornia” with context entity “La Paz”.

Query Intent Resource

	0	1	2	3
0	46	16	6	0
1	-	61	37	19
2	-	-	13	9
3	-	-	-	23

Full Text Query

	0	1	2	3
0	148	29	6	0
1	-	15	7	5
2	-	-	3	9
3	-	-	-	8

Figure 6.3: Inter-judge agreement on relevance scores from 0 to 3 for both relevance to the query resource and relevance to the full text query.

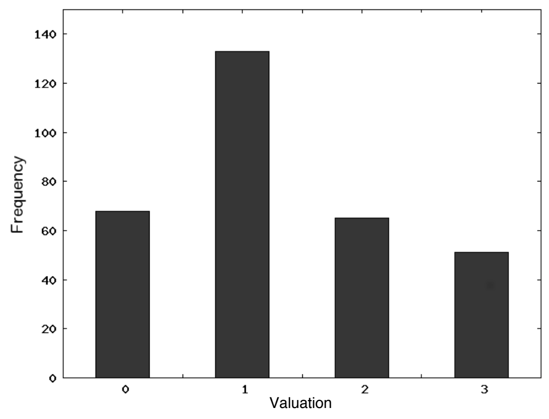
Analysis of Human Judgments

Even with a given description of the evaluation task, determining the relevance of an RDF resource to a query is a difficult process that can be highly subjective at times. To validate our human assessments, we had the judges evaluate an overlapping set of 230 query results and compared the assessments. Figure 6.3 shows a table of evaluation decisions for the four-point relevance scale. Each cell in the (symmetric) table shows the frequency of a judgment for the value of its respective row and column. Thus the diagonal entries show the frequency of perfect agreement, which occurs on 64% of the result evaluations for query intent, and 71% of the result evaluations for full query. If we also consider the cells adjacent to the diagonals, we see that the "off-by-one" agreement of judges produces an agreement of 93% for query intent 94% for full query. This indicates that our judges give generally similar valuations to results, despite the inherent subjectiveness of the task.

In Figure 6.4 a histogram for both relevance of query intent to the result and relevance of full query to the result is shown. These histograms illustrate the frequencies of each value on our four-point scale over the full result set. While our entity retrieval scores mostly in the moderate relevance range, the results are not very relevant to the full text queries. This is a result of our simple baseline metric used for retrieval. Our results, however, do contain enough diversity in relevance to conduct an evaluation for distinguishing ranking algorithms.

Stability

We evaluate the stability of our chosen metrics on the resource ranking task. In order to achieve a 95% confidence interval, we apply a standard bootstrap method [34]. We compute a sample of size n from our query workload of size n, sampling with replacement. From



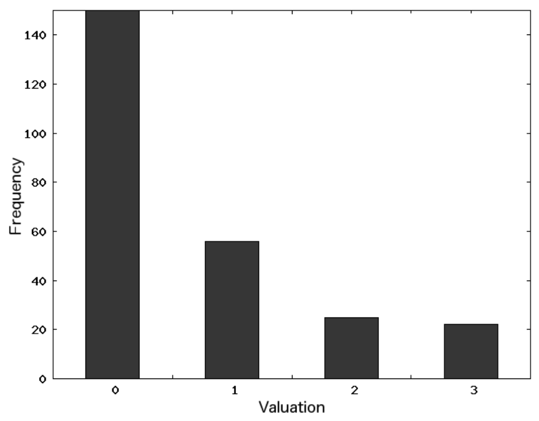


Figure 6.4: Frequencies of valuations from the four-point evaluation scale. Valuations were made for query intent (top) and full query (bottom).

this sample, we compute the mean value of the metric being evaluated over all queries in the sample. We repeat this process one million times to get a set of means, and compute the mean and standard error over the resulting set of means to obtain a 95% confidence interval. This final mean can then be compared with the empirical mean obtained from the actual workload to test the stability of the metric.

Figure 6.5 shows the results of our stability test. Averaged across many samples, the bootstrapped mean of the metric becomes very close to the empirical mean and produces a tight confidence interval at 95% confidence (we find a difference after the sixth decimal place in all cases). This indicates that the metrics are stable across samples of queries from the same distribution over RDF resource results. We note that this stability is also dependent on our data, the sampling method used to obtain our query workload from the Web query log, and the human relevance judgments. We must also make the assumption that the bootstrap sampling draws scores from a normal distribution to apply this analysis.

Discriminant Power

We aim to show that applying our test metrics to RDF resources with a fixed retrieval and presentation strategy has the ability to discriminate among ranking approaches in a meaningful way. We consider three ranking strategies, all based on re-ranking the same top-5 results returned by our fixed baseline retrieval strategy. The first approach is a random re-ranking of the top-5 results, the second is our baseline ranking algorithm described in

Measurement	Query Resource	Full Query
Empirical NDCG	0.953	0.972
Bootstrap NDCG	0.953 ± 1.02e-05	0.973 ± 9.08e-06
Empirical MAP	0.719	0.668
Bootstrap MAP	0.719 ± 7.04e-05	0.668 ± 7.69e-05
Empirical p@5	0.411	0.153
Bootstrap p@5	0.411 ± 5.25e-05	0.153 ± 4.15e-05

Figure 6.5: Stability of evaluation metrics for resource ranking. Confidence intervals are computed at 95% confidence. Results are shown for both result resource to query resource, and result resource to full text query.

	Query Resource			Full Query		
Metric	Random	Baseline	Ideal	Random	Baseline	Ideal
DCG	5.004	5.072	5.402	2.878	2.923	3.0670
NDCG	0.942	0.952	1.0	0.968	0.972	1.0
MAP	0.644	0.719	0.820	0.603	0.668	0.790

Figure 6.6: Distinguishing among the baseline, random, and ideal rankings for various evaluation metrics.

Section 6.2.1, and the third is the ideal re-ranking where results are ordered based on the valuations given by a human judge.

Figure 6.6 shows the results of the three ranking approaches. We see that for both the query resource relevance and the full text query relevance the scores distinguish among the ranking approaches as expected. This is a positive result for discriminating power, given how easy it is to obtain a high score with a random re-ranking of only 5 results. Note that we omit p@5 since re-ranking the same results does not affect the score, and thus all systems would be ranked equally.

6.3 Related Work

Due to the diverse nature of data models considered in semantic search systems (e.g., text, annotated data, ontologies) a wide spectrum of query languages are used for semantic search. These query languages range from expressive structured queries, like SPARQL [87], to simple keyword queries. A number of works also consider semi-structured variations (hybrid search) which integrate structured and keyword queries [81, 82, 89, 102, 108].

From the point of view of evaluation, structured queries have an explicit semantics and thus a well defined query result. Such systems can be evaluated using standard approaches from the database community, and as such, we omit them from our discussion. Keyword queries and semi-structured queries however, have uncertain results and are thus much more challenging to evaluate.

A number of end-to-end semantic search systems have been developed up to date, e.g. [32, 33, 41, 43, 69, 78, 80, 100]. Fernandez et al. perform an evaluation using the TREC benchmark [40]. TREC has been designed for measuring the relevance of results returned by text retrieval engines and finds its origins in the Cranfield experiments [26]. However, there are two potential problems in applying TREC measurements directly. First, many semantic search engines fall back to basic keyword search when the query cannot be understood in terms of the available semantic models. Thus it is not known how much of the evaluation tests the part of the system that actually employs semantics, and how much covers traditional keyword search. In fact, in the work of Fernandez et al. only 20% of the TREC queries used are covered by ontology concepts, the rest are removed leaving 20 queries for evaluation. Simply dismissing the queries that cannot be semantically interpreted is not easy as different search engines may have vastly different query understanding capabilities. Furthermore, reaching consensus on a query workload in this manner may be impossible as each approach to query interpretation can vary significantly. Thus, taking the intersection of queries that a set of semantic search engines can handle would likely produce an "easy" workload (if not empty), which cannot be justified as representative of real users. Evaluating each search engine only on the queries it can interpret also has drawbacks since the results obtained by two different search engines may not be directly comparable. The second and even more apparent problem is that not all semantic search engines perform document retrieval, but rather retrieve knowledge encoded in some semantic data model. In many contexts, the data in the system is not necessarily associated with any particular text document. This is the case for example with search engines that crawl and index Linked Data such as Sindice [78]. Even in cases where there is a document, an evaluation based on document rankings is not able to measure some of the key advantages of semantic search such as being able to give precise answers to factual questions or that answers can be computed by aggregating knowledge from different documents.

While TREC itself is thus not applicable to our scenario, the general concepts of the Cranfield experiments can be directly carried over. In particular, we take a system-oriented perspective with a reference collection, a fixed set of test queries and a set of relevance judgments, which form the benchmark. In this respect our work is similar to the INEX series [52], which is an adaptation of the methodology to XML content retrieval. Among the various tracks of INEX, the entity-retrieval experiments the INEX Entity Ranking Track is most relevant in terms of the query type. However, this INEX track also focuses on textual corpora and that means that systems compete also on information extraction functionality. In contrast, semantic search engines work with structured data.

In terms of the queries we consider, there are also commonalities to benchmarks that consider queries for particular types of entities such as the Web People Search Evaluation (WEPS) [2], or the expert finding task of the TREC Enterprise Track [93]. With respect to addressing keyword retrieval on structured data, there is also existing work in the database literature (e.g., [10, 45, 71]). This field of research has not produced a common evaluation methodology for ranking effectiveness that we could adapt.

Since the presentation of this research, a number of follow up projections have been completed. In particular, the exploration of crowd-sourcing relevance judgements in order to scale the size of the benchmark and to quickly develop evaluation campaigns [123, 124] and a comprehensive overview of the evaluation of semantic search systems [12]. Also, the SEALS project has been established with a core goal of developing accessible and reusable evaluation resources for semantic technologies [104, 122].

Chapter 7

Conclusions and Future Work

We have shown how keyword queries can be interpreted over large scale heterogeneous Web knowledge bases by learning semantic structures from an annotated query log. Our experiments verify that an accurate structuring model can be learned from a relatively small training set by performing the structural mapping from high level summaries of keyword queries to structured query templates.

We have proposed a keyword-based structured query language that trades off expressivity and flexibility in utilizing Web knowledge graphs. This language can serve as an end-user query language for expert users, or as an intermediate representation of the intention of keyword queries over knowledge graphs. We have explored ambiguity issues with basing a structured query language on keywords and proposed a solution for disambiguation. Our experiments demonstrate that our proposed disambiguation model can quickly achieve high quality disambiguations, even in the case where only partial knowledge is available.

We have shown that a user's click behaviour can be used to build a model of the types of entities a user is interested in, and that such a model can be effectively used to rerank entity results. We have shown effective reranking over four general baselines, though our reranking methods could be applied to any entity retrieval system.

We have outlined the details of a methodology for entity-based search evaluation over Web knowledge graphs. Our proposal builds on the well established ad-hoc document retrieval task, allowing us to reuse existing efforts. We empirically justified the applicability of ADR metrics to our entity ranking problem showing that our adaptations to entity retrieval are justified. We also have constructed a classification of Web queries from the entity retrieval point of view and have proposed the notion of a result and what relevance means in this context.

Future directions for research on interpreting keyword queries over knowledge graphs include exploring refinements on the granularity of semantic annotations. For example, distinguishing locations from other types of entities could provide additional structuring hints, since locations often appear as context information in keyword queries. Similarly, person entities often appear as the primary intention of queries. There are a number of parameters in the system that can control how much of the search space is explored (e.g., the number of semantic annotations to consider, the number of structurings of semantically annotated queries, the number of mappings of structured keyword queries into the knowledge graph), future work could explore learning parameter values or auto-tuning query dependent values. Future work may also consider applying these techniques to other knowledge graphs and search verticals such as product catalogues or medical knowledge bases. Query logs for such verticals may exhibit different structures and learning a model for structuring specific to a search domain may improve effectiveness.

Future work may also investigate the effectiveness of personalized user interest models for reranking entity results over other ranking baselines. Integrating negative feedback could also be beneficial, for example if a user clicks a different URL quickly after an initial click or if a user reformulates a query. This information could be used to ignore clicks in the learned model or even to negatively weight entity types. Also, varying the definition of session length to determine how long a user context is relevant, at and what point a user interest model should be discarded and a new model learned. It would also be interesting to investigate the possibility of propagating a user's search context to users with similar sessions using some form of collaborative filtering.

We have presented a procedure for concept search query evaluation in order to have a complete system for evaluation. Future work should explore and compare the performance of the proposed procedure to alternative implementations based on RDF databases that support inference over entity type hierarchies as well as knowledge graph updates. Such research would necessarily have to explore new benchmarks for knowledge graphs that emphasize reasoning over type hierarchies with variable mixes of update and search queries. Resources such as DBpedia Live[1] could form a basis for a real-world workload of knowledge graph updates.

[1]http://live.dbpedia.com

References

[1] Ganesh Agarwal, Govind Kabra, and Kevin Chen-Chuan Chang. Towards rich query interpretation: walking back and forth for mining query templates. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 1–10, New York, NY, USA, 2010. ACM.

[2] Javier Artiles, Satoshi Sekine, and Julio Gonzalo. Web people search: results of the first evaluation and the plan for the second. In *WWW*, pages 1071–1072, 2008.

[3] K. Balog, M. Ciglan, R. Neumayer, W. Wei, and K. Nørvåg. NTNU at SemSearch 2011. In *Proceedings of the 4th International Semantic Search Workshop (SEMSEARCH'11)*, 2011.

[4] K. Balog, A. P. de Vries, P. Serdyukov, P. Thomas, and T. Westerveld. Overview of the trec 2009 entity track. In *TREC 2009 Working Notes*. NIST, November 2009.

[5] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction from the web. In *IJCAI*, pages 2670–2676, 2007.

[6] Jeremy Barbay, Alejandro Lopez-Ortiz, and Tyler Lu. Faster adaptive set intersections for text searching. In *Experimental Algorithms*, volume 4007 of *Lecture Notes in Computer Science*, pages 146–157. Springer Berlin Heidelberg, 2006.

[7] Cory Barr, Rosie Jones, and Moira Regelson. The linguistic structure of english web-search queries. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 1021–1030, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[8] Holger Bast, Alexandru Chitea, Fabian M. Suchanek, and Ingmar Weber. ESTER: efficient search on text, entities, and relations. In *SIGIR*, pages 671–678, 2007.

[9] Jon Louis Bentley and Andrew Chi chih Yao. An Almost Optimal Algorithm for Unbounded Searching. *Information Processing Letters*, 5:82–87, 1976.

[10] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *ICDE*, pages 431–440, 2002.

[11] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia - a crystallization point for the web of data. *Web Semant.*, 7(3):154–165, September 2009.

[12] Roi Blanco, Harry Halpin, Daniel M. Herzig, Peter Mika, Jeffrey Pound, Henry S. Thompson, and Thanh Tran. Repeatable and reliable semantic search evaluation. *To Appear: Web Semantics: Science, Services and Agents on the World Wide Web*, 2013.

[13] Roi Blanco, Peter Mika, and Sebastiano Vigna. Effective and efficient entity search in rdf data. In *The Semantic Web – ISWC 2011*, volume 7031 of *Lecture Notes in Computer Science*, pages 83–97. Springer Berlin / Heidelberg, 2011.

[14] Roi Blanco, Peter Mika, and Hugo Zaragoza. Entity Search Track submission by Yahoo! Research Barcelona. Technical report, SemSearch, 2010.

[15] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3c recommendation, W3C, February 2004.

[16] Charles George Broyden, John E Dennis, and Jorge J Moré. On the local and superlinear convergence of quasi-newton methods. *IMA Journal of Applied Mathematics*, 12(3):223–245, 1973.

[17] Michael J. Cafarella. Extracting and querying a comprehensive web database. In *CIDR*, 2009.

[18] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. WebTables: exploring the power of tables on the web. *Proc. VLDB Endow.*, 1(1):538–549, 2008.

[19] Michael J. Cafarella, Christopher Re, Dan Suciu, and Oren Etzioni. Structured querying of web text data: A technical challenge. In *CIDR*, pages 225–234, 2007.

[20] P. Castells, M. Fernandez, and D. Vallet. An adaptation of the vector-space model for ontology-based information retrieval. In *IEEE Transactions on Knowledge and Data Enginering 19(02)*, pages 261–272, 2007.

[21] Soumen Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 571–580, New York, NY, USA, 2007. ACM.

[22] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[23] Tao Cheng, Xifeng Yan, and Kevin Chen-Chuan Chang. EntityRank: searching entities directly and holistically. In *VLDB*, pages 387–398, 2007.

[24] Paul Alexandru Chirita, Wolfgang Nejdl, Raluca Paiu, and Christian Kohlschütter. Using odp metadata to personalize search. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 178–185, New York, NY, USA, 2005. ACM.

[25] Charles L. A. Clarke, Gordon V. Cormack, and Thomas R. Lynam. Exploiting redundancy in question answering. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '01, pages 358–365, New York, NY, USA, 2001. ACM.

[26] C. Cleverdon and M. Kean. Factors Determining the Performance of Indexing Systems. Technical report, Aslib Cranfield Research Project, Cranfield, England, 1968.

[27] Francisco M. Couto, Mrio J. Silva, and Pedro M. Coutinho. Measuring semantic similarity between Gene Ontology terms. *Data & Knowledge Engineering*, 61(1):137 – 152, 2007.

[28] Carolyn J Crouch and Bokyung Yang. Experiments in automatic statistical thesaurus construction. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 77–88. ACM, 1992.

[29] Hoa Trang Dang, Diane Kelly, and Jimmy J. Lin. Overview of the trec 2007 question answering track. In *TREC*, 2007.

[30] Mariam Daoud, Lynda Tamine-Lechani, Mohand Boughanem, and Bilal Chebaro. A session based personalized search using an ontological user profile. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 1732–1736, New York, NY, USA, 2009. ACM.

[31] Gianluca Demartini, Tereza Iofciu, and ArjenP. Vries. Overview of the inex 2009 entity ranking track. In *Focused Retrieval and Evaluation*, volume 6203 of *Lecture Notes in Computer Science*, pages 254–264. Springer Berlin Heidelberg, 2010.

[32] Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, CIKM '04, pages 652–659, New York, NY, USA, 2004. ACM.

[33] Alistair Duke, Tim Glover, and John Davies. Squirrel: An advanced semantic search and browse facility. In *ESWC*, pages 341–355, 2007.

[34] B. Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

[35] Shady Elbassuoni and Roi Blanco. Keyword search over rdf graphs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, CIKM '11, pages 237–242, New York, NY, USA, 2011. ACM.

[36] Shady Elbassuoni, Maya Ramanath, Ralf Schenkel, and Gerhard Weikum. Searching rdf graphs with sparql and keywords. *IEEE Data Eng. Bull.*, 33(1):16–24, 2010.

[37] Oren Etzioni, Anthony Fader, Janara Christensen, Stephen Soderland, and Mausam. Open information extraction: The second generation. In *IJCAI*, pages 3–10, 2011.

[38] Ronald Fagin, Benny Kimelfeld, Yunyao Li, Sriram Raghavan, and Shivakumar Vaithyanathan. Understanding queries in a search database system. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '10, pages 273–284, New York, NY, USA, 2010. ACM.

[39] Fernando Farfan, Vagelis Hristidis, Anand Ranganathan, and Michael Weiner. XOntoRank: Ontology-Aware search of electronic medical records. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 820–831, Washington, DC, USA, 2009. IEEE Computer Society.

[40] M. Fernandez, V. Lopez, M. Sabou, V. Uren, D. Vallet, E. Motta, and P. Castells. Semantic search meets the web. In *Semantic Computing, 2008 IEEE International Conference on*, pages 253 –260, August 2008.

[41] R. Guha, Rob McCool, and Eric Miller. Semantic search. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 700–709, New York, NY, USA, 2003. ACM.

[42] Jiafeng Guo, Gu Xu, Xueqi Cheng, and Hang Li. Named entity recognition in query. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 267–274, New York, NY, USA, 2009. ACM.

[43] Andreas Harth, Jürgen Umbrich, Aidan Hogan, and Stefan Decker. YARS2: A Federated Repository for Querying Graph Structured Data from the Web. *The Semantic Web*, pages 211–224, 2008.

[44] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient IR-style keyword search over relational databases. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '2003, pages 850–861. VLDB Endowment, 2003.

[45] Vagelis Hristidis and Yannis Papakonstantinou. Discover: keyword search in relational databases. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 670–681. VLDB Endowment, 2002.

[46] Vagelis Hristidis, Louiqa Raschid, and Yao Wu. Scalable link-based personalization for ranking in entity-relationship graphs. In *WebDB*, 2011.

[47] Jian Hu, Gang Wang, Fred Lochovsky, Jian-tao Sun, and Zheng Chen. Understanding user's query intent with wikipedia. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 471–480, New York, NY, USA, 2009. ACM.

[48] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. *The VLDB Journal*, 13(3):207–221, 2004.

[49] Alpa Jain, AnHai Doan, and Luis Gravano. Optimizing sql queries over text databases. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 636–645, Washington, DC, USA, 2008. IEEE Computer Society.

[50] Jay J. Jiang and David W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Intl. Conf. on Computational Linguistics*, 1997.

[51] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 505–516. VLDB Endowment, 2005.

[52] Jaap Kamps, Shlomo Geva, Andrew Trotman, Alan Woodley, and Marijn Koolen. Overview of the inex 2008 ad hoc track. *Advances in Focused Retrieval: 7th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2008, Dagstuhl Castle, Germany, December 15-18, 2008. Revised and Selected Papers*, pages 1–28, 2009.

[53] Eser Kandogan, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and Huaiyu Zhu. Avatar semantic search: a database approach to information retrieval. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 790–792, New York, NY, USA, 2006. ACM.

[54] G. Kasneci, F. M Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. NAGA: searching and ranking knowledge. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 953 –962, April 2008.

[55] Boris Katz, Sue Felshin, Deniz Yuret, Ali Ibrahim, Jimmy J. Lin, Gregory Marton, Alton Jerome McFarland, and Baris Temelkuran. Omnibase: Uniform access to heterogeneous data for question answering. In *NLDB*, pages 230–234, 2002.

[56] Boris Katz and Beth Levin. Exploiting lexical regularities in designing natural language systems. In *Proceedings of the 12th conference on Computational linguistics - Volume 1*, COLING '88, pages 316–323, Stroudsburg, PA, USA, 1988. Association for Computational Linguistics.

[57] Oleksandr Kolomiyets and Marie-Francine Moens. A survey on question answering technology from an information retrieval perspective. *Information Sciences*, In Press, Corrected Proof:–, 2011.

[58] Oren Kurland and Lillian Lee. Corpus structure, language models, and ad hoc information retrieval. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '04, pages 194–201, New York, NY, USA, 2004. ACM.

[59] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[60] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[61] Claudia Leacock and Martin Chodorow. Combining local context with wordnet similarity for word sense identification. In *WordNet: A Lexical Reference System and its Application*, 1998.

[62] Yuangui Lei, Victoria S. Uren, and Enrico Motta. SemSearch: a search engine for the semantic web. In *EKAW*, pages 238–245, 2006.

[63] Douglas B. Lenat, R. V. Guha, Karen Pittman, Dexter Pratt, and Mary Shepherd. Cyc: toward programs with common sense. *Commun. ACM*, 33(8):30–49, August 1990.

[64] Xiao Li. Understanding the semantic structure of noun phrase queries. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1337–1345, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[65] Percy Liang, Michael I. Jordan, and Dan Klein. Learning Dependency-Based compositional semantics. In *ACL*, pages 590–599, 2011.

[66] Dekang Lin. An information-theoretic definition of similarity. In *ICML '98: Proc. of the Fifteenth Intl. Conf. on Machine Learning*, pages 296–304. Morgan Kaufmann Publishers Inc., 1998.

[67] Fang Liu, Clement Yu, and Weiyi Meng. Personalized web search by mapping user queries to categories. In *Proceedings of the eleventh international conference on Information and knowledge management*, CIKM '02, pages 558–565, New York, NY, USA, 2002. ACM.

[68] Shuang Liu, Clement Yu, and Weiyi Meng. Word sense disambiguation in queries. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, CIKM '05, pages 525–532, New York, NY, USA, 2005. ACM.

[69] V. Lopez, M. Fernndez, E. Motta, and N. Stieler. PowerAqua: supporting users in querying and exploring the semantic web content. *Semantic Web Journal*, 2011.

[70] Vanessa Lopez, Victoria S. Uren, Marta Sabou, and Enrico Motta. Is question answering fit for the semantic web?: A survey. *Semantic Web*, 2(2):125–155, 2011.

[71] Yi Luo, Xuemin Lin, Wei Wang, and Xiaofang Zhou. Spark: top-k keyword query in relational databases. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 115–126, New York, NY, USA, 2007. ACM.

[72] Ana G. Maguitman, Filippo Menczer, Heather Roinestad, and Alessandro Vespignani. Algorithmic detection of semantic similarity. In *WWW '05: Proc. of the 14th Intl. Conf. on World Wide Web*, pages 107–116. ACM, 2005.

[73] C.D. Manning, P. Raghavan, and H. Schütze. *An Introduction to Information Retrieval*. Cambridge University Press, 2008.

[74] Mehdi Manshadi and Xiao Li. Semantic tagging of web search queries. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, pages 861–869, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[75] J. Mayfield and T. Finin. Information retrieval on the semantic web: Integrating inference and retrieval. In *Workshop on the Semantic Web at the 26th International SIGIR Conference on Research and Development in Information Retrieval*, 2003.

[76] George A. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38:39–41, 1995.

[77] Kotaro Nakayama, Takahiro Hara, and Shojiro Nishio. Wikipedia mining for an association web thesaurus construction. In Boualem Benatallah, Fabio Casati, Dimitrios Georgakopoulos, Claudio Bartolini, Wasim Sadiq, and Claude Godart, editors, *Web Information Systems Engineering WISE 2007*, volume 4831 of *Lecture Notes in Computer Science*, pages 322–334. Springer Berlin Heidelberg, 2007.

[78] Eyal Oren, Renaud Delbru, Michele Catasta, Richard Cyganiak, Holger Stenzhorn, and Giovanni Tummarello. Sindice.com: a document-oriented lookup index for open linked data. *Int. J. Metadata Semant. Ontologies*, 3(1):37–52, November 2008.

[79] Stelios Paparizos, Alexandros Ntoulas, John Shafer, and Rakesh Agrawal. Answering web queries using structured data sources. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pages 1127–1130, New York, NY, USA, 2009. ACM.

[80] Jeffrey Pound, Alexander K. Hudek, Ihab F. Ilyas, and Grant Weddell. Interpreting keyword queries over web knowledge bases. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12, pages 305–314, New York, NY, USA, 2012. ACM.

[81] Jeffrey Pound, Ihab F Ilyas, and Grant Weddell. Expressive and flexible access to web-extracted data: a keyword-based structured query language. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 423–434, New York, NY, USA, 2010. ACM.

[82] Jeffrey Pound, Ihab F Ilyas, and Grant Weddell. QUICK: expressive and flexible search over knowledge bases and text collections. *Proc. VLDB Endow.*, 3(1-2):1573–1576, September 2010.

[83] Jeffrey Pound, Peter Mika, and Hugo Zaragoza. Ad-hoc object retrieval in the web of data. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 771–780, New York, NY, USA, 2010. ACM.

[84] Jeffrey Pound, Stelios Paparizos, and Panayiotis Tsaparas. Facet discovery for structured web search: a query-log mining approach. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 169–180, New York, NY, USA, 2011. ACM.

[85] Jeffrey Pound, David Toman, Grant E. Weddell, and Jiewen Wu. An assertion retrieval algebra for object queries over knowledge bases. In *IJCAI*, pages 1051–1056, 2011.

[86] A. Pretschner and S. Gauch. Ontology based personalized search. In *Tools with Artificial Intelligence, 1999. Proceedings. 11th IEEE International Conference on*, pages 391–398, 1999.

[87] Eric Prud'hommeaux and Andy Seaborne. Sparql query language for rdf. W3c recommendation, W3C, January 2008.

[88] Philip Resnik. Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language. *Journal of Artificial Intelligence Research*, 11:95–130, 1999.

[89] Cristiano Rocha, Daniel Schwabe, and Marcus P. Aragao. A hybrid approach for searching in the semantic web. In *Proceedings of the 13th international conference on World Wide Web*, pages 374–383. ACM Press, 2004.

[90] M. Andrea Rodríguez and Max J. Egenhofer. Determining Semantic Similarity among Entity Classes from Different Ontologies. *IEEE Trans. on Knowledge and Data Engineering.*, 15(2):442–456, 2003.

[91] Nikos Sarkas, Stelios Paparizos, and Panayiotis Tsaparas. Structured annotations of web queries. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 771–782, New York, NY, USA, 2010. ACM.

[92] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 134–141, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[93] I. Soboroff, A.P. de Vries, and N. Craswell. Overview of the TREC 2006 Enterprise Track, page 32. Technical Report SP 500-272. TREC, NIST Special Publication, 2006.

[94] Ian Soboroff. Overview of the trec 2004 novelty track. In *The Thirteenth Text Retrieval Conference (TREC 2004)*, 2004.

[95] M. Speretta and S. Gauch. Personalized search based on user search histories. In *Web Intelligence, 2005. Proceedings. The 2005 IEEE/WIC/ACM International Conference on*, pages 622–628, 2005.

[96] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A core of semantic knowledge - unifying WordNet and wikipedia. In *16th Intl. World Wide Web Conference (WWW 2007)*, pages 697–706, 2007.

[97] Sandeep Tata and Guy M. Lohman. Sqak: doing more with keywords. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 889–902, New York, NY, USA, 2008. ACM.

[98] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 405–416, 2009.

[99] Thanh Tran, Philipp Cimiano, Sebastian Rudolph, and Rudi Studer. Ontology-Based interpretation of keywords for semantic search. In *The Semantic Web*, volume 4825 of *Lecture Notes in Computer Science*, pages 523–536. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-76298-0_38.

[100] Thanh Tran, Haofen Wang, and Peter Haase. SearchWebDB: Data Web Search on a Pay-As-You-Go Integration Infrastructure. Technical report, Universitat Karlsruhe (TH), 2008.

[101] AJ Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In *IEEE Transactions on Information Theory*, volume 13 (2), pages 260–269, 1961.

[102] Haofen Wang, Thanh Tran, and Chang Liu. Ce2: towards a large scale hybrid search engine with integrated ranking support. In *Proceedings of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 1323–1324, New York, NY, USA, 2008. ACM.

[103] Ye-Yi Wang, Raphael Hoffmann, Xiao Li, and Jakub Szymanski. Semi-supervised learning of semantic classes for query understanding: from the web and for the web. In *Proceeding of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 37–46, New York, NY, USA, 2009. ACM.

[104] Stuart N. Wrigley, Raúl García-Castro, and Lyndon Nixon. Semantic evaluation at large scale (seals). In *Proceedings of the 21st international conference companion on World Wide Web*, WWW '12 Companion, pages 299–302, New York, NY, USA, 2012. ACM.

[105] Zhibiao Wu and Martha Palmer. Verb semantics and lexical selection. In *32nd. Annual Meeting of the Association for Computational Linguistics*, pages 133 –138, 1994.

[106] Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, Maya Ramanath, Volker Tresp, and Gerhard Weikum. Natural language questions for the web of data. In *EMNLP-CoNLL*, pages 379–390, 2012.

[107] Luke S. Zettlemoyer and Michael Collins. Online learning of relaxed CCG grammars for parsing to logical form. In *EMNLP-CoNLL*, pages 678–687, 2007.

[108] Lei Zhang, QiaoLing Liu, Jie Zhang, HaoFen Wang, Yue Pan, and Yong Yu. Semplore: An IR approach to scalable hybrid query of semantic web data. In *The Semantic Web*, volume 4825 of *Lecture Notes in Computer Science*, pages 652–665. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-76298-0_47.

[109] Xian Zhang, Yu Hao, Xiaoyan Zhu, and Ming Li. New information distance measure and its application in question answering system. *J. Comput. Sci. Technol.*, 23(4):557–572, 2008.

[110] Qi Zhou, Chong Wang, Miao Xiong, Haofen Wang, and Yong Yu. SPARK: adapting keyword query to semantic search. In *The Semantic Web*, volume 4825 of *Lecture Notes in Computer Science*, pages 694–707. Springer Berlin / Heidelberg, 2007.

[111] Berkeley DB. `http://www.oracle.com/technetwork/products/berkeleydb/`.

[112] CRF++: Yet Another CRF Toolkit. `http://code.google.com/p/crfpp/`.

[113] Freebase. `http://www.freebase.com/`.

[114] The Gene Ontology. `http://www.geneontology.org/`.

[115] Apache HBase. `http://hbase.apache.org/`.

[116] Linked Data. `linkeddata.org`.

[117] Apache Lucene. `http://lucene.apache.org/`.

[118] MG4J: Managing gigabytes for java. `http://mg4j.dsi.unimi.it/`.

[119] MusicBrainz: The Open Music Encyclopedia. `http://musicbrainz.org/`.

[120] OWL 2 Web Ontology Language Document Overview (Second Edition). `http://www.w3.org/TR/owl2-overview/`, December 2012.

[121] schema.org. `http://schema.org/`.

[122] SEALS - Semantic Evaluation at Large Scale. `http://www.seals-project.eu/`.

[123] Semantic search workshop 2010. `http://km.aifb.kit.edu/ws/semsearch10/`.

[124] Semantic search challenge 2011. `http://semsearch.yahoo.com/`.

[125] SNOMED Clinical Terms. `http://www.ihtsdo.org/snomed-ct/`.

[126] Stanford Natural Language Processing Software. `http://www-nlp.stanford.edu/software`.

[127] UMBEL. `http://www.umbel.org/`.

[128] Wikipedia, the free encyclopedia. `http://www.wikipedia.org/`.

[129] Yahoo! Academic Relations. `http://webscope.sandbox.yahoo.com/catalog.php` - L13 - Yahoo! Search Query Tiny Sample.

Appendix A

Evaluation Workloads

A.1 Query Understanding Workload

The following tables encode the positive queries from the query understanding workload. The negative queries from the Yahoo query log are subject to copyright and cannot be published. They are available as part of the WebScope program [129].

QID	Keyword Query	Semantic Summary	Structured Query Template
1	the cat in the hat	ent	ent
2	john lennon	ent	ent
3	electronic frontier foundation	ent	ent
4	once upon a time in mexico	ent	ent
5	merkel chancellor of germany	ent,type	ent $\sqcap$ type
6	clapton football club	ent,type	ent $\sqcap$ type
7	john smith poet	ent,type	ent $\sqcap$ type
8	apollo 13 movie	ent,type	ent $\sqcap$ type
9	barcelona artists	ent,type	type $\sqcap$ rel(ent)
10	chicago comedians	ent,type	type $\sqcap$ rel(ent)
11	turing award mathematicians	ent,type	type $\sqcap$ rel(ent)
12	emmy award actors	ent,type	type $\sqcap$ rel(ent)
13	toronto high park	ent,ent	ent1 $\sqcap$ rel(ent0)
14	madrid battle of somosierra	ent,ent	ent1 $\sqcap$ rel(ent0)
15	paris pulcinella	ent,ent	ent1 $\sqcap$ rel(ent0)
16	sony playstation	ent,ent	ent1 $\sqcap$ rel(ent0)
17	stephen harper conservative party of canada	ent,ent	ent0 $\sqcap$ rel(ent1)
18	guernica picasso	ent,ent	ent0 $\sqcap$ rel(ent1)
19	hard days night the beatles	ent,ent	ent0 $\sqcap$ rel(ent1)
20	from hell johnny depp	ent,ent	ent0 $\sqcap$ rel(ent1)
21	lead guitarists	type	type
22	spanish painters	type	type
23	coastal cities	type	type
24	companies based in silicon valley	type	type

Figure A.1: Query understanding workload: semantic summaries and structured query templates.

QID	Keyword Query	Semantic Summary	Structured Query Template
25	ibm revenue	ent,attr	attr(ent)
26	barack obama's birthday	ent,attr	attr(ent)
27	san diego zoo opening date	ent,attr	attr(ent)
28	facebook number of employees	ent,attr	attr(ent)
29	physicists that won the nobel physics prize	type,rel,ent	type $\sqcap$ rel(ent)
30	corporate execs born in sydney	type,rel,ent	type $\sqcap$ rel(ent)
31	guitarists awarded a grammy	type,rel,ent	type $\sqcap$ rel(ent)
32	companies established in 1972	type,rel,ent	type $\sqcap$ rel(ent)
33	john lennon's musical roles	ent,rel	rel(ent)
34	woody allen awards	ent,rel	rel(ent)
35	canada's capital	ent,rel	rel(ent)
36	richard feynmans advisor	ent,rel	rel(ent)
37	birthplace stephen hawking	rel,ent	rel(ent)
38	capital australia	rel,ent	rel(ent)
39	author of the waste lands	rel,ent	rel(ent)
40	director of pulp fiction	rel,ent	rel(ent)
41	protest song musicians	type,type	type1 $\sqcap$ rel(type0)
42	reggae rock trios	type,type	type1 $\sqcap$ rel(type0)
43	ivy league school US presidents	type,type	type1 $\sqcap$ rel(type0)
44	awarded guitarists	type,type	type1 $\sqcap$ rel(type0)
45	brewer park ottawa canada	ent,ent,ent	ent0 $\sqcap$ rel(ent1 $\sqcap$ rel(ent2))
46	recoleta cemetery buenos aires argentina	ent,ent,ent	ent0 $\sqcap$ rel(ent1 $\sqcap$ rel(ent2))
47	caveau huchette paris france	ent,ent,ent	ent0 $\sqcap$ rel(ent1 $\sqcap$ rel(ent2))
48	tiergarten berlin germany	ent,ent,ent	ent0 $\sqcap$ rel(ent1 $\sqcap$ rel(ent2))

QID	Keyword Query	Structured Keyword Query
1	the cat in the hat	the cat in the hat
2	john lennon	john lennon
3	electronic frontier foundation	electronic frontier foundation
4	once upon a time in mexico	once upon a time in mexico
5	merkel chancellor of germany	merkel $\sqcap$ chancellor of germany
6	clapton football club	clapton $\sqcap$ football club
7	john smith poet	john smith $\sqcap$ poet
8	apollo 13 movie	apollo 13 $\sqcap$ movie
9	barcelona artists	artists $\sqcap$ * (barcelona)
10	chicago comedians	comedians $\sqcap$ * (chicago)
11	turing award mathematicians	mathematicians $\sqcap$ * (turing award)
12	emmy award actors	actors $\sqcap$ * (emmy award)
13	toronto high park	high park $\sqcap$ * (toronto)
14	madrid battle of somosierra	battle of somosierra $\sqcap$ * (madrid)
15	paris pulcinella	pulcinella $\sqcap$ * (paris)
16	sony playstation	playstation $\sqcap$ * (sony)
17	stephen harper conservative party of canada	stephen harper $\sqcap$ * (conservative party of canada)
18	guernica picasso	guernica $\sqcap$ * (picasso)
19	hard days night the beatles	hard days night $\sqcap$ *(the beatles)
20	from hell johnny depp	from hell $\sqcap$ *(johnny depp)
21	lead guitarists	lead guitarists
22	spanish painters	spanish painters
23	coastal cities	coastal cities
24	companies based in silicon valley	companies based in silicon valley

Figure A.2: Query understanding workload: structured keyword query encodings.

QID	Keyword Query	Structured Keyword Query
25	ibm revenue	revenue (ibm)
26	barack obama's birthday	birthday (barack obama)
27	san diego zoo opening date	opening date (san diego zoo)
28	facebook number of employees	number of employees (facebook)
29	physicists that won the nobel physics prize	physicists ⊓ that won (the nobel physics prize)
30	corporate execs born in sydney	corporate execs ⊓ born in (sydney)
31	guitarists awarded a grammy	guitarists ⊓ awarded (a grammy)
32	companies established in 1972	companies ⊓ established in (1972)
33	john lennon's musical roles	musical roles (john lennon's)
34	woody allen awards	awards (woody allen)
35	canada's capital	capital (canada's)
36	richard feynmans advisor	richard feynmans advisor
37	birthplace stephen hawking	birthplace (stephen hawking)
38	capital australia	capital (australia)
39	author of the waste lands	author of (the waste lands)
40	director of pulp fiction	director of pulp fiction
41	protest song musicians	musicians ⊓ * (protest song)
42	reggae rock trios	rock trios ⊓ * (reggae)
43	ivy league school US presidents	US presidents ⊓ * (ivy league school)
44	awarded guitarists	guitarists ⊓ * (awarded)
45	brewer park ottawa canada	brewer park ⊓ * (ottawa ⊓ * (canada))
46	recoleta cemetery buenos aires argentina	recoleta cemetery ⊓ * (buenos aires ⊓ * (argentina))
47	caveau huchette paris france	caveau huchette ⊓ * (paris ⊓ * (france))
48	tiergarten berlin germany	tiergarten ⊓ * (berlin ⊓ * (germany))

QID	Keyword Query	Concept Search Query
1	the cat in the hat	The_Cat_in_the_Hat
2	john lennon	John_Lennon
3	electronic frontier foundation	Electronic_Frontier_Foundation
4	once upon a time in mexico	Once_Upon_a_Time_in_Mexico
5	merkel chancellor of germany	Angela_Merkel ⊓ wikicategory_Chancellors_of_Germany
6	clapton football club	Clapton_F.C. ⊓ wikicategory_English_football_clubs
7	john smith poet	John_Gibson_Smith ⊓ wordnet_poet_110444194
8	apollo 13 movie	Apollo_13_(film) ⊓ wordnet_movie_106613686
9	barcelona artists	wordnet_artist_109812338 ⊓ ∃originatesFrom(Barcelona) UNION wordnet_artist_109812338 ⊓ ∃livesIn(Barcelona)
10	chicago comedians	wordnet_comedian_109940146 ⊓ ∃livesIn (Chicago) UNION wordnet_comedian_109940146 ⊓ ∃bornIn (Chicago)
11	turing award mathematicians	wordnet_mathematician_110301261 ⊓ ∃hasWonPrize(Turing_Award)
12	emmy award actors	wordnet_actor_109765278 ⊓ ∃hasWonPrize(Emmy_Award)
13	toronto high park	High_Park ⊓ locatedIn(Toronto)
14	madrid battle of somosierra	Battle_of_Somosierra ⊓ ∃happenedIn (Madrid)
15	paris pulcinella	Pulcinella_(ballet) ⊓ ∃happenedIn(Paris)
16	sony playstation	PlayStation ⊓ ∃hasProduct^{-}(Sony)
17	stephen harper conservative party of canada	Stephen_Harper ⊓ ∃isAffiliatedTo(Conservative_Party_of_Canada)
18	guernica picasso	Guernica_(painting) ⊓ ∃created^{-}(Pablo_Picasso)
19	hard days night the beatles	A_Hard_Day's_Night_(song) ⊓ ∃created^{-}(The_Beatles)
20	from hell johnny depp	From_Hell_(film) ⊓ ∃actedIn^{-}(Johnny_Depp)
21	lead guitarists	wikicategory_Lead_guitarists
22	spanish painters	wikicategory_Spanish_painters
23	coastal cities	wikicategory_Coastal_cities
24	companies based in silicon valley	wikicategory_Companies_based_in_Silicon_Valley

Figure A.3: Query understanding workload: concept search query encodings.

QID	Keyword Query	Concept Search Query
25	ibm revenue	hasRevenue^{-}(IBM)
26	barack obama's birthday	bornOnDate^{-}(Barack_Obama)
27	san diego zoo opening date	establishedOnDate^{-}(San_Diego_Zoo)
28	facebook number of employees	hasNumberOfPeople^{-}(Facebook)
29	physicists that won the nobel physics prize	wordnet_physicist_110428004 ⊓ ∃hasWonPrize(Nobel_Prize_in_Physics)
30	corporate execs born in sydney	wordnet_corporate_executive_109966255 ⊓ ∃bornIn (Sydney)
31	guitarists awarded a grammy	wordnet_guitarist_110151760 ⊓ ∃hasWonPrize(Grammy_Award)
32	companies established in 1972	wordnet_company_108058098 ⊓ ∃establishedOnDate (1972-##-##)
33	john lennon's musical roles	∃musicalRole^{-}(John_Lennon)
34	woody allen awards	∃hasWonPrize^{-}(Woody_Allen)
35	canada's capital	∃hasCapital^{-}(Canada)
36	richard feynmans advisor	∃hasAcademicAdvisor^{-}(Richard_Feynman)
37	birthplace stephen hawking	∃bornIn^{-}(Stephen_Hawking)
38	capital australia	∃hasCapital^{-}(Australia)
39	author of the waste lands	∃wrote^{-}(The_Waste_Lands)
40	director of pulp fiction	∃directed^{-}(Pulp_Fiction_(film))
41	protest song musicians	wordnet_musician_110340312 ⊓ ∃created(wikicategory_Protest_songs)
42	reggae rock trios	wikicategory_Rock_trios ⊓ ∃isOfGenre(wordnet_reggae_107066285)
43	ivy league school US presidents	wikicategory_Presidents_of_the_United_States ⊓ ∃graduatedFrom (wikicategory_Ivy_League_business_schools)
44	awarded guitarists	wordnet_guitarist_110151760 ⊓ ∃hasWonPrize(wordnet_award_106696483)
45	brewer park ottawa canada	Brewer_Park ⊓ ∃locatedIn(Ottawa ⊓ hasCapital^{-}(Canada))
46	recoleta cemetery buenos aires argentina	La_Recoleta_Cemetery ⊓ ∃locatedIn (Buenos_Aires ⊓ ∃hasCapital^{-}(Argentina))
47	caveau huchette paris france	Le_Caveau_de_la_Huchette ⊓ ∃locatedIn (Paris ⊓ ∃hasCapital^{-}(France))
48	tiergarten berlin germany	Tiergarten ⊓ ∃locatedIn(Berlin ⊓ ∃hasCapital^{-}(Germany))

A.2 TREC-QA Structured Keyword Query Encoded Workload

TREC ID	TREC target	KB Target
216	Paul Krugman	Paul_Krugman
217	Jay-Z	Jay-Z
218	Darrell Hammond	Darrell_Hammond
219	Iraqi defector Curveball	Curveball_(informant)
220	International Management Group (IMG)	IMG_(business)
221	U.S. Mint	United_States_Mint
222	3M	3M
223	Merrill Lynch & Co.	Merrill_Lynch
224	WWE	World_Wrestling_Entertainment
225	Sago Mine disaster	Sago_Mine_disaster
228	March Madness 2006	2006_NCAA_Men's_Division_I_Basketball_Tournament
230	AMT	Alternative_Minimum_Tax
231	USS Abraham Lincoln	USS_Abraham_Lincoln_(CVN-72)
232	Dulles Airport	Washington_Dulles_International_Airport
233	comic strip Blondie	Blondie_(comic_strip)
234	Irving Berlin	Irving_Berlin
235	Susan Butcher	Susan_Butcher
236	Boston Pops	Boston_Pops_Orchestra
237	Cunard Cruise Lines	Cunard_Line
238	2004 Baseball World Series	2004_World_Series
239	game show Jeopardy	Jeopardy!
241	Jasper Fforde	Jasper_Fforde

TREC ID	Structured Keyword Query
216	writes (new york times) ⊓ teach(princeton university) ⊓ received doctorate (mit) ⊓ academic specialty (economics) ⊓ won prize(john bates clark medal) ⊓ wrote(Pop Internationalism)
217	produced by (Def Jam Records) ⊓ real name (shawn carter) ⊓ planning to marry (Beyonce Knowles) ⊓ president of(Def Jam) ⊓ grew up (brooklyn) ⊓ album (reasonable doubt)
218	old (50) ⊓ graduated (UF) ⊓ regularly appears (Saturday Night Live) ⊓ appear (NBC) ⊓ featured on (Comedy Central Presents) ⊓ impersonated(bush)
219	defected year (1999) ⊓ scientist ⊓ real name(Rafid Ahmed Alwan) ⊓ employed (Germany's Federal Intelligence Service) ⊓ claims accepted by (George W. Bush) ⊓ lives (germany)
220	founder (mark mccormack) ⊓ founded in year (1960) ⊓ acquired by (Forstmann Little & Co.) ⊓ has board member (Theodore J. Forstmann) ⊓ head (mark steinberg) ⊓ represents (Vijay Singh) ⊓ represents (Tiger Woods)
221	headquartered (washington) ⊓ established (1792) ⊓ part of department (Treasury Department) ⊓ has director (henrietta holsman)
222	founded when (1902) ⊓ based (St. Paul ⊓ Minnesota) ⊓ has CEO(George Buckley) ⊓ CEO predecessor(Desi DeSimone) ⊓ stands for (Minnesota Mining and Manufacturing) ⊓ web address(http://www.3M.com) ⊓ manufactures product (Post-It notes)
223	has chief executive(Stanley O'Neal) ⊓ has prime business (investment bank) ⊓ headquartered (new york) ⊓ headquarters street (Wall Street) ⊓ other name known as (Princeton Portfolio Research and Management)
224	chairman (Vince McMahon) ⊓ chief executive (Linda McMahon) ⊓ headquartered (Stamford ⊓ Connecticut) ⊓ short form (WWE) ⊓ evolved from (WWF) ⊓ airs on (Spike TV) ⊓ had wrestler appear (Chyna)
225	occurred on date (January 2 ⊓ 2006) ⊓ survivor (Randal McCloy Jr.) ⊓ miners died (12) ⊓ state (West Virginia) ⊓ investigated by (MSHA) ⊓ victim (Terry Helms)
228	lasts (19 days) ⊓ has slang expression (Big Dance) ⊓ concluding night(April 3) ⊓ described by (Final Four) ⊓ teams (64) ⊓ coach (Billy Donovan)
230	has expansion (Alternative Minimum Tax) ⊓ year added (1969) ⊓ exemption amount ($33,750)
231	abbreviated ship designation (CVN-72) ⊓ commissioned on date (November 11 ⊓ 1989) ⊓ ported where (Everett ⊓ Washington) ⊓ has commander (Captain Kendall Card) ⊓ aircraft (Sea Hawk helicopter)
232	airport code (IAD) ⊓ approved designation (Washington Dulles International Airport) ⊓ state located (Virginia) ⊓ named after (John Foster Dulles) ⊓ run by authority (Metropolitan Washington Airports Authority) ⊓ used by airline (Delta Airlines)

233	has creator (Chic Young) ⊓ syndicated by (King Features Syndicate) ⊓ has character (Dagwood)
234	born (Russia) ⊓ first big hit(Alexander's Ragtime Band) ⊓ composed (God Bless America) ⊓ died (September 22 ⊓ 1989) ⊓ died(heart attack)
235	born year (1954) ⊓ won race (Iditarod Race) ⊓ lives (alaska)
236	conductor (keith lockhart) ⊓ previous conductor (Arthur Fiedler) ⊓ manager (Tony Beadle) ⊓ concert venue (Symphony Hall)
237	owned by (Carnival Cruise Lines) ⊓ based in city (Miami) ⊓ president (Carol Marlow) ⊓ largest ship (Queen Mary 2) ⊓ has ship (Queen Elizabeth 2)
238	won by (Boston Red Sox) ⊓ was defeated (St. Louis Cardinals) ⊓ has star (Curt Schilling)
239	aired for first time (March 30 ⊓ 1964) ⊓ first host (Art Fleming) ⊓ current host(Alex Trebek) ⊓ had contestant (Ken Jennings)
241	first book (The Eyre Affair) ⊓ wrote (Lost in a good book)

TREC ID	Concept Search Query (Raw KB)
216	∅
217	∃isMarriedTo(Beyonc_Knowles) ⊓ ∃created (Reasonable_Doubt)
218	∅
219	isCalled ("Rafid Ahmed Alwan")
220	∅
221	establishedOnDate ("1792-##-##")
222	establishedOnDate ("1902-##-##") ⊓ hasWebsite ("http://www.3m.com")
223	∃hasProduct (Investment_management)
224	isCalled ("WWE")
225	∅
228	∅
230	∅
231	isCalled ("USS Abraham Lincoln (CVN-72)")
232	isCalled ("Washington Dulles International Airport")
233	∅
234	∃created (God_Bless_America) ⊓ diedOnDate ("1989-09-22")
235	bornOnDate ("1954-##-##")
236	∅
237	∅
238	∅
239	∅
241	∃wrote (The_Eyre_Affair) ⊓ ∃wrote (Lost_in_a_Good_Book)

TREC ID	Concept Search Query (Extended KB)
216	∃worksAt (The_New_York_Times) ⊓ ∃worksAt(Princeton_University) ⊓ ∃graduatedFrom (Massachusetts_Institute_of_Technology) ⊓ ∃academicSpecialty (Economics) ⊓ ∃hasWonPrize(John_Bates_Clark_Medal) ⊓ ∃wrote(Pop_Internationalism)
217	∃producedBy (Def_Jam_Recordings) ⊓ isCalled (“Shawn Carter”) ⊓ ∃isMarriedTo (Beyonc_Knowles) ⊓ ∃worksAt(Def_Jam_Recordings) ⊓ ∃livesIn (Brooklyn) ⊓ ∃created (Reasonable_Doubt)
218	age (“50”) ⊓ ∃graduatedFrom (University_of_Florida) ⊓ ∃worksAt (Saturday_Night_Live) ⊓ ∃worksAt (NBC) ⊓ ∃worksAt (Comedy_Central_Presents) ⊓ ∃impersonated(George_W._Bush)
219	defectedInYear (“1999-##-##”) ⊓ wordnet_scientist_110560637 ⊓ isCalled (”Rafid Ahmed Alwan”) ⊓ ∃worksAt (Bundesnachrichtendienst) ⊓ ∃claimsAcceptedBy (George_W._Bush) ⊓ ∃livesIn (Germany)
220	∃founder (Mark_McCormack) ⊓ establishedOnDate (“1960-##-##”) ⊓ ∃acquiredBy (Forstmann_Little_&_Company) ⊓ ∃hasBoardMember (Theodore_J._Forstmann) ⊓ ∃hasCEO (Mark_Steinberg) ⊓ ∃hasClient (Vijay_Singh) ⊓ ∃hasClient (Tiger_Woods)
221	∃locatedIn (Washington ⊓ _D.C.) ⊓ establishedOnDate (“1792-##-##”) ⊓ ∃isMemberOf (United_States_Department_of_the_Treasury) ⊓ ∃hasDirector (Henrietta_Holsman)
222	establishedOnDate (“1902-##-##”) ⊓ ∃locatedIn (Saint_Paul\,_Minnesota) ⊓ ∃hasCEO(George_Buckley) ⊓ ∃hasCEO(Desi_DeSimone) ⊓ isCalled (”Minnesota Mining and Manufacturing”) ⊓ hasWebsite(“http://www.3m.com”) ⊓ ∃hasProduct (Post-it_note)
223	∃hasCEO(Stanley_O’Neal) ⊓ ∃hasProduct (Investment_management) ⊓ ∃locatedIn (New_York) ⊓ ∃locatedIn (Wall_Street) ⊓ isCalled (“Princeton Portfolio Research and Management”)
224	∃hasChairman (Vince_McMahon) ⊓ ∃hasCEO (Linda_McMahon) ⊓ ∃locatedIn (Stamford\,_Connecticut) ⊓ isCalled (“WWE”) ⊓ isCalled (“WWF”) ⊓ ∃isAffiliatedTo (Spike_\(TV_channel\)) ⊓ ∃hasEmployee (Chyna)
225	happendOnDate (“2006-01-02”) ⊓ ∃hadSurvivor (Randal_McCloy) ⊓ deaths (“12”) ⊓ ∃locatedIn (West_Virginia) ⊓ ∃hadInvestigator (Mine_Safety_and_Health_Administration) ⊓ ∃hadVictim (Terry_Helms)
228	hasDuration (“19 days”) ⊓ isCalled (“Big Dance”) ⊓ endsOn (“2006-04-03”) ⊓ isCalled (“Final Four”) ⊓ teams (“64”) ⊓ ∃hasCoach (Billy_Donovan)
230	isCalled (“Alternative Minimum Tax”) ⊓ establishedOnDate (“1969-##-##”) ⊓ hasValue (“33750#dollar”)

231	isCalled (“USS Abraham Lincoln $CVN - 72$”) ⊓ establishedOnDate (“1989-11-11”) ⊓ ∃locatedIn (Everett ⊓ _Washington) ⊓ ∃hasCommander (Captain_Kendall_Card) ⊓ ∃supports (SH-60_Seahawk)
232	isCalled (“IAD”) ⊓ isCalled (“Washington Dulles International Airport”) ⊓ ∃locatedIn (Virginia) ⊓ ∃namedAfter (John_Foster_Dulles) ⊓ ∃isAffiliatedTo (Metropolitan_Washington_Airports_Authority) ⊓ ∃hasClient (Delta_Air_Lines)
233	∃createdBy (Chic_Young) ⊓ ∃isAffiliatedTo (King_Features_Syndicate) ⊓ ∃hasCharacter (Dagwood_Bumstead)
234	∃bornIn (Russia) ⊓ ∃created (Alexander’s_Ragtime_Band) ⊓ ∃created (God_Bless_America) ⊓ diedOnDate (“1989-09-22”) ⊓ ∃diedOf (Myocardial_infarction)
235	bornOnDate (“1954-##-##”) ⊓ ∃hasWonPrize (Iditarod_Trail_Sled_Dog_Race) ⊓ ∃livesIn (Alaska)
236	∃hasConductor (Keith_Lockhart) ⊓ ∃hasConductor (Arthur_Fiedler) ⊓ ∃manager (Tony_Beadle) ⊓ ∃concertVenue (Symphony_Hall\\,_Boston)
237	∃hasOwner (Carnival_Corporation_&_PLC) ⊓ ∃locatedIn (Miami) ⊓ ∃hasPresident (Carol_Marlow) ⊓ ∃hasProduct (RMS_Queen_Mary_2) ⊓ ∃hasProduct (RMS_Queen_Elizabeth_2)
238	∃wonBy (Boston_Red_Sox) ⊓ ∃lostBy (St._Louis_Cardinals) ⊓ ∃hasStar (Curt_Schilling)
239	establishedOnDate (“1964-03-30”) ⊓ ∃hasHost (Art_Fleming) ⊓ ∃hasHost (Alex_Trebek) ⊓ ∃hadParticipant (Ken_Jennings)
241	∃wrote (The_Eyre_Affair) ⊓ ∃wrote (Lost_in_a_Good_Book)

A.3 Synthetic Efficiency Workload

ID	Concept Search Query
1	wikicategory_German_immigrants_to_Switzerland
2	wordnet_broadcast_journalist_109875979
3	wikicategory_New_Age_musicians
4	wikicategory_United_States_Navy_officers
5	wordnet_painter_110391653
6	wordnet_village_108672738
7	wordnet_war_correspondent_110766718 ⊓ wikicategory_Royal_Green_Jackets_officers
8	wikicategory_Roman_emperors ⊓ wikicategory_Ancient_Olympic_competitors
9	wikicategory_American_jazz_pianists ⊓ wikicategory_Jewish_American_musicians
10	wordnet_biologist_109855630 ⊓ wordnet_novelist_110363573
11	wordnet_writer_110794014 ⊓ wordnet_actor_109765278
12	wikicategory_Double-named_places ⊓ wikicategory_Settlements_established_in_1836 ⊓ wikicategory_County_seats_in_Washington
13	wikicategory_Johns_Hopkins_University_alumni ⊓ wikicategory_20th_century_philosophers ⊓ wikicategory_American_philosophers
14	wordnet_botanist_109868270 ⊓ wordnet_physicist_110428004 ⊓ wordnet_biologist_109855630
15	wordnet_writer_110794014 ⊓ wordnet_actor_109765278 ⊓ wordnet_poet_110444194
16	wikicategory_American_television_reporters_and_correspondents ⊓ wordnet_broadcast_journalist_109875979 ⊓ wikicategory_60_Minutes_correspondents ⊓ wikicategory_Dalton_School_alumni
17	wikicategory_20th_century_philosophers ⊓ wikicategory_Academics_of_the_University_of_Manchester ⊓ wikicategory_Artificial_intelligence_researchers ⊓ wikicategory_Scientists_who_committed_suicide
18	wikicategory_American_film_directors ⊓ wikicategory_American_film_producers ⊓ wordnet_film_maker_110088390 ⊓ wordnet_manufacturer_110292316
19	wordnet_actor_109765278 ⊓ wordnet_poet_110444194 ⊓ wordnet_director_110014939 ⊓ wordnet_professional_110480253
20	wikicategory_Languages_of_Australia ⊓ wikicategory_Languages_of_Bangladesh ⊓ wikicategory_Languages_of_Botswana ⊓ wikicategory_Languages_of_Fiji ⊓ wikicategory_Languages_of_Ghana ⊓ wikicategory_Languages_of_Guyana

21	wikicategory_American_baritones ⊓ wikicategory_American_B-movie_actors ⊓ wikicategory_American_gospel_singers ⊓ wikicategory_Blues_musicians_from_Mississippi ⊓ wikicategory_Tennessee_actors ⊓ wikicategory_Tennessee_musicians
22	wordnet_actress_109767700 ⊓ wikicategory_American_film_directors ⊓ wikicategory_American_film_producers ⊓ wikicategory_American_screenwriters ⊓ wikicategory_American_essayists ⊓ wikicategory_American_novelists
23	wikicategory_American_film_actors ⊓ wordnet_singer_110599806 ⊓ wordnet_actor_109765278 ⊓ wikicategory_American_film_actors ⊓ wikicategory_American_male_singers ⊓ wordnet_soldier_110622053
24	wikicategory_Languages_of_The_Gambia ⊓ wikicategory_Languages_of_Australia ⊓ wikicategory_Languages_of_Bangladesh ⊓ wikicategory_Languages_of_Botswana ⊓ wikicategory_Languages_of_Fiji ⊓ wikicategory_Languages_of_Ghana ⊓ wikicategory_Languages_of_Guyana ⊓ wikicategory_Languages_of_Hong_Kong
25	wikicategory_Jazz_bandleaders ⊓ wikicategory_New_Orleans_jazz_musicians ⊓ wikicategory_American_jazz_singers ⊓ wikicategory_American_jazz_trumpeters ⊓ wikicategory_Swing_trumpeters ⊓ wikicategory_Swing_bandleaders ⊓ wikicategory_African_American_brass_musicians ⊓ wikicategory_Decca_artists
26	wordnet_philosopher_110423589 ⊓ wikicategory_20th_century_philosophers ⊓ wikicategory_American_political_writers ⊓ wikicategory_Jewish_scientists ⊓ wikicategory_Fellows_of_the_Royal_Society_of_Canada ⊓ wikicategory_Guggenheim_Fellows ⊓ wikicategory_Jewish_American_writers ⊓ wikicategory_American_atheists
27	wordnet_novelist_110363573 ⊓ wordnet_critic_109979321 ⊓ wordnet_composer_109947232 ⊓ wordnet_poet_110444194 ⊓ wordnet_dramatist_110030277 ⊓ wordnet_screenwriter_110564400 ⊓ wordnet_writer_110794014 ⊓ wordnet_presenter_110466387
28	∃hasWonPrize(Babe_Ruth_Award)
29	∃bornIn(Alberta)
30	∃bornIn(Moscow)
31	∃locatedIn(Texas)
32	∃isMarriedTo(wikicategory_American_film_actors)
33	∃locatedIn(wordnet_village_108672738)
34	∃happenedIn(∃locatedIn(Australia))
35	∃bornIn(∃locatedIn(Belgium))
36	∃bornIn(∃locatedIn(Czech_Republic))
37	∃locatedIn(∃hasOfficialLanguage(wordnet_language_106282651))
38	∃bornIn(∃locatedIn(wordnet_village_108672738))
39	∃means(∃happenedIn(∃locatedIn(Australia)))
40	∃isMarriedTo(∃bornIn(∃hasOfficialLanguage(French_language)))
41	∃means(∃locatedIn(∃locatedIn(Missouri)))
42	∃livesIn(∃locatedIn(∃hasOfficialLanguage(wordnet_language_106282651)))
43	∃livesIn(∃locatedIn(∃locatedIn(wordnet_village_108672738)))
44	∃means(∃happenedIn(locatedIn(Germany)))
45	∃familyNameOf(∃bornIn(∃locatedIn(∃hasOfficialLanguage(German_language))))
46	∃isMarriedTo(∃livesIn(∃locatedIn(∃hasOfficialLanguage(wordnet_language_106282651))))
47	∃givenNameOf(∃bornIn(∃locatedIn(∃locatedIn(wordnet_village_108672738))))
48	∃politicianOf(Ukraine) ⊓ ∃hasSuccessor(Leonid_Kuchma)
49	∃bornIn(Edmonton) ⊓ ∃hasPredecessor(Piers_McDonald)
50	∃isMarriedTo(wordnet_saxophonist_110554243) ⊓ ∃graduatedFrom(Yale_Law_School)

51	∃bornIn(wordnet_county_108546183) ⊓ ∃diedIn(wikicategory_Host_cities_of_the_Summer_Olympic_Games)
52	∃locatedIn(wordnet_city_108524735) ⊓ ∃locatedIn(wordnet_town_108665504)
53	∃created(Peggy_Sue) ⊓ ∃created(20_Golden_Greats) ⊓ ∃created(Words_of_Love)
54	∃hasWonPrize(Saturn_Award) ⊓ ∃actedIn(The_Adventures_of_Pluto_Nash) ⊓ ∃created(wikicategory_African_American_films)
55	∃hasWonPrize(wikicategory_Music_halls_of_fame) ⊓ ∃influences(wikicategory_American_rock_musicians) ⊓ ∃influences(wikicategory_American_stand-up_comedians)
56	∃directed(wikicategory_Short_films) ⊓ ∃created(wikicategory_2003_albums) ⊓ ∃produced(wikicategory_Comedy_films)
57	∃created(wordnet_company_108058098) ⊓ ∃bornIn(wordnet_city_108524735) ⊓ ∃directed(wikicategory_American_films)
58	∃participatedIn(Battle_of_Cut_Knife) ⊓ ∃hasOfficialLanguage(French_language_in_Canada) ⊓ ∃hasCapital(Ottawa) ⊓ ∃participatedIn(Upper_Canada_Rebellion)
59	∃hasWonPrize(Grammy_Lifetime_Achievement_Award) ⊓ ∃produced(Appalachian_Pride) ⊓ ∃created(Ride_This_Train) ⊓ ∃created(Strawberry_Cake)
60	∃hasWonPrize(wikicategory_Music_halls_of_fame) ⊓ ∃influences(wikicategory_American_rock_musicians) ⊓ ∃influences(wikicategory_American_stand-up_comedians) ⊓ ∃actedIn(wikicategory_Television_specials)
61	∃hasCapital(wordnet_capital_108518505) ⊓ ∃hasOfficialLanguage(wordnet_language_106282651) ⊓ ∃participatedIn(wordnet_military_action_100952963) ⊓ ∃participatedIn(wikicategory_Conflicts_in_1944)
62	∃musicalRole(wordnet_stringed_instrument_104338517) ⊓ ∃hasWonPrize(wordnet_award_106696483) ⊓ ∃influences(wordnet_musician_110340312) ⊓ ∃actedIn(wikicategory_English-language_films)

Appendix B

Proofs

B.1 Proof of Lemma 1

Proof: $(\leftarrow)$
Case 1:
This case is trivial as the assertion $A(e)$ is given explicitly as being part of $\mathcal{K}$.

Case 2:

1. $A_1(e) \in \mathcal{A}$	given.
2. $\mathcal{K} \models A_1(e)$	1. and Definition 2.
3. $A_1 \sqsubseteq A_2 \dots A_n \sqsubseteq A \subseteq \mathcal{O}$	given.
4. $\mathcal{K} \models A_1 \sqsubseteq A_2 \dots A_n \sqsubseteq A$	3 and Definition 2.
5. $\mathcal{K} \models A_1 \sqsubseteq A$	4. and transitivity of subsumption.
6. $\mathcal{K} \models A(e)$	2., 5., and Definition 2.

Proof: $(\rightarrow)$
Assume both case 1. and 2. are false for a proof by contradiction.

1. $\mathcal{K} \models A(e)$	given.
2. $A(e) \notin \mathcal{A}$	given by assumption, case 1. is false.
3. $\forall A'$ s.t. $A'(e) \in \mathcal{A}$:	
3.1 $A' \sqsubseteq A_2 \dots A_n \sqsubseteq A \not\subseteq \mathcal{O}$	3 and assumption case 2. is false.
3.2 $\mathcal{K} \models A'(e)$	3. and Definition 2.
3.3 $\mathcal{K} \not\models A' \sqsubseteq A$	3.1 and transitivity of subsumption.
3.4 $\mathcal{K} \not\models A(e)$	2, 3, 3.3.
4. Both case 1. and 2. are not false.	Contradiction 1. and 3.4.
5. $\mathcal{K} \models A(e)$	4.

B.2 Proof of Theorem 1

We consider a proof by structural induction on the concept search query C. We define the depth of the constructs $\{e\}, f(k)$, and A to be 0; the depth of $\exists R(C), \exists R^{-}(C)$ and $f^{-}(C)$ to be 1 plus the depth of C; and the depth of $C_1 \sqcap C_2$ to be 1 plus the larger of the depth of C_1 and C_2.

Inductive Base Cases:	
Case C = "$\{e\}$":	
$answer(\{e\}, \mathcal{K}) = \{e \mid \mathcal{K} \models \{e\}(e)\}$	Definition 4.
$= \{e \mid e \in (\{e\})^{\mathcal{I}}\}$	Definition 3.
$= \{e\}$	Definition 1.
$= eval(\{e\}, \mathcal{K}, index)$	definition of eval procedure.
Case C = "$f(k)$":	
$answer(f(k), \mathcal{K}) = \{e \mid \mathcal{K} \models (f(k))(e)\}$	Definition 4.
$= \{e \mid (e, k) \in (f(k))^{\mathcal{I}}\}$	Definition 3.
$= \{e \mid f(e, k) \in \mathcal{A}\}$	Definition 2.
$= index(\mathcal{K}, k, f^{-})$	Property 4.5.
$= eval(f(k), \mathcal{K}, index)$	definition of eval procedure.
Case C = "A":	
$answer(A, \mathcal{K}) = \{e \mid \mathcal{K} \models A(e)\}$	Definition 4.
$= \{e \mid A(e) \in \mathcal{A} \vee (A_1(e) \in \mathcal{A} \wedge A_1 \sqsubseteq A_2 \ldots A_n \sqsubseteq A \subseteq \mathcal{O})\}$	Lemma 1.
$= index(\mathcal{K}, A, type^{-}) \ \cup \ \{e \mid A_1(e) \in \mathcal{A} \wedge A_1 \sqsubseteq A_2 \ldots A_n \sqsubseteq A \subseteq \mathcal{O}\}$	Property 4.1.
$= index(\mathcal{K}, A, type^{-}) \ \cup$ $\{e \mid e \in index(\mathcal{K}, A_1, type^{-}) \wedge A_1 \sqsubseteq A_2 \ldots A_n \sqsubseteq A \subseteq \mathcal{O}\}$	Property 4.1.
$= index(\mathcal{K}, A, type^{-}) \ \cup$ $\{e \mid e \in index(\mathcal{K}, A_1, type^{-}) \wedge A_1 \in closure(\mathcal{K}, A)\}$	Property 4.7.
$= index(\mathcal{K}, A, type^{-}) \ \cup \ \forall_{A' \in closure(\mathcal{K}, A)} index(\mathcal{K}, A', type^{-})$	def of eval.

Inductive Hypothesis: Assume $eval(C, \mathcal{K}, index) = answer(C, \mathcal{K})$ for an arbitrary C of depth n.

Inductive Step: Consider the four cases in which a concept C has depth $n + 1$.

Case C = “$C_1 \sqcap C_2$”:

$answer(C_1 \sqcap C_2, \mathcal{K}) = \{e \mid \mathcal{K} \models (C_1 \sqcap C_2)(e)\}$	Definition 4.
$= \{e \mid e \in (C_1 \sqcap C_2)^{\mathcal{I}}\}$	Definition 3.
$= \{e \mid e \in (C_1)^{\mathcal{I}} \cap (C_2)^{\mathcal{I}}\}$	Definition 1.
$= \{e \mid e \in (C_1)^{\mathcal{I}} \wedge e \in (C_2)^{\mathcal{I}}\}$	definition of intersection.
$= \{e \mid \mathcal{K} \models C_1(e) \wedge \mathcal{K} \models C_2(e)\}$	Definition 3.
$= \{e \mid e \in answer(C_1, \mathcal{K}) \wedge e \in answer(C_2, \mathcal{K})\}$	Definition 4.
$= answer(C_1, \mathcal{K}) \cap answer(C_2, \mathcal{K})$	definition of intersection.
$= eval(C_1, \mathcal{K}, index) \cap eval(C_2, \mathcal{K}, index)$	Inductive Hypothesis.
$= eval(C_1 \sqcap C_2, \mathcal{K}, index)$	definition of eval procedure.

Case C = “$\exists R(C_1)$”:

$answer(\exists R(C_1), \mathcal{K}) = \{e \mid \mathcal{K} \models (\exists R(C_1))(e)\}$	Definition 4.
$= \{e \mid e \in (\exists R(C_1))^{\mathcal{I}}\}$	Definition 3.
$= \{e \mid \exists e' \in (C_1)^{\mathcal{I}} \wedge (e, e') \in (R)^{\mathcal{I}}\}$	Definition 1.
$= \{e \mid \exists e' \ \mathcal{K} \models C_1(e') \wedge (e, e') \in (R)^{\mathcal{I}}\}$	Definition 3.
$= \{e \mid \exists e' \in answer(C_1, \mathcal{K}) \wedge (e, e') \in (R)^{\mathcal{I}}\}$	Definition 4.
$= \{e \mid \exists e' \in eval(C_1, \mathcal{K}, index) \wedge (e, e') \in (R)^{\mathcal{I}}\}$	Inductive Hypothesis.
$= \{e \mid \exists e' \in eval(C_1, \mathcal{K}, index) \wedge R(e, e') \in \mathcal{A}\}$	Definition 2.
$= \{e \mid \exists e' \in eval(C_1, \mathcal{K}, index) \wedge e \in index(\mathcal{K}, e', R^-)\}$	Property 4.3.
$= eval(\exists R(C_1), \mathcal{K}, index)$	definition of eval procedure.

Case C = “$\exists R^-(C_1)$”:

$answer(\exists R^-(C_1), \mathcal{K}) = \{e \mid \mathcal{K} \models (\exists R^-(C_1))(e)\}$	Definition 4.
$= \{e \mid e \in (\exists R^-(C_1))^{\mathcal{I}}\}$	Definition 3.
$= \{e \mid \exists e' \in (C_1)^{\mathcal{I}} \wedge (e', e) \in (R)^{\mathcal{I}}\}$	Definition 1.
$= \{e \mid \exists e' \ \mathcal{K} \models C_1(e') \wedge (e', e) \in (R)^{\mathcal{I}}\}$	Definition 3.
$= \{e \mid \exists e' \in answer(C_1, \mathcal{K}) \wedge (e', e) \in (R)^{\mathcal{I}}\}$	Definition 4.
$= \{e \mid \exists e' \in eval(C_1, \mathcal{K}, index) \wedge (e', e) \in (R)^{\mathcal{I}}\}$	Inductive Hypothesis.
$= \{e \mid \exists e' \in eval(C_1, \mathcal{K}, index) \wedge R(e', e) \in \mathcal{A}\}$	Definition 2.
$= \{e \mid \exists e' \in eval(C_1, \mathcal{K}, index) \wedge e \in index(\mathcal{K}, e', R)\}$	Property 4.2.
$= eval(\exists R^-(C_1), \mathcal{K}, index)$	definition of eval procedure.

Case C = "$f^-(C_1)$":

$answer(f^-(C_1),\mathcal{K}) = \{k \mid \mathcal{K} \models (f^-(C_1))(k)\}$	Definition 4.
$= \{k \mid k \in (f^-(C_1))^{\mathcal{I}}\}$	Definition 3.
$= \{k \mid \exists e \in (C_1)^{\mathcal{I}} \wedge (e,k) \in (f)^{\mathcal{I}}\}$	Definition 1.
$= \{k \mid \exists e\ \mathcal{K} \models C_1(e) \wedge (e,k) \in (f)^{\mathcal{I}}\}$	Definition 3.
$= \{k \mid \exists e \in answer(C_1,\mathcal{K}) \wedge (e,k) \in (f)^{\mathcal{I}}\}$	Definition 4.
$= \{k \mid \exists e \in eval(C_1,\mathcal{K},index) \wedge (e,k) \in (f)^{\mathcal{I}}\}$	Inductive Hypothesis.
$= \{k \mid \exists e \in eval(C_1,\mathcal{K},index) \wedge f(e,k) \in \mathcal{A}\}$	Definition 2.
$= \{k \mid \exists e \in eval(C_1,\mathcal{K},index) \wedge k \in index(\mathcal{K},e,f)\}$	Property 4.4.
$= eval(f^-(C_1),\mathcal{K},index)$	definition of eval procedure.

By induction, $eval(C,\mathcal{K},index) = answer(C,\mathcal{K})$ for all C of depth ≥ 0.