Expressive and Flexible Access to Web-Extracted Data: A Keyword-based Structured Query Language

Jeffrey Pound
University of Waterloo
Waterloo, Canada
jpound@cs.uwaterloo.ca

Ihab F. Ilyas
University of Waterloo
Waterloo, Canada
ilyas@cs.uwaterloo.ca

Grant Weddell
University of Waterloo
Waterloo, Canada
gweddell@cs.uwaterloo.ca

ABSTRACT

Automated extraction of structured data from Web sources often leads to large heterogeneous knowledge bases (KB), with data and schema items numbering in the hundreds of thousands or millions. Formulating information needs with conventional structured query languages is difficult due to the sheer size of schema information available to the user. We address this challenge by proposing a new query language that blends keyword search with structured query processing over large information graphs with rich semantics. Our formalism for structured queries based on keywords combines the flexibility of keyword search with the expressiveness of structures queries.

We propose a solution to the resulting disambiguation problem caused by introducing keywords as primitives in a structured query language. We show how expressions in our proposed language can be rewritten using the vocabulary of the web-extracted KB, and how different possible rewritings can be ranked based on their syntactic relationship to the keywords in the query as well as their semantic coherence in the underlying KB.

An extensive experimental study demonstrates the efficiency and effectiveness of our approach. Additionally, we show how our query language fits into *QUICK*, an end-to-end information system that integrates web-extracted data graphs with full-text search. In this system, the rewritten query describes an arbitrary topic of interest for which corresponding entities, and documents relevant to the entities, are efficiently retrieved.

Categories and Subject Descriptors

H.2.3 [**Information Systems**]: Database Management—*Query languages*; H.3.3 [**Information Systems**]: Information Search and Retrieval—*Query formulation, Retrieval models*

General Terms

Design, Experimentation, Languages, Performance

1. INTRODUCTION

The World Wide Web hoards massive amounts of unstructured text data. Recent efforts to extract structured data sets from unstructured and semi-structured Web documents, such as ExDB [4], YAGO [24], and WebTables [3], have resulted in the creation of massive knowledge bases: structured data sets often encoding rich schematic information over millions of entities. As an example, both ExDB and YAGO 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. 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 people of German nationality who have won a Nobel award.* The user may pose the following conjunctive query to an information system.

$$q(x)\text{:- } \texttt{GERMAN\_PEOPLE}(x),\ \mathit{hasWonPrize}(x,y),\ \texttt{NOBEL\_PRIZE}(y) \quad (1)$$

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

$$\textit{“german has won nobel award”} \quad (2)$$

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* German, rather than those which contain the keyword *“german”*. Second, the keyword query will look for data items with any syntactic occurrence of *“nobel award”*, while the structured query uses this as a selection condition over the qualifying entities, exploiting the structure of the query to find qualifying results.

$$\overset{\text{“german”,}}{\left\{\begin{array}{c}\texttt{German\_Language}\\ \texttt{GERMAN\_PEOPLE}\\ \texttt{German\_Alphabet}\end{array}\right\}},\quad \overset{\text{“has won”}}{\left\{\begin{array}{c}\textit{hasWeight}\\ \textit{hasWonPrize}\end{array}\right\}}\;\overset{\text{(“nobel award”)}}{\left(\left\{\begin{array}{c}\texttt{NOBEL\_PRIZE}\\ \texttt{AWARD}\\ \texttt{Turing\_Award}\end{array}\right\}\right)}.$$

Figure 1: An ambiguous conjunctive query and the possible interpretations. The first and third predicates are an example of matching on keyword occurrence, while the second predicate is an example of matching on edit distance.

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, 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 or regular expressions to take the place of entities or relations. In this setting the query from the previous example may be written using structure as in (1), but with keywords as in (2).

“german, has won(nobel award)”

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 1. 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 size 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 `GERMAN_PEOPLE` who have won an `AWARD` (the super class of all awards) will produce many results that obstruct the user from finding those of interest (those who have won a `NOBEL_PRIZE`). Note that syntactic matching alone could not possibly be sufficient as a disambiguation solution in all cases, as the keyword *“german”* is a natural choice to express both a nationality and a language. This phenomena, known as *polysemy*, is just one of the challenges of disambiguation. Similar problems arise with synonymy.

In this paper, we investigate a solution to the problem of querying over rich massive 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.

1.1 Contributions

In this work, we make the following contributions.

- We propose a model for keyword-based structured queries that allows users to create expressive descriptions of their information need without having detailed knowledge of the underlying schema.
- We analyze the consequences of introducing ambiguity into a structured query language (by means of keywords as base level constructs), and propose a model for efficient and effective disambiguation.
- We show how to accommodate incomplete knowledge in the underlying KB by allowing query terms to be used as keywords during document retrieval, integrating full text search in the cases where full disambiguation can not be achieved.
- Lastly, we show how our proposed query processing model can be integrated into *QUICK* (Queries Using Inferred Concepts from Keywords) an end-to-end semantic search system.

We demonstrate the viability of our proposed approach with an extensive experimental evaluation of both the quality and performance of a prototype implementation of the complete system.

1.2 Outline

The remainder of the paper is organized as follows. Section 2 reviews relevant background information and gives an overview of our problem and proposed architecture. Section 3 presents our structured keyword query language, with the disambiguation problem explored in Section 4. In Section 5 we show experimental results of a prototype implementation. Section 6 surveys and compares related work with our approach and Section 7 concludes.

2. BACKGROUND

Entity Search Entity search is a variant of traditional information retrieval (IR) in which *entities* (e.g., Albert Einstein, Los Angeles, etc..) form the most basic data-level construct, as opposed to *documents* in traditional IR. This view of the search problem opens up a variety of new query processing opportunities, in particular the ability to process more structured types of queries over the entities and their

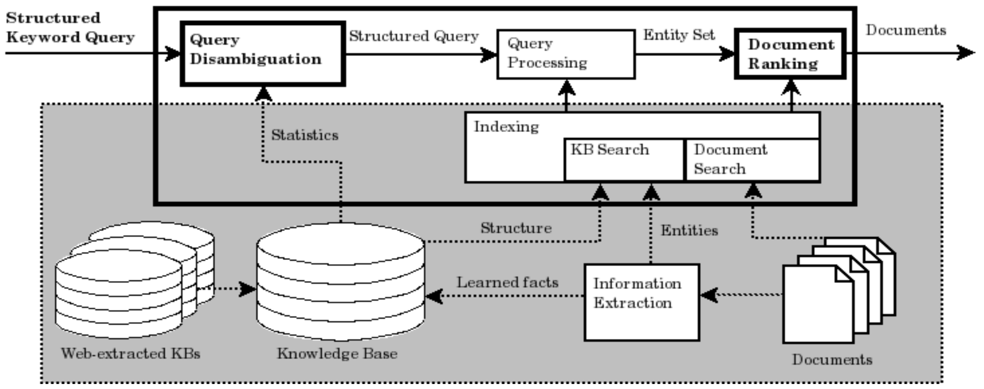

Figure 2: System architecture.

relations to one another becomes feasible (e.g., [4, 6, 13, 14]). In contrast to traditional IR, query processing in this model may aggregate data from multiple documents to determine which entities are part of a query result. In these models, structured or unstructured query languages can be used to describe entities of interest, and data can reside in a structured model or in text documents. The common facet is that entities form the primitives for search.

As an example, consider an entity-based search system over text documents. Given a query for documents containing the entity `Max_Planck`, the system would retrieve all documents mentioning `Max_Planck`, and *not* the documents which mention the `Max_Planck_Institute` or the `Max_Planck_Medal` (an award in physics). Having entities as primitives reduces ambiguity in the semantics of the search, which avoids problems common to traditional keyword search systems. Also, information can be aggregated from different sources to answer a query, such as a search for all documents containing entities *bornIn*(`Canada`). Documents about qualifying entities will be returned independent of the location of the information that encodes where they were born.

Semantic Search KB-based semantic search is a similar problem to entity search in which entities and relations form the base level data constructs. In these systems, rich semantics defined by KBs are taken into consideration during query processing (or as a preprocessing phase). However, semantic search systems generally operate over structured semantic data, such as RDF or OWL [10, 21], fact triples [17], or XML [9] (with explicit references to the ontology concepts). A variant of the semantic search problem is to support searching over the semantics of entities found in text documents. Again these models may make use of structured or unstructured query languages. The common thread in semantic search systems is the support for inference over a reference schema graph.

As an example, consider a semantic search system over text documents. A search for documents containing entities of type `GUITARIST` would return documents about `B._B._King`, `Jimi_Hendrix`, and other entities known to be guitarists. These documents are query results independent of whether or not they contain a syntactic occurrence of the word "guitarist." Thus, the search relates to the semantics of the entities, rather than the surface syntax.

Knowledge Representation The field of knowledge representation is concerned with formalisms for encoding knowledge with clear semantics to support inference over the encoded information. A collection of schematic information, which forms general rules about a domain of interest, is called an ontology. Concrete instance information, or entities, describe explicit information about particular entities in the domain, such as `Albert_Einstein`. A *knowledge base* is a general term for a domain ontology and instance data, e.g., the union of a data graph and schema graph in the case of web-extracted data.

As an example, an ontology might define relationships among abstract primitive concepts, such as the relationship `PHYSICIST` *is-a* `SCIENTIST`. Ontologies may also include arbitrary relations, such as *hasWonPrize*, to encode information like `NOBEL_LAUREATE` *is-a* `PERSON` that has a *hasWonPrize* relation to some `NOBEL_PRIZE`. The encoding of the knowledge base information and the associated semantics rest on an underlying knowledge representation formalism. More details on the underlying formalisms used in knowledge representation can be found in [1].

2.1 Preliminaries

We assume access to a domain related knowledge base and document corpus. These data sources are preprocessed to: (1) extract entities from the text (used for entity search over documents), (2) compute statistics over the knowledge base (used in disambiguation), and (3) index the knowledge base's content and structure (used in query processing), as well as the document text (used to integrate full text search).

The schema language for the underlying knowledge bases assumed in our work includes entities, primitive concepts, conjunctions, existential relations, and acyclic concept hierarchies. This is sufficient to capture many real world knowledge bases and ontologies such as YAGO [24], a large knowledge base derived from Wikipedia and WordNet.

The basic knowledge base terminology used in our discussion, as well as our notational conventions, are as follows.

- *Entities* are written in a typed font with leading capitals. Entities denote constant terms such as people, places, and organizations (e.g., `Albert_Einstein`, `Canada`).
- *Primitive Concepts* are written in a typed font in all capitals. Primitive concepts are types or classes to which entities may belong (e.g., `SCIENTIST`, `COUNTRY`).
- *Relations* are written in italic camel caps. Relations are binary relationships that can exist between enti-

ties and/or primitive concepts (e.g., *hasWonPrize* or *bornIn*). There is also a transitive *is-a* relation used to encode hierarchies among concepts.

- A *Concept* is any number of entities or primitive concepts joined by logical connectives such as conjunctions and relations (note that relations are implicitly existentially qualified). We use "," to denote a conjunction. (e.g. `SCIENTIST`, *hasWonPrize*(`Nobel_Prize_in_Physics`)). As a concept describes a specific set of entities, it can also be viewed as a query which we refer to as a *Concept Query*.

We also adopt the convention of writing *keywords* in quoted italic lower case (e.g., *"scientist"*).

2.2 System Overview

Our complete system architecture is illustrated in Figure 2. The solid lined box on top shows the run-time components, with data flow represented as solid lined arrows. The grey dotted lined box shows the preprocessing phase, with preprocessor data flow represented as dotted lined arrows. The bold typed components in the diagram show the parts of the system which we focus on in this paper, namely, the query language, query disambiguation, and integrating full text search for document ranking. Details of our structured query processor and KB index, which allow efficient evaluation of structured KB queries, are beyond the scope of this paper. We use existing solutions to support the remaining components of the system (knowledge base construction and information extraction).

The preprocessor consists of two components, the information extraction system and the indexing module. These two components make use of two domain related data sources, a knowledge base and a document corpus. The preprocessing phase proceeds as follows. First the information extraction tool is run over the document corpus to mine structured information from the text. This information includes extracted named entities as well as relationships among the entities. Facts that were previously unknown to the knowledge base can be contributed (note that we omit this learning phase in our implementation and discussion as the focus of this paper is not on the extraction component). The named entities are then sent to the indexer to build an inverted index over entities and documents. The full text of the documents is also indexed to support full text search.

In the second preprocessing phase, the structure of the knowledge base is indexed to support efficient entity search within the knowledge base. It is important to note that the entity search module must be capable of handling the expressivity of the constructs used in the underlying knowledge base. The final preprocessing step is to compute statistics over the knowledge base which will be used during query disambiguation.

At run time, an arbitrary keyword-based structured query is taken as input to the system. Because the query is based on keywords, there is an inherent ambiguity as to the semantics of the query (i.e., the user's intention). We compute a ranked set of possible query disambiguations based on the vocabulary of the knowledge base. Each disambiguated query is thus a general concept over the underlying knowledge base. One or more of these disambiguated queries can then be evaluated to find potentially relevant entities and their corresponding documents. The documents are then returned to the user ranked by some relevance metric.

3. STRUCTURED KEYWORD QUERIES

We now introduce our keyword-based structured query language.

Definition 1. **(Document Query)** A *document query DQ* is given by the following.

$$\begin{array}{rcl} DQ & ::= & Q \\ & | & (Q_1), (Q_2), ..., (Q_n) \end{array}$$

where Q_i is a structured keyword query.

A document query specifies documents of interest based on entities that qualify for the structured keyword queries using "AND" semantics at the document level. A structured keyword query describes a set of entities as defined below.

Definition 2. **(Structured Keyword Query)** Let k be a keyword phrase (one or more keywords), then a *structured keyword query* Q is defined by the following grammar.

$$\begin{array}{rcl} Q & ::= & k \\ & | & k(Q) \\ & | & Q_1, Q_2, ..., Q_n \end{array}$$

The first construct allows a primitive concept in a query to be described by a set of one or more keywords (e.g., "*nobel prize*"). 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(Germany)*"). The third construct allows a set of queries to describe a single class of entities in conjunction (e.g., "*harmonica player, songwriter*"). The comma operator is used to explicitly label the conjunctions, reducing the ambiguity among multi-keyword phrases that occur in conjunction.

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

More formally, the resulting entity set $\{e\}$ for a structured keyword query Q denoting general concept C is given by the following: $\{e \mid e \in C\}$. We use the notation $e \in Q$ to say that entity e is in the result set of query Q. Section 4 discusses how to find the concept C which is denoted by Q.

To illustrate how structured keyword queries are used in document queries, consider the following example document query.

(person, born in(Germany)), (participated(World War II))

The query indicates that qualifying documents should mention both a person born in Germany and some entity that participated in World War II. More formally, the resulting document set $\{D\}$ for a document query $DQ = (Q_1), (Q_2), ...,$ (Q_n) is given by the following:

$$\{D \mid \forall_{Q \in DQ} \exists_{e \in Q} \text{ such that } e \in D\}$$

where $e \in D$ denotes entity e occurring in the text of document D. Intuitively, the document retrieval uses "OR" semantics per entity set denoted by a subquery, and "AND" semantics across subqueries in the document query.

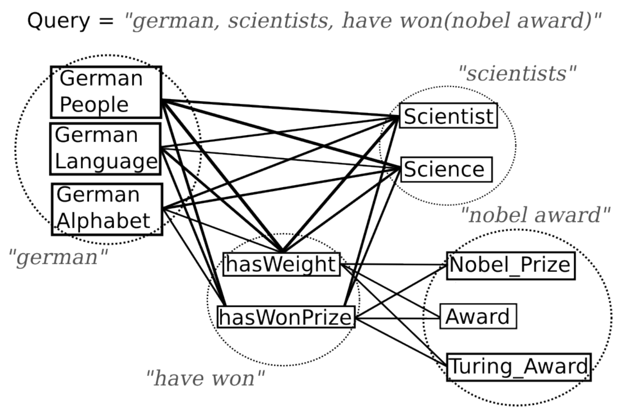

Figure 3: An example query disambiguation graph.

4. DISAMBIGUATING QUERY INTENT

Because our 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 which we could select an item for each keyword to create a general concept.

4.1 The Disambiguation Model

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

The goal is to choose one schema item for each keyword in the query, such that the resulting disambiguation is a meaningful representation of the query with respect to the knowledge base, and is also representative of the user's intention. There is a tradeoff that must be considered when choosing concept interpretations for each keyword: we want mappings that represent the users intentions 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 entity, concept, or relation to each keyword phrase. In the discussion that follows, we refer to a concept or the subgraph that denotes a concept interchangeably.

Definition 3. **(Disambiguation Graph)** Let Q 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 disjoint partitions of the nodes in V, are given by the following.

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

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

$$\begin{aligned} edges(k) &= \{\} \\ edges(k(Q)) &= \left(\bigcup_{\substack{n_1\in M(k),\\ n_2\in root(Q)}} \langle n_1, n_2\rangle \right) \cup\ edges(Q) \\ edges(Q_1, Q_2) &= \left(\bigcup_{\substack{n_1\in root(Q_1),\\ n_2\in root(Q_2)}} \langle n_1, n_2\rangle \right) \cup\ edges(Q_1) \\ &\qquad \cup\ edges(Q_2) \end{aligned}$$

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

$$\begin{aligned} root(k) &= M(k) \\ root(k(Q)) &= M(k) \\ root(Q_1, Q_2) &= root(Q_1) \cup root(Q_2) \end{aligned}$$

(Note that we have abstracted n-ary conjunctions as binary conjunctions for ease of presentation.)

Figure 3 depicts an example of a disambiguation graph for the query *"german, scientists, have won(nobel award)"*. The boxes denote vertices (corresponding to concepts, entities, or relations in the underlying knowledge base), the dotted circles denote partitions (the sets M(k_i)), the italic font labels denote the keywords for which the partition 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 Q, any induced subgraph of G that spans all partitions in P corresponds to a concept interpretation of Q. For example, the induced subgraph spanning the nodes

$$\{\texttt{GERMAN\_PEOPLE}, \texttt{SCIENTIST}, \textit{hasWonPrize}, \texttt{NOBEL\_PRIZE}\}$$

corresponds to the concept

$$\texttt{GERMAN\_PEOPLE},\ \texttt{SCIENTIST},\ \textit{hasWonPrize}(\texttt{NOBEL\_PRIZE}).$$

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.

4.2 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 concepts A and B. 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

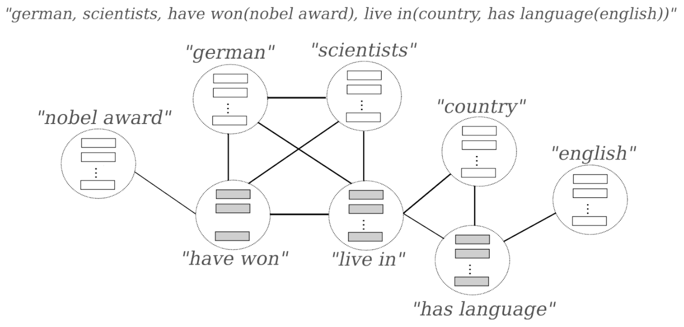
"german, scientists, have won(nobel award), live in(country, has language(english))"
"german"
"scientists"
"nobel award"
"country"
"english"
"have won"
"live in"
"has language"

Figure 4: An example of a query disambiguation graph. Each edge represents complete bipartite connectivity between all vertices in the partitions.

similarities over taxonomies, often including the graph distance of concepts [16, 18, 25], and probabilistic notions of information content of classes in the taxonomy [16, 19, 22]. More recent works have looked at generalizing semantic similarity metrics from taxonomies to general ontologies (i.e., from trees to graphs) [7, 20, 23]. As an example, Lin defines semantic similarity between two concepts as the following [19].

$$Sim_{Lin}(A,B) = \frac{2 \cdot log(P(LCS(A,B))}{log(P(A)) + log(P(B)))}$$

where $LCS(A,B)$ denotes the least common subsumer of A and B, i.e., the first common ancestor in an "is-a" graph, and $P(A)$ denotes the probability of a randomly drawn entity belonging to concept A.

Lastly, there are the traditional notions of set similarity which can be applied, since concepts in a taxonomy or ontology denote sets of entities. Two commonly used metrics are the Dice coefficient and the Jaccard index, the latter illustrated 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 to be introduced in Section 4.3.

Definition 4. **(Knowledge Base Support)** Let C be a general concept expressed using the primitive concepts, relations, and entities in the knowledge base KB. Then the *support* of C by KB is given by the following.

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

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

Intuitively, this is the similarity according to the structure of the disambiguation graph (which corresponds to the structure of the query).

Figure 4 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 disambiguation.

$$\texttt{GERMAN\_PEOPLE},\ \texttt{SCIENTIST},\ hasWonPrize(\texttt{NOBEL\_PRIZE}),$$
$$livesIn(\texttt{COUNTRY},\ hasOfficialLanguage(\texttt{English\_Language}))$$

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

$$C_1 = hasWonPrize(\texttt{NOBEL\_PRIZE})$$
$$C_2 = livesIn(\texttt{COUNTRY},\ C_3)$$
$$C_3 = hasOfficialLanguage(\texttt{English\_Language})$$

Now C can be expressed as the following.

$$\texttt{GERMAN\_PEOPLE}, \texttt{SCIENTIST}, C_1, C_2$$

The support of C corresponds to the following expression.

$$\begin{aligned} &support(C,KB) = semanticSim(\texttt{GERMAN\_PEOPLE}, \texttt{SCIENTIST}, C_1, C_2) \\ &+ semanticSim(hasWonPrize, \texttt{NOBEL\_PRIZE}) \\ &+ semanticSim(livesIn, \texttt{COUNTRY}, C_3) \\ &+ semanticSim(hasOfficialLanguage, \texttt{English\_Language}) \end{aligned}$$

Syntactic Similarity The second component to our scoring model is the syntactic matching of the concept 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 maximizing semantic similarity, the better our query interpretation will be with respect to both the user's intentions and the knowledge encoded in our knowledge base. We use the function label $syntaxSim(a,b)$ 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 distance.

Score Aggregation Our final scoring model is thus some combination of knowledge base support, which quantifies semantic coherence for the candidate disambiguation, 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 5. **(Concept Score)** Let Q be a structured keyword query, C a concept, $\oplus$ a binary aggregation function, and KB a knowledge base. Then the *score* of concept C, with respect to Q, KB, and $\oplus$ is given by the following.

$$score(C,Q,KB) = support(C,KB) \oplus syntaxSim(C,Q)$$

Given a disambiguation graph G with partition set P and a parameter integer k, the goal of disambiguation is to find the top-k maximum scoring subgraphs of G that span all partitions 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.

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 primitive concepts and relations). From the disambiguation graph point of view, this means that the score (or weight) of each edge *changes* depending on which other edges are part of the subgraph forming the candidate concept interpretation. The volume of statistics that must be (pre)computed in order to support queries with n terms would be very large. One would need to have access to the semantic similarity of all n-way compositions of entities, primitive concepts, and relations occurring in the knowledge base.

4.3 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. 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 primitive concepts or relations 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 KB and the keywords k from a structured keyword query Q.

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

where $\mathit{item}(v)$ denotes the knowledge base schema item (primitive concept, relation, or entity) represented by vertex v and $\mathit{label}(v)$ denotes the string representation of the schema item represented by v.[1]

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

Definition 6. **(Approximate Score)** Let Q be a structured keyword query, KB a knowledge base, $\oplus$ a binary aggregation function, and G a subgraph of the disambiguation graph of Q representing a concept C. Then the *approximate score* of C represented by G with respect to Q, KB, and $\oplus$ is given by the following.

$$\hat{\mathit{score}}(G, Q, KB) = \left(\sum_{\langle v_1, v_2\rangle \in E} w(\langle v_1, v_2\rangle) \ \oplus \ \sum_{v \in V} w(v) \right)$$

4.4 Solving the Disambiguation Problem

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. Note that simply taking the maximum scoring subgraph for each subquery will not necessarily result in a connected graph, and thus does not map to a concept interpretation of the query. If our scoring function is monotonic, we can implement a rank-join algorithm [12] 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.

[1]In practice, it would be beneficial to consider the syntactic similarity of the keyword to the closest synonym of the concept label.

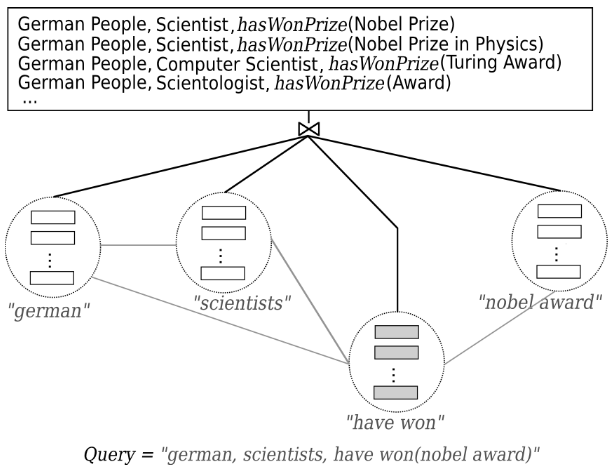
German People, Scientist, hasWonPrize(Nobel Prize)
German People, Scientist, hasWonPrize(Nobel Prize in Physics)
German People, Computer Scientist, hasWonPrize(Turing Award)
German People, Scientologist, hasWonPrize(Award)
...
"german"
"scientists"
"nobel award"
"have won"
Query = "german, scientists, have won(nobel award)"

Figure 5: Encoding disambiguation as a rank-join with random access to edges. The join conditions are on the existence of edges in the disambiguation graph for each subquery, and on the equality of relation bindings to join across all subqueries.

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 partition corresponding to the keyword. Each partition is ordered by syntactic similarity which corresponds to vertex weight in the disambiguation graph.

Consider each possible schema item match for a keyword phrase (partition $M(k)$), ordered by syntactic similarity, as an input. We "join" two concepts if they have 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.

As an example, consider the disambiguation graph from Figure 3, we illustrate a rank-join approach to disambiguation of the query in Figure 5. The illustration shows each partition as an input to the rank join. Conjunctive components are joined based on the existence of non-zero weight edges in the disambiguation graph, while the entire nested queries are joined based on equality of the binding of the relation keyword to a schema relation.

For brevity, we omit a review of the rank-join algorithm. The challenge here is in the modeling of the problem as a rank-join as we have presented. With this model in mind, the actual execution does not differ from that of a traditional rank-join. We refer the reader to [12] for an overview of the algorithm.

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 5).

4.5 Integrating Full Text Search

As a final consideration, it may be the case that no subgraph can be found that spans all partitions in a disambiguation graph for some query. This happens when we can not find a meaningful interpretation for one or more keywords. This may be due to a failure to generate good candidate partitions, or due to a lack of coverage in the underlying KB. In this case we proceed as follows:

- We relax the problem definition to not require spanning all partitions when finding a subgraph of the disambiguation graph. This allows us to build partial interpretations.
- We include all keywords that are unmapped as part of the document retrieval process. In this setting, we will retrieve documents that contain one or more entities from the concept interpretation of the query (as before), and favor those which contain the additional keywords in the ranking by re-ranking the document results.

For example, a partial interpretation for the query *"german, scientists, have won (nobel award)"* could be the concept `SCIENTIST`, *hasWonPrize*(`NOBEL_PRIZE`). All qualifying documents are then retrieved, and those which mention the keyword *"german"* have their score boosted by a some factor. The boosting factor could be either a constant factor, or a factor proportional to the relevance of the residual keyword to the document. We use the latter in our implementation by adding the document scores. The qualifying documents are then re-ordered based on the new scores. We can achieve this efficiently using the document index by doing a look-up of the residual keywords and then boosting documents from the entity search that occur in this set. Note that in the worst case (in terms of disambiguation), our system degrades to a regular keyword search.

4.6 On Query Results and Ranking

There are a number of ways one could envision presenting and ranking results in our system. One approach could have entities as results, each with a list of associated documents. The entities could be ranked by their relationships to the query in the KB, or based on statistics over the document corpus. Alternatively, one may want a document list as the result, ranked by the relevance of the corresponding entities to the document. A third option could combine these two approaches, ranking groups of documents per entity, and ranking the groups based on an overall ranking of entities.

While ranking is not the focus of this paper, it is a necessary component to a deployable system. We take the simple approach of ranking the resulting document set as a whole. Ranking is based on the tf-idf score of the corresponding entities that qualified the document as a result, with a re-ranking of documents based on residual keywords as described in Section 4.5.

5. EXPERIMENTAL EVALUATION

We have implemented the entire *QUICK* system as depicted in Figure 2, with the exception of the feedback mechanism for learning new facts from text. We treat the information extraction tool as a black box system, and only extract the named entities for indexing.

We have designed a number of tasks to evaluate the effectiveness and efficiency of our disambiguation procedure, and of our system as a whole. We directly evaluate the disambiguations produced by our algorithm against the ideal structured concept query which is constructed manually. We use an *entity search task* to evaluate the precision and recall of entities returned by our system as the number of query interpretations is varied. We also evaluate quality from an end-to-end system point of view using a *document retrieval task*. Lastly, we give an overview of the performance of both the disambiguation procedure and the system as a whole.

We compare our system to three baseline approaches, one involving hand-coded (i.e., disambiguated) concept queries, one using syntax-only disambiguations, and the last being adaptations of a traditional IR engine for the given tasks.

5.1 System Descriptions

QUICK is built on a variety of open-source software. For the information extraction component, we built a simple text processor using the OpenNLP library.[2] This tool will parse and annotate person, location, and organization named entities. We built a simple text and named entity indexer using the Lucene library.[3] Our document index supports entity-based retrieval by indexing entities separately from keywords. We also create a text index over all KB items and any synonyms encoded in YAGO. We use this index to generate the partitions of candidate items based on keyword occurrence. The concept query processor was written in Java and uses Berkeley DB[4] as a back-end data store. Our query disambiguation system was also written in Java and uses a separate Berkeley DB store to maintain the knowledge base semantic similarity measures. The pairwise semantic similarity values used for disambiguation are pre-computed. 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 distance for varying values of q. We use the Jaccard measure for semantic similarity.

IR-AND 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. The keywords are encoded using AND semantics, such that qualifying documents must contain all of the given keywords in the query. For the entity retrieval task (see Section 5.3.2) we search over the Wikipedia text describing each of the entities.

IR-OR is the same IR approach described above but with OR semantics for the keywords. Because the information needs can be quite large (by IR keyword query standards) the AND semantics can often produce empty results. Thus the more broad IR-OR system allows a more relaxed, but less precise, variation of the IR retrieval approach.

[2]http://opennlp.sourceforge.net/
[3]http://lucene.apache.org/
[4]http://www.oracle.com/technology/products/berkeley-db/je/index.html

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 6: An example entry from TREC QA task and its translations.

Syntax-Only is a simple disambiguation approach that uses only syntactic similarity to perform disambiguations. 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.

Concept-Query is a variant of the *QUICK* system that uses hand-coded (i.e., manually disambiguated) structured concept queries. This system skips the disambiguation phase and begins processing with correct structured concept queries as constructed by the experimenters.

For the entity task, our IR systems use a commercial web search engine for document retrieval over Wikipedia by building on Yahoo BOSS[5]. For the document retrieval task, we build a simple IR engine using Lucene since our document collection is not available online.

5.2 Data and Workload

Data Set We use the 2008 version of YAGO [24] as our knowledge base (note that this is approximately three times larger than the initial instance reported in [24]). YAGO consists of about 6 GB of raw ontology and entity data. It contains over 2 million entities, about 250,000 primitive concepts, 100 relations, and over 20 million facts about these entities, concepts, and relations.

In order to measure the disambiguation quality 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 disambiguations 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. 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.

We use the AQUAINT2 news collection as our corpus, consisting of over 900,000 english news articles from various sources, collected between 2004 and 2006.

Effectiveness Workload For the quality evaluation, we needed a set of information needs which form non-trivial queries when expressed as structured 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 [8]. 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 about the target. Answers to questions (along with relevant documents producing the answers) are also provided. 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 6 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 our 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.

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 wide 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. Our workload includes flat queries (conjunctions with no relations), chain queries (a nesting of relations), and star queries (a conjunction of relations) of each length and generality.

5.3 Experimental Results

5.3.1 Disambiguation Task

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

We consider the following question for each query. Was the computed (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.

Figure 7 shows the results of running our disambiguation algorithm with the raw (incomplete) and extended KB. This

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

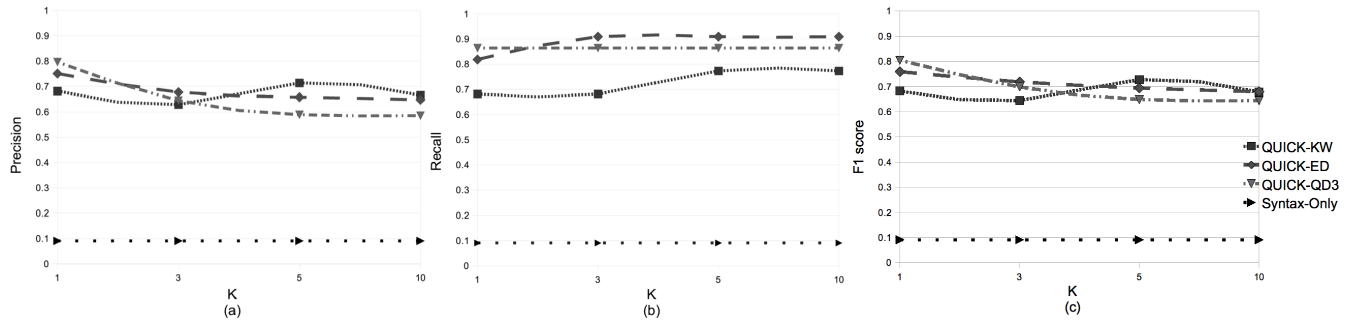

Figure 8: Average precision (a), recall (b), and F$_1$-score (c) vs. the number of query disambiguations (k) considered for the entity retrieval task.

demonstrates disambiguation behaviour in the case of an incomplete KB and a complete KB. We take the top ranking disambiguation and use our systems default of q-gram distance and the Jaccard index for semantic similarity.

	Mapped	Empty	
Correct	0.455	0.500	0.955
Incorrect	0.045	0.000	0.045
	0.500	0.500	

(a) Raw KB

	Mapped	Empty	
Correct	0.909	0.000	0.909
Incorrect	0.091	0.000	0.091
	1.000	0.000	

(b) Extended KB

Figure 7: Proportion of queries that get disambiguated (Mapped) or do not get disambiguated (Empty) vs. whether the disambiguation is correct.

The results demonstrate that even in the presence of incomplete knowledge (table (a)), our system can still compute a correct disambiguation around 95% of the time. Interestingly, our approach proves to be very accurate in determining when a disambiguation does not exist, and returning no disambiguation rather than an incorrect one. This is particularly promising since in the document retrieval scenario, this situation corresponds to falling back to keyword search. Correctly identifying when to fall back to keyword search means we do not process incorrect disambiguations which would produce irrelevant results. When a complete KB is introduced (table (b)), the task becomes more difficult since there exists a correct disambiguation for every query. Our algorithm sees a slight drop in overall correct disambiguations, but still stays around 90% correct. It is important to note that all of the correct runs come from correct disambiguations rather than identifying situations in which no disambiguation exists as in the raw KB situation.

5.3.2 *Entity Search Task*

In the entity search task, we used our TREC-based workload to evaluate the accuracy of retrieving entities described by the queries. The results are compared against the baseline Concept-Query system which returns correct results by definition. For this task, *QUICK* does not make use of the residual keywords left over from partial disambiguations since there is no text retrieval component for entity search. Thus evaluation in the raw KB scenario is not well defined for this task, and we consider only the extended KB.

Figure 8 shows the precision, recall, and F$_1$-score of entities retrieved as a function of k, the number of query disambiguations considered by the *QUICK* systems. In all cases, we take the union of the top-k disambiguations. The three variants of *QUICK* correspond to the configurations of syntactic similarity (keyword occurrence (KW), edit distance (ED), and q-gram distance 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 with the top ($k = 1$) disambiguation, 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 distance at $k = 1$, while recall is best with edit distance at $k = 3$.

Figure 9 shows the results for all systems, including three variations of *QUICK* with fixed k values. The IR-OR result proves to be too general in using the "OR" retrieval semantics and produces no relevant results for any query in the workload. The IR-AND and Syntax-Only disambiguation approach 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 8.

We also varied the size of candidate sets from 10 to 100, but found this had little to no impact on result quality. This means that the correct schema items generally appear within the first 10 candidates, or not at all within the first 100. 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 distance 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 systems can not. The poor performance of the Syntax-Only system emphasizes the need for disambiguations that consider the semantic coherence of the mapping of the query into the underlying structured data.

5.3.3 *Document Retrieval Task*

The goal of the document retrieval task is to find documents relevant to the entities described by the query. Note that this is slightly different from the traditional IR setting, in which documents relevant to the query keywords are the

System	Precision	Recall	F-Score
QUICK-QD3-K1	**0.80**	0.86	**0.80**
QUICK-ED-K3	0.67	**0.91**	0.72
QUICK-KW-K5	0.71	0.77	0.72
IR-AND	0.07	0.11	0.08
IR-OR	0.00	0.00	0.00
Syntax-Only	0.09	0.09	0.09

Figure 9: Average precision, recall, and F_1-score for the entity retrieval task.

target. This difference emphasizes the schema based querying enabled by semantic search systems.

We pooled the top-10 results from each system and manually assessed relevance. Relevance scores were assigned on a five point 0 to 4 scale, with scores of 0 to 2 counting as irrelevant and scores of 3 and 4 counting as relevant for precision measures. For each assessment, the judge was shown only the document and query, with no information as to which system produced the result. This ensures there was no bias in relevance judgements. We computed *normalized discounted cumulative gain* (NDCG) [15] normalized against an ideal ranking using the human valuations, *mean average precision* (MAP), and *precision at 10* (P@10).

Figure 10 shows the results for the document retrieval task, with the default *q*-gram distance using the top disambiguation for *QUICK*. There is a clear relationship between the amount of reference knowledge used and the quality of retrieval. The IR systems use no background knowledge and

System	NDCG	MAP	P@10
QUICK-EXT	**0.61**	**0.61**	**0.53**
QUICK-RAW	0.46	0.44	0.40
Concept-Query	0.40	0.54	0.42
IR-OR	0.48	0.44	0.37
Syntax-Only	0.05	0.09	0.06
IR-AND	0.0	0.0	0.0

Figure 10: Results for the document retrieval task.

are unable to make use of the structure in the query resulting in moderate relevance scores for the IR-OR system. The IR-AND system proves to be too selective for news articles and returns empty results in all cases. *QUICK* with the raw KB is able to use background knowledge on roughly half of the queries (see Section 5.3.2), giving an increase in quality. Finally, *QUICK* with the extended KB improves quality even further. Interestingly, the Concept-Query system using the explicit structured concept queries does not perform quite as well as the *QUICK* systems. This shows that the integration of context keywords in ranking can produce an observable difference in quality.

Note that all systems use a common implementation for the actual document retrieval, a Lucene index. Improvements made to this component of the system would likely benefit all systems.

5.3.4 System Performance

The performance of the disambiguation procedure was measured across all queries in both workloads. We found that disambiguations took under one second in all cases, with an average of 0.2 seconds.

The full system performance on our TREC derived workload averaged 0.81 seconds due to the relative simplicity in the structure and generality of the queries. The synthetic workload on the other hand exercises a much broader

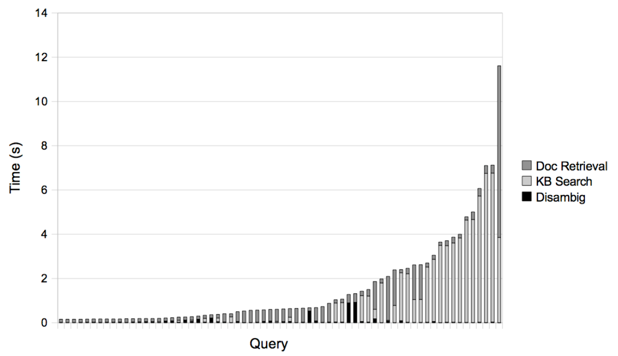

Figure 11: Full system run times for all queries in the synthetic workload.

class of query shapes and generalities. Figure 11 shows the breakdown of performance over the synthetic workload for the three phases of query processing (query disambiguation, KB search, and document retrieval). KB search is generally the most expensive task, while document retrieval occasionally dominates. This corresponds to situations in which there are a large number of qualifying entities for the query, and thus the document retrieval must make a large number index look-ups. In general the disambiguation component accounts for only a small fraction of the total run-time.

6. RELATED WORK

The problem of mapping keywords into structured data 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 [11]. The challenge in this 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 in 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. Finally, our approach composes complex queries by considering how keyword expressions can be mapped into conjunctions of and relations between schema items, giving much greater expressivity than flat keyword queries.

The WebTables project [3] 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 [5, 6, 14, 17], in the integration of ambiguous terms into the structured part of queries. NAGA [17] 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. The EntityRank system [6] incorporates keywords into structured queries as so called context keywords. Our approach of including parts of queries which can not be disambiguated as residual keywords for ranking is similar. 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.

With respect to the semantic search side of our system, our approach differs from current works in a number of ways. First, semantic search systems generally assume documents are annotated with structured semantic data, such as RDF or OWL [10, 21], or XML [9] (with explicit references to the ontology concepts) and often assume a structured query language. Our system uses a mix of extracted structure from text, like the fact triples in NAGA, and the free text itself, like the text-based semantic search engine ESTER [2]. Retrieval based on entities occurring in text is also done by [5] using a structured query language with no ambiguity.

A few works have considered KB-based semantic search over text databases making use of keyword search interfaces, the most successful of which is ESTER [2], an ontology-assisted Wikipedia search engine. In ESTER the keyword query is matched against known primitive concepts and entities, and a dynamic user interface allows users to disambiguate among possible matches. However, keyword mappings are based solely on primitive concepts and entities, and no attempts to compose complex concepts from sets of keywords are made.

7. CONCLUSIONS AND FUTURE WORK

We have proposed a keyword-based structured query language that trades off expressivity and flexibility in utilizing web-based structured data extractions. 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 also shown how our query processing framework fits in to an end-to-end semantic search system.

An interesting problem for future work is ranking. There is an abundance of information available at ranking time: the qualifying entities, the qualifying documents, statistical metrics of entities to documents (e.g., tf-idf scores), the terms mentioned in the query, and so on. More sophisticated approaches may exploit this information to improve ranking. Another avenue for future work is in applying our disambiguation model to flat keyword queries. In this scenario we lose the expressiveness of the query structure, but can still attempt to exploit the semantics of the structured data by mapping keywords into the structured data graph.

8. REFERENCES

[1] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[2] H. Bast, A. Chitea, F. Suchanek, and I. Weber. ESTER: efficient search on text, entities, and relations. In *SIGIR '07: Proc. of the 30th intl. conf. on information retrieval*, pages 671–678. ACM, 2007.

[3] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. WebTables: exploring the power of tables on the web. *Proc. VLDB Endow.*, 1(1):538–549, 2008.

[4] M. J. Cafarella, C. Re, D. Suciu, and O. Etzioni. Structured Querying of Web Text Data: A Technical Challenge. In *CIDR*, pages 225–234, 2007.

[5] 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 Engineering 19(02)*, pages 261–272, 2007.

[6] T. Cheng, X. Yan, and K. C.-C. Chang. EntityRank: Searching Entities Directly and Holistically. In *VLDB*, pages 387–398, 2007.

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

[8] H. T. Dang, D. Kelly, and J. J. Lin. Overview of the trec 2007 question answering track. In *TREC*, 2007.

[9] F. Farfan, V. Hristidis, A. Ranganathan, and M. Weiner. XOntoRank: Ontology-Aware Search of Electronic Medical Records. In *To appear in 25th Intl. Conf. on Data Engineering (ICDE 2009)*, 2009.

[10] R. Guha, R. McCool, and E. Miller. Semantic search. In *WWW '03: Proc. of the 12th Intl. Conf. on World Wide Web*, pages 700–709. ACM, 2003.

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

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

[13] P. G. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano. Towards a query optimizer for text-centric tasks. *ACM Trans. Database Syst.*, 32(4):21, 2007.

[14] A. Jain, A. H. Doan, and L. Gravano. Optimizing SQL Queries over Text Databases. *Data Engineering, 2008. ICDE 2008. IEEE 24th intl. conf. on*, pages 636–645, 2008.

[15] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

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

[17] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. NAGA: Searching and Ranking Knowledge. In *24th Intl. Conf. on Data Engineering (ICDE 2008)*, 2008.

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

[19] D. 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.

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

[21] 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.

[22] P. 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.

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

[24] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge - Unifying WordNet and Wikipedia. In *16th Intl. World Wide Web Conference (WWW 2007)*, pages 697–706, 2007.

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