# HyKSS: Hybrid Keyword and Semantic Search

Andrew Zitzelberger

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

David W. Embley, Chair
Stephen W. Liddle
Tony R. Martinez

Department of Computer Science

Brigham Young University

December 2011

ABSTRACT

HyKSS: Hybrid Keyword and Semantic Search

Andrew Zitzelberger
Department of Computer Science, BYU
Master of Science

The rapid production of digital information makes the task of locating relevant information increasingly difficult. Keyword search alleviates this difficulty by retrieving documents containing keywords of interest. However, keyword search suffers from a number of issues such ambiguity, synonymy, and the inability to handle semantic constraints. Semantic search helps resolve these issues but is limited by the quality of annotations which are likely to be incomplete or imprecise. Hybrid search, a search technique that combines the merits of both keyword and semantic search, appears to be a promising solution.

In this work we introduce HyKSS, a hybrid search system driven by extraction ontologies for both annotation creation and query interpretation. HyKSS is not limited to a single domain, but rather allows queries to cross ontological boundaries. We show that our hybrid search system, which uses a query driven dynamic ranking mechanism, outperforms keyword and semantic search in isolation, as well as a number of other non-HyKSS hybrid ranking approaches, over data sets of short topical documents. We also find that there is not a statistically significant difference between using multiple ontologies for query generation and simply selecting and using the best matching ontology.

Keywords: hybrid search, information retrieval, ontologies, cross-ontology queries

# ACKNOWLEDGMENTS

I would like to thank Dr. Embley for taking a chance on me and for teaching me how to conduct research and write papers. I would also like to express gratitude to my committee members, the DEG group, and all of the students and professors at BYU that helped me on my way. Most of all, I want to thank my dear wife and our children for their support, and for giving me the time necessary to see this project through to completion. My sincere thanks to all who contribute funding to support education and research. I look forward to passing on the favor.

## Table of Contents

# Chapter 1

## Introduction

The world is producing digital information at a prodigious rate. An already well outdated study estimated the indexable Web alone to contain more that 11.5 billion pages [GS05]. More recently, Google[1] announced the discovery of 1 trillion unique URLs with a growth rate of several billion per day.[2] The sheer amount of information available has made it increasingly difficult to locate the most relevant information.

Keyword search methods alleviate the problem by retrieving documents containing user specified keywords. These documents are likely to be relevant due to the presence and importance of keywords of interest. However, keyword search has a number of limitations. Ambiguous keywords may result in the retrieval of irrelevant documents. Document publishers may use words that are synonymous with, but not identical to, the terms in the query causing relevant documents to be missed. Further, keyword search is incapable of recognizing semantic constraints on information. If a query specifies "under 12 grand", a keyword search will treat each word as a keyword (or stopword) despite the fact that many, if not most, relevant documents will likely not contain any of these words.

Semantic search helps resolve the shortcomings of keyword search by considering possible semantics of queries and documents. For purposes of this work, we consider semantics to be the classification of information according to some schema. We use machine-readable annotations to provide these semantics. Semantic search is the process of interpreting a user's query with respect to underlying semantics and using those semantics to provide relevant

[1] http://www.google.com/

[2] http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html

search results. Thus, for example, semantic search can interpret the query "under 12 grand" to indicate interest in retrieving documents containing monetary values less than $12,000. Searching using this interpretation is more meaningful than retrieving documents based on the words "under", "12", and "grand".

For semantic search to be effective, high quality annotations must exist. However, producing such annotations is not a trivial task. Manual hand annotation is time consuming and human annotators may disagree about which annotations are most appropriate. Alternatively, semi-automatic or automatic tools can extract information and produce annotations. While great progress has been made on such tools, even the best miss critical information or provide inaccurate annotations. Searching using these annotations alone may miss relevant information that is not properly annotated.

Hybrid search has been proposed as a means for mitigating the weaknesses of both approaches in the presence of incomplete annotations [BCC+08]. In this work we introduce HyKSS[3] (pronounced "hikes"), our *Hy*brid *K*eyword and *S*emantic *S*earch system. HyKSS is fundamentally an information retrieval engine concerned with returning links to documents that are relevant to user free-form queries. The use of semantics, in addition to keywords, allows the system to retrieve relevant documents more effectively.

The driving force behind processing semantics in HyKSS is a library of conceptual models called extraction ontologies. An extraction ontology is a means of tying linguistic information to conceptual models [EZ10]. Extraction ontologies are used to recognize concepts in text and create canonical annotations that can be used downstream for data storage or query interpretation. HyKSS is not limited to the use of a single extraction ontology per query, but rather can execute queries that cross multiple ontology boundaries by dynamically generating relevant ontology sets.

We show that extraction ontologies can be used as the basis for a cross ontology hybrid search system and that our system outperforms both keyword and semantic search in

[3]See http://dithers.cs.byu.edu/wok/hykss/ for an evolving demo of HyKSS.

isolation over data sets of short topical documents, which includes pages from craigslist[4] and Wikipedia.[5] We also show that our dynamic query driven approach to ranking outperforms several other approaches on the same data sets. Further, we demonstrate that in some instances using multiple ontologies outperforms selecting and using the single best matching ontology for queries in which a single ontology does not adequately cover the query domain. However, we find that the difference in performance is not statistically significant.

We explain the details of the contributions in this thesis as follows. Chapter 2 begins with discussion of extraction ontologies and their use as extractors and query interpreters. Chapter 3 describes the indexing architecture of HyKSS and the algorithm used for processing user queries. In Chapter 4 we discuss the search interfaces made available to HyKSS users. A discussion of our experimental results is in Chapter 5, and in Chapter 6 we compare HyKSS with similar hybrid search approaches. Finally, in Chapter 7, we draw conclusions and address possible areas for future work.

[4]http://www.craigslist.org
[5]http://www.wikipedia.org

# Chapter 2

# Extraction Ontologies

## 2.1 Extraction Ontology Components

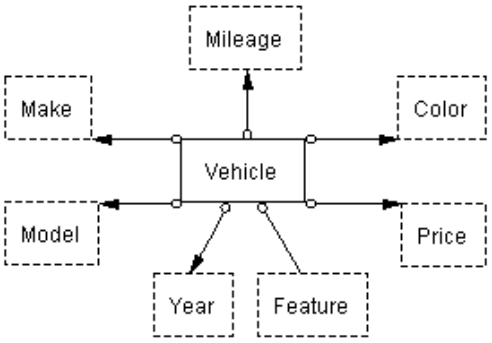


Figure 2.1: Graphical view of a simple ontology for the *Vehicle* concept.

An extraction ontology is a conceptual model augmented with linguistic information to enable information extraction over text. The primary components of the model are object sets, relationships sets, constraints, and linguistic recognizers. Figure 2.1 shows an example of an extraction ontology for the *Vehicle* concept in its conceptual-model form.

Each rectangular box represents an object set. A box with a dashed border, such as *Price* in Figure 2.1, denotes a lexical object set—one that has associated linguistic information for recognizing instances in text. A box with a solid border, such as *Vehicle*, denotes a non-lexical object set—one whose elements are object identifiers denoting real-world objects. Non-lexical instances are generated when sufficient evidence appears in related lexical concepts and in keywords or keyword phrases.

The line segments between object sets denote relationship sets. The endpoints of each relationship set are used to denote participation constraints on objects from the connected object sets participating in relationships within the relationship set. These endpoints can be a white circle or a black arrowhead, or neither, or both. The white circle represents an optional constraint—a constraint allowing object participation in a relationship to be optional. All participation constraints for *Vehicle* in Figure 2.1 are optional, meaning that during extraction the *Vehicle* ontology does not need to extract values for any connected object set.[1] For example, if a *Color* is not extracted from a vehicle advertisement, an object for the connected, non-lexical object set *Vehicle* can still be extracted. The black arrowhead indicates that the constraint is functional—from tail to head, the constraint has a maximum participation constraint of one. All of the relationship sets in Figure 2.1, except the one connected to the *Feature* object set, are functional, meaning that only one of each lexical concept can be extracted from a document. For example, only a single *Make* or *Model* will be extracted from a document, even if more values that could be extracted are present. A line segment that does not have a black arrowhead on its endpoint has an unbounded participation constraint. This type of constraint can be seen on the relationship set connecting the *Vehicle* and *Feature* object sets in Figure 2.1. This unbounded constraint indicates that the *Vehicle* ontology will extract as many *Feature*s as the recognizers find in a document.

Extraction ontologies have several other types of components, such as aggregation and generalization/specialization, and have other constraint types. These facilities come from the conceptual modeling language OSM [EKW92]. However, these facilities are not present in the ontologies used in this work and currently receive varying levels of support in HyKSS.

The linguistic information in lexical concepts is represented using data frames. A data frame uses regular expressions to capture the textual properties of concept instances [Emb80] and is one means of creating a linguistically grounded ontology [BCHS09]. Figure 2.2 shows part of a sample data frame for *Price*. The *internal representation* indicates how the system

[1] In practice we require at least one lexical object set extraction to generate a non-lexical instance even if all lexical object set extractions are optional.

will store extracted values internally. *External representations* consist of a series of regular expressions specifying how instances might appear in text. The textual distance of matches from *context keywords* helps determine which match to choose for potentially ambiguous concepts within an ontology. The string "40K", for example, could be interpreted as a *Mileage* or a *Price*, but would be interpreted as a *Price* when words such as *asking* or *negotiable* appear nearer to it than to the context keywords for *Mileage*.

Price
  **internal representation:** Double
  **external representations:** \$[1-9]\d{0,2},?\d{3} | \d?\d [Gg]rand | ...
  **context keywords:** price|asking|obo|neg(\.|otiable)| ...
  ...
  **units:** dollars|[Kk] ...
  **canonicalization method:** toUSDollars
  **comparison methods:**
    LessThan(p1: Price, p2: Price) **returns** (Boolean)
    **external representation:** (less than | < | under | ...)\s*{p2} | ...
    ...
  **output method:** toUSDollarsFormat
  ...
**end**

Figure 2.2: Sample data frame for price.

In a data frame, *units*, the *canonicalization method*, and *comparison methods* allow for semantic comparisons over extracted values. *Units* express units of measure or value qualifications that help quantify extracted values. In Figure 2.2, *K* indicates multiplication by 1,000 and *dollars* specifies a type of currency. A *canonicalization method* converts an extracted value and units (if any) to a unified internal representation. Once in this representation, *comparison methods* can compare values extracted from different documents despite being represented in different ways. These methods can correctly confirm, for example, that "$4,500" is less than "5 grand." Comparison constraint extraction via the external representations of *comparison method*s are intended for query extraction only and not for general document extraction. The *output method* is responsible for displaying internally-stored values to the user in a readable format. HyKSS makes use of this feature when displaying output to users.

## 2.2 OntoES: Ontology Extraction System

OntoES, our *Onto*logy *E*xtraction *S*ystem, applies extraction ontologies to text in order to extract information and produce annotations. The extraction process uses the linguistic information provided by data frames and the constraints of the model structure along with several heuristics to perform the information extraction task. Past work shows that OntoES performs well in terms of precision and recall for the extraction task when documents are rich in recognizable constants and narrow in ontological breadth [ECSL98].

Additional prior work has also focused on the ability of OntoES to discover separate records in multiple record documents and extract from each record separately [ECJ+99]. For our work on HyKSS, we chose to use a data set consisting of short topical documents and thus treat each document as a single record. Future work could attempt to properly handle multiple record documents and even properly handle completely unstructured text where clear record boundaries do not exist.

# Chapter 3

# HyKSS Architecture and Processing

## 3.1 Indexing Architecture

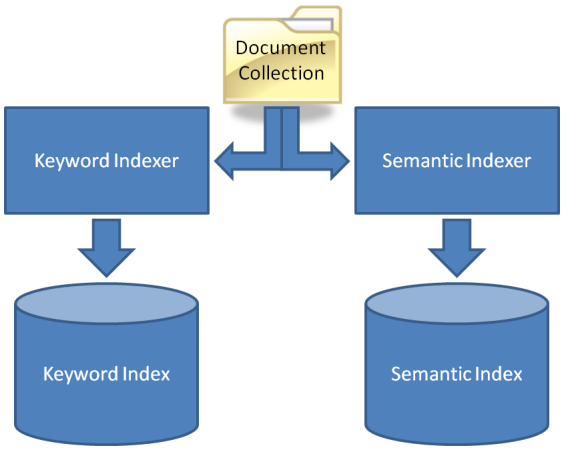


Figure 3.1: A visual representation of the indexing architecture of HyKSS.

The architecture of HyKSS consists of a number of generic interfaces that allow interchangeability among components. As shown in Figure 3.1, the required components are a document collection to index, keyword and semantic indexers to generate the necessary internal document representations, and keyword and semantic indexes to store the document representations produced by the indexers. The remaining details in this chapter are specific to our implementation of this architecture.

Each indexer contains internal resource handling capabilities for extracting text content from different document types. For the HTML documents used in this work, this process consists of stripping away all tags and retaining the title and the content of all text

nodes. After resource handling, the text content appears as: "title: <extracted title>\ntext: <extracted text>". After the text content for a document is obtained, the indexer proceeds to create and index a document representation for that document.

The keyword processing line is driven primarily by Lucene[1] which provides functionality for creating, storing, and searching document representations. We use Lucene's full text indexing functionality to index the text content of each document as a single field.[2] We chose to use full text indexing to maintain the ability to meaningfully process queries containing phrases that include stopwords. The generated index is stored on the local file system unoptimized.[3]

Our implementation relies heavily on extraction ontologies. The semantic indexer consists of OntoES and a specified library of ontologies. OntoES applies each ontology in the library to the text content of each document in the collection. An internal module converts the annotations generated by OntoES into an RDF[4] format that is stored in the semantic index. We implement the semantic index using Sesame[5] with the local file system storage option.[6]

The library of ontologies used by the semantic indexer is not fixed and can be updated as desired. Developers can add new ontologies and remove or modify existing ontologies. These pay-as-you-go [FHM05] changes can be used to enhance indexing and resulting query performance. Currently the entire document collection must be re-indexed after modification of the library. Query execution involving new or modified ontologies will not work properly until the re-indexing takes place.

[1] http://lucene.apache.org/

[2] The title and location of each document are stored in separate fields for later retrieval.

[3] Version 3.0.2 is used for all Lucene operations.

[4] http://www.w3.org/RDF/

[5] http://www.openrdf.org/

[6] Version 2.1 of Sesame is used with spoc, posc, and psoc indexes. Index selection chosen from benchmarking results at http://www4.wiwiss.fu-berlin.de/benchmarks-200801/.

## 3.2 Query Processing

HyKSS can begin processing queries after the keyword and semantic indexes are created. Similar to the indexing architecture, the query processing mechanism also has two distinct paths: one for keyword query processing and one for semantic query processing. Both of these execution paths consist of a query pre-processing step, a query execution step, and a query post-processing step. After both execution paths have completed, the results are combined to provide a meaningful ranking.

To demonstrate the implementation of these steps in HyKSS consider the query "hondas in 'excellent condition' in orem for under 12 grand". Suppose that we have an ontology library containing ontologies for *Vehicle* and *Location*. The *Vehicle* ontology is similar to the ontology in Figure 2.1, but does not include the *Feature* object set or its connected relationship set. The *Location* ontology is a smaller ontology that has recognizers for US states and some US cities. We use this query and these ontologies to describe the query processing of HyKSS.

### 3.2.1 Keyword Query Processing

The pre-processing of keyword queries begins with the removal of comparison constraints. A comparison constraint is a phrase that indicates a relational constraint on the information. For example, the phrase "under 12 grand" in the example query indicates that the user expects to retrieve documents where the price of the vehicle is less than $12,000. The terms "under", "12", and "grand" likely constitute noise as they are unlikely to occur in relevant documents.

To remove comparison constraints, the keyword query processor stores a cache of *comparison method* recognizers from the ontologies in the library. The keyword query processor runs each of these expressions against the query and collects the matches. The processor then removes these matches from the query, beginning with the longest match. The longer matches are removed first to avoid leaving comparison constraint words in the query.

This ensures, for example, that "under 12 grand" is removed, rather than just "under 12", which would incorrectly leave "grand" as a keyword.

However, the keyword query processor leaves comparison constraints that use the equals method in the query. The positive equals comparison denotes a single value whereas inequality comparisons denote a range of values. As such, the value of the positive equals comparison may often contain keywords that are helpful in the keyword ranking. In the example query, the processor recognizes the constraints $Vehicle.Make$ = "honda" and $Location.US\_City$ = "orem" using the words "hondas" and "orem". We leave these words in the keyword query because the presence of these keywords is likely to indicate a document relevant to these constraints. In our running example query, the keyword query processor recognizes the phrase "for under 12 grand" as an inequality comparison constraint and removes it from the query. The remaining query is "hondas in 'excellent condition' in orem".

After removing non-equality comparison constraints, the keyword query processor removes Lucene special characters[7] and common punctuation characters (except for quotes) from the query. We remove Lucene special characters to prevent users from triggering functionality unknowingly. For example, a user may use a '?' character in a query and get unexpected results, not realizing that '?' is the Lucene wildcard character.

Next, the keyword query processor removes all non-phrase stopwords[8] from the query. A phrase is any quoted string such as 'excellent condition' in our example query. We do not remove stopwords from phrases because certain phrases[9] such as 'no dings' may contain relevant stopwords in them. The processor only removes stopwords if they are surrounded by white space characters, which is why common punctuation is removed along with Lucene special characters. Thus the processor removes two occurrences of "in" from the running example query, leaving it further reduced to "hondas 'excellent condition' orem".

[7] http://lucene.apache.org/java/2_4_0/queryparsersyntax.html#Escaping Special Characters

[8] Stopword list obtained from http://armandbrahaj.blog.al/2009/04/14/list-of-english-stop-words/.

[9] Lucene's phrase slop parameter is set at two to assist in locating useful phrases.

Upon completion of the keyword query pre-processing step the keyword query processor uses Lucene to handle the query execution of the processed query. The post-processing step simply returns the results in the same order and with the same scores as they were returned by Lucene.

### 3.2.2 Semantic Query Processing

The processing of semantic queries is more intensive than that of processing keyword queries. The pre-processing step transforms a user's free-form query into a structured semantic query. We use the SPARQL query language[10] for our implementation of HyKSS. The transformation process in HyKSS is an extension of the AskOntos system [Vic06]. Given a free-form query, AskOntos chooses the best matching ontology in the library and uses it to generate a structured query. HyKSS semantic processing differs in that ontology sets are also considered. (AskOntos does not have a keyword component.)

The semantic query processor extracts from each query as if it were the text content of a document. However, in addition to document extraction functionality, the semantic query processor also extracts comparison constraints. As user queries are generally quite short, extraction from queries generally requires little processing time. The semantic query processor begins pre-processing by applying OntoES to the query using each ontology in the library. AskOntos then assigns a score to each ontology indicating how well it matched the query. The mechanism for computing these scores is much the same as that used in the original AskOntos. AskOntos gives an ontology one point for each *external representation* match and each *context keyword* match (this portion of the scoring matches the description given in [Vic06]), half a point for each *parameter match* in a *comparison method*, and 3.5 points for matching the object set OntoES deems to be the primary object set ($Vehicle$ in our example). This scoring scheme assumes the primary object set is always non-lexical and the 3.5 point score is not added to ontologies with no non-lexical object sets.

[10] http://www.w3.org/TR/rdf-sparql-query/

HyKSS uses the scores supplied by AskOntos to consider ontology sets that may better correspond to the query. The semantic query processor sorts the ontologies according to their scores. In the event of a tie, the processor gives preference to the ontology containing the larger number of object sets. Ties that occur on this criterion are broken arbitrarily. The processor immediately discards ontologies with a score of zero from further processing as they can never enhance an ontology set.[11] We expect that this will often largely reduce the number of ontologies for consideration in large applications.

The semantic query processor adds the highest scoring ontology to its own ontology set. Moving down the list, the processor considers each ontology below its position for inclusion in the set. The processor adds an ontology to the set if it contains a match not currently contained by any of the ontologies in the set, or if it contains a match that subsumes a match currently in the set. For each such match, the processor adds a single point to the value of the set. Set generation then begins with the second highest scoring ontology and proceeds until each ontology has been considered as the first ontology in the set. This set generation algorithm can generate any subset of the power set (except the empty set) but considers only a fraction of the possibilities due to aggresive pruning. The runtime complexity of our ontology set generation algorithm is $O(N^2)$ where $N$ is the number of ontologies to consider. The complexity also increases as the size of the ontologies and the number of matches discovered increases. The scoring used for ontology set generation is completely heuristic based, and we leave more experimentation with this process to future work.

For the running example query, "hondas in 'excellent condition' in orem for under 12 grand", the process proceeds as follows. The $Vehicle$ ontology recognizes "hondas" and "for under 12 grand" as constraints on $Vehicle.Make$ and $Vehicle.Price$, respectively. These two matches influence the score of the $Vehicle$ ontology positively as scored by AskOntos. Suppose the library contains another ontology for $ContractualServices$. The $ContractualServices$ ontology would also recognize "under 12 grand" as a constraint. However, the keyword $Vehicle$

[11] If all ontologies have a score of zero then a singleton ontology set containing an arbitrary ontology from the library is used.

would receive a higher score than *ContractualServices* because of the additional recognized value. When generating sets, *ContractualServices* could not be merged with *Vehicle* because they overlap in their extractions. *Location*, which recognizes "orem" as a *Location.US_City*, could be merged with both ontologies, but merging with *Vehicle* would result in a higher score than merging with *ContractualServices*. As such, the *Vehicle-Location* ontology set would win out as the highest ranking ontology set.

The semantic pre-processing step continues by using the ontology set with the highest score to generate a generic query and then a structured query. The generic query is an intermediate format that allows extension to other structured query languages and is also used for internal and evaluation purposes. The structured query is the query HyKSS actually executes against the semantic index. The framework allows multiple ontology sets to be used in the event of a tie, but in our implementation we break ties by choosing the ontology set that was discovered first. Due to the sorting performed after the initial ranking, this favors ontology sets with an individually high scoring ontology.

When generating the generic query, the semantic query processor uses all matches in the free-form query to form a conjunctive query (disjuncts and negations are not considered for free-form queries, but can be specified in advanced form-based queries). Additionally, ontologies whose matches have all been subsumed by other ontologies during the ontology set generation process are dropped from the ontology set during generic query generation. For the running example query, the selections of the generated generic query consist of $Vehicle.Make$ = "honda", $Vehicle.Price < 12000$, and $Location.US\_City$ = "orem". A projection is also generated on each attribute used in a selection. We assume that if users select on an attribute-value pair they are also interested in projecting on the attribute for the value as well. Projections can also be generated individually if an ontology finds specified keyword matches in the query. For example, if the query were "Honda models" a selection would be generated where $Vehicle.Make$ = "honda" and projections would be generated on

$Vehicle.Make$ and $Vehicle.Model$. Figure 3.2 shows the generated selections and projections for the running example query.

Selections:
($Vehicle.Make$ = "honda")
($Vehicle.Price < 12000$)
($Location.US\_City$ = "orem")

Projections:
($Vehicle.Make$)
($Vehicle.Price$)
($Location.US\_City$)

Figure 3.2: The projections and selections generated for the running example query.

After generating the generic query, the semantic query processor transforms it into a suitable structured query. When generating the structured query, HyKSS uses an open world assumption. A document is considered relevant unless an annotation explicitly violates a constraint. If the price of a vehicle is unknown for a particular document, for example, that document might still be relevant for the example query because we do not know that it violates a constraint. In addition to verifying that constraints are met, the query also picks up annotation information such as the location of the cached document, the document's title, the original text, the generated canonical value, and the generated output value. The cached document location of annotations across different ontologies is verified to be the same to ensure that all ontologies refer to the same document. HyKSS does not support the returning of facts that span multiple documents. Figure 3.3 shows the generated SPARQL query for the running example.

Sesame executes the generated SPARQL query over the indexed annotations. However, the results returned by Sesame (or any other semantic query system), by principle, are not ranked. The results are either correct and thus returned, or incorrect and thus not returned. As such, HyKSS uses a post-processing step to assign scores to the returned results.

The ranking of semantic results is still an open area of research. As no definitive methods exist, we use a simple ranking process based primarily on the amount of requested

```
PREFIX ann:<http://dithers.cs.byu.edu/owl/ontologies/annotation#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX Vehicle:<http://dithers.cs.byu.edu/owl/ontologies/Vehicle#>
PREFIX US_Location:<http://dithers.cs.byu.edu/owl/ontologies/US_Location#>
SELECT ?VehicleLocation ?VehicleTitle ?MakeValue
      ?MakeDisplayValue ?MakeTextValue ?PriceValue
      ?PriceDisplayValue ?PriceTextValue ?US_LocationLocation
      ?US_LocationTitle ?US_CityValue ?US_CityDisplayValue
      ?US_CityTextValue
WHERE
{
?Vehicle ann:primaryInResource ?VehicleResource .
?VehicleResource ann:Location ?VehicleLocation ;
  ann:Title ?VehicleTitle .
?Vehicle Vehicle:Vehicle-Make ?Make .
?Vehicle Vehicle:Vehicle-Price ?Price .
?US_Location ann:primaryInResource ?US_LocationResource .
?US_LocationResource ann:Location ?US_LocationLocation .

FILTER (?US_LocationLocation = ?VehicleLocation)
?US_Location US_Location:US_Location-US_City ?US_City .
OPTIONAL{?Make Vehicle:MakeValue ?MakeValue .
?Make ann:DisplayValue ?MakeDisplayValue .
?Make ann:OriginalText ?MakeTextValue .}

FILTER (!bound(?MakeValue) || regex( str(?MakeValue),"honda", "i")) .
OPTIONAL{?Price Vehicle:PriceValue ?PriceValue .
?Price ann:DisplayValue ?PriceDisplayValue .
?Price ann:OriginalText ?PriceTextValue .}

FILTER (!bound(?PriceValue) || ?PriceValue < 12000) .
OPTIONAL{?US_City US_Location:US_CityValue ?US_CityValue .
?US_City ann:DisplayValue ?US_CityDisplayValue .
?US_City ann:OriginalText ?US_CityTextValue .}

FILTER (!bound(?US_CityValue) || regex( str(?US_CityValue), "orem", "i")) .
}
```

Figure 3.3: The SPARQL query generated from the running example query.

information a result includes. During execution, the query removes all results that do not satisfy the selections, but due to the open world assumption some documents that do not specify values for some concepts may be returned. Semantic post-processing begins by merging all semantic results that share a common document. The semantic query processor

then gives a point to a document for each projection on an attribute that returns at least one value. The more attributes with information requests the document is able to supply the higher the score it receives. HyKSS normalizes the resulting scores by the highest score given to ensure that scores are between zero and one, and then sorts the documents before returning them.

### 3.2.3 Hybrid Query Processing

After the completion of both the keyword and semantic query processing lines, HyKSS combines the keyword and semantic results to produce final hybrid document rankings. Our implementation combines these scores using a linear interpolation approach, where the interpolation weights are determined dynamically based on the interpretation of the query.

The hybrid query processor assigns a concept match score to each query by counting the number of selections and projections in its semantic interpretation (the generated generic query) and multiplying this count by 0.5. This scheme implicitly gives selections more weight because a selection always generates a corresponding projection. The combining process also computes the number of keywords remaining after non-equality comparison constraints and non-phrase stopwords are removed in the keyword processing line. Any sequence of characters separated by white space, including individual words in phrases, is considered to be a word for this calculation. We include each word in a phrase individually because phrases are often useful in locating relevant documents and we don't want to diminish the importance of an entire phrase to be the same as a single keyword. The number of keywords remaining and the concept match score are normalized by the sum of the two terms in order to produce the keyword and semantic weights, respectively. HyKSS computes the final score of a result using a linear interpolation as follows: $(keyword\_score)$ * $(keyword\_weight)$ + $(semantic\_score)$ * $(semantic\_weight)$. HyKSS sorts the result documents according to the new scores to produce the final rankings. We chose this ranking scheme empirically after a small amount of experimentation.

To demonstrate this process consider the example query "hondas in 'excellent condition' in orem for under 12 grand". As Figure 3.2 shows, the system generates three selections and three projections for this query. As such, the hybrid query processor assigns the query a concept match score of $(3 + 3) * 0.5 = 3$. The keyword query processor recognizes the phrase "for under 12 grand" as a non-equality comparison constraint and the two occurrences of "in" as non-phrase stopwords, and removes them from the query. The query now has 4 keywords remaining: hondas, excellent, condition, orem. Normalizing these scores produces a keyword weight of $\frac{4}{4+3} = \frac{4}{7}$ and a semantic weight of $\frac{3}{4+3} = \frac{3}{7}$.

The weights in this dynamic ranking scheme are driven by the query being asked. If HyKSS does not recognize any semantic information then the keyword ranking will be used. The semantic ranking will be used if no keywords remain after keyword and stopword removal. If HyKSS recognizes neither semantic information nor keywords it returns no results. The weights adjust on the spectrum in between the extremes of all semantic queries and all keyword queries based on the amount of semantic information and keywords discovered in the query. This ranking approach is advantageous because it does not require users to manually set weights or annotate data to tune weights that work well for the document collection being used.

Figure 3.4 shows a result table for the running example query. This table displays the rank and title for each document, along with relevant keywords and requested semantic information discovered in each document. This table is taken from a clipping of the result page interface that we explain in further detail in Section 4.3. Here we focus on the ranking produced by our hybrid ranking algorithm.

The top five results in Figure 3.4 have values in the semantic columns ($Us\_city$, $Make$, and $Price$). This indicates that these documents did not violate any semantic constraints. Further, each result has a value for each column indicating that our semantic ranking algorithm assigned the same score to these top five results. The remaining five results violated at least one semantic constraint indicating they received a semantic score of zero. If we were

| Rank | Document | Keywords | Us_city | Make | Price |
|---|---|---|---|---|---|
| 1 | 2002 Honda Accord LX Sedan(highlighted) | excellent condition(1) orem(2) | Orem | Honda | $4,995 |
| 2 | 2002 Honda Accord LX Sedan(highlighted) | excellent condition(1) orem(2) | Orem | Honda | $4,995 |
| 3 | 1997 Honda Accord EX(highlighted) | hondas(1) orem(2) | Orem | Honda | $3,200 |
| 4 | HONDA CIVIC '97(highlighted) | orem(2) | Orem | Honda | $2,700 |
| 5 | 2002 Honda Odyssey OBO(highlighted) | orem(2) | Orem | Honda | $6,800 |
| 6 | 2007 Toyota Yaris Sport Edition Low Mileage(highlighted) | excellent condition(1) orem(2) | | | |
| 7 | Ford Fusion SEL 2009(highlighted) | excellent condition(1) | | | |
| 8 | TOYOTA CAMRY 2007(highlighted) | excellent condition(1) | | | |
| 9 | 1993 Plymouth Acclaim - GREAT CONDITION!(highlighted) | excellent condition(1) orem(2) | | | |
| 10 | 2002 Mazda Protege LX(highlighted) | excellent condition(1) | | | |

Figure 3.4: Results for the running example query "hondas in 'excellent condition' in orem for under 12 grand".

limited to only using our semantic ranking algorithm we would be required to return the top five documents in an arbitrary order.

Using the keyword score in combination with semantic score, however, enables HyKSS to move the relevant documents to the top of the ranking. The top five documents all contain the word "orem", but only the top two documents also contain the phrase "excellent condition". The keyword ranking mechanism thus assigns a higher keyword score to these two documents, and the hybrid ranking mechanism correctly places these two relevant documents at the top of the ranking.

# Chapter 4

# Search Interface

The HyKSS user interface provides two different modes for searching a document collection. The basic search option allows users to enter textual free-form queries and the advanced search option allows users to submit queries using a form-based interface. We discuss these alternatives in more detail below.

## 4.1 Basic Search

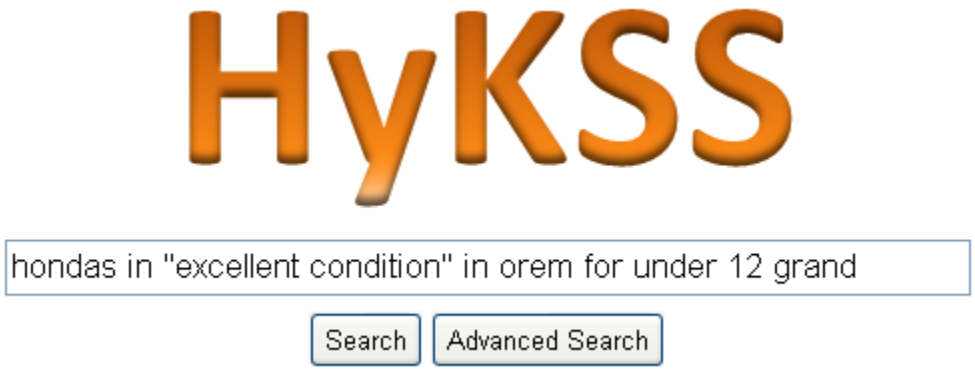


Figure 4.1: The HyKSS basic interface showing the free-form query "hondas in 'excellent condition' in orem for under 12 grand".

The basic search interface is intended to be intuitive to traditional search engine users. The Google-like interface allows users to enter a textual free-form query in a familiar manner. Figure 4.1 shows the implementation of this interface. The "Search" button causes the user query to execute as discussed in Section 3.2.

## 4.2 Advanced Search

The advanced search option of HyKSS functions differently from the advanced search option available on typical search engines. HyKSS expects users to have entered a free-form query before selecting the advanced search option. This query can be as simple as a single world (e.g., "cars") or as advanced as the running example query. This initial query gives HyKSS context in deciding which ontology-based form(s) to generate. If users do not enter a query before selecting the advanced search option they are presented only with a keyword-based form allowing the entry of keywords, required keywords, and disallowed keywords, which is, in any case, appended to all generated forms.

The advanced search mechanism processes a user-submitted free-form query using the query pre-processing steps on both lines. However, the query is not executed. Instead, the HyKSS display uses the best matching ontology set discovered through semantic pre-processing to generate the semantic portion of the form. This portion of the form includes a flattened layout of each ontology in the set along with fields and buttons for modifying the original query. The form interface allows for the creation of more powerful queries as it adds disjunction and negation capabilities not available through the basic search option.

In addition to allowing for more powerful query creation, the form interface allows users to see what concepts are available to the system in the domain area they are searching, and what the system "understood" from the original query.[1] Figure 4.2 shows the form generated using the example query with a disjunction expanded (by a user) to allow the entry of "provo" as an additional option for a *US_City*.

The HyKSS display generates the flattened form layout of an ontology by first selecting a primary object set for the ontology. Many ontologies specify a primary object set to use. For those that do not, the system chooses the non-lexical object set with the highest number of connections. If the ontology does not contain a non-lexical object set, then the lexical

[1]The generation of ontology-based form is an expansion of previous work presented in [EZ10]. HyKSS adds the ability to handle multiple ontologies and keywords.

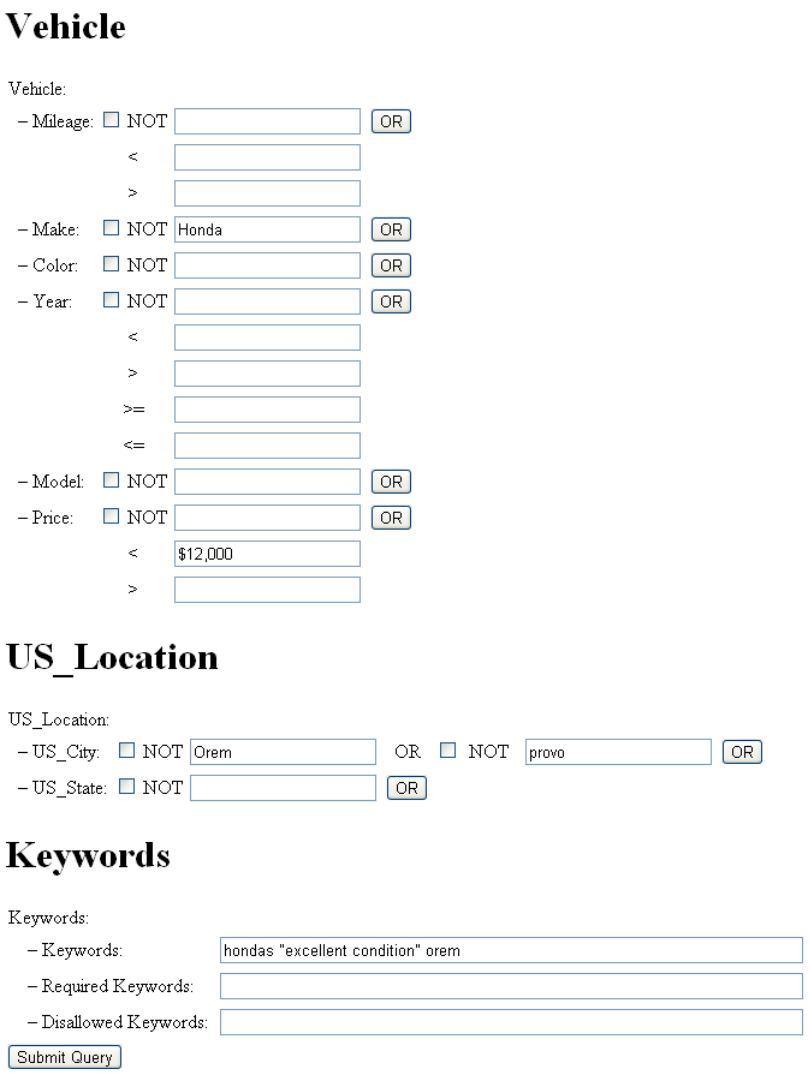


Figure 4.2: The advanced search form generated for the query "hondas in 'excellent condition' in orem for under 12 grand". The *US_City* disjunction has been expanded by a user to allow for "provo" as an additional city option.

object set with the highest number of connections is chosen. Ties are broken arbitrarily. The system then performs a depth-first search through the connections provided by the primary object set to establish the display order of the object sets.

At display time, each ontology begins with its name as a header (e.g., Vehicle and US_Location in Figure 4.2). The primary object set is then displayed followed by other object sets in the order they are visited. Each object set is displayed on its own row. A dash before the name of an object set indicates that it was visited using a relationship set

connection. Other symbol types indicate other types of connections, such as aggregation and generalization/specialization connections (which are not used in this work). The primary object sets (e.g., Vehicle and US_Location in Figure 4.2) are not preceded by a connection symbol as they are not visited using a connection. The space between the left margin and the connection symbol indicates the depth of the object set from the primary object set. Each level of depth is displayed using a two space indent.

The form displays non-lexical object sets using the object set's name and an appended colon. Lexical object sets are displayed in the same manner but are also followed by three input fields allowing for query modification. These fields consist of a "NOT" checkbox for negation, a text field for value input and modification, and an "OR" button to expand the query with disjunctions. Clicking the "OR" button expands the row to include another "NOT" checkbox, another input field, and another "OR" button for further expansion. The original "OR" button is replaced by an "OR" label to indicate that it has been expanded. An object set's methods for executing comparison constraints appear in the rows beneath the object set and are aligned with the first two input fields to indicate that they are methods and not additional object sets. The initial form display includes the semantic values discovered when pre-processing the original free-form query. These extracted values appear in their output-value forms allowing users to see what the system "understood" from the original query. For example, "hondas" and "12 grand" from the free-form query appear as "Honda" and "$12,000" in the form, respectively.

Query execution using the advanced search interface differs slightly from the basic search mechanism. The query inherently contains more structure that can be directly translated into keyword and semantic-search queries. The keyword portion of the form is initially populated using the keywords remaining from the original free-from query after comparison-constraint and stopword removal. Any keywords a user enters in the keyword portion of the form, however, are treated as keywords even if they would have been removed during processing of a free-from query. The advanced form keyword query processor generates

the keyword query from the form by treating the keywords, required keywords, and disallowed keywords as their names suggest according to Lucene keyword processing.

Semantic query generation uses OntoES to interpret the value in each semantic field. Each field is transformed into a string with the format "<field name>: <field value>". For *Price* in the running example, the string appears as "Price: $12,000". OntoES applies the parent ontology to the string format as if it were the text content of a document. If a value is extracted (and canonicalized) using the specified object set, it is used to generate a selection in the semantic query. This application of OntoES is necessary to handle a wide array of input values. For example, suppose that a user enters "100K" into the < field for *Mileage* in the form in Figure 4.2. This value must be canonicalized to "100000" before it can provide meaningful comparisons. OntoES does this by recognizing the string to be a *Mileage*, the substring "K" to be a *unit*, and applying the *canonicalization method* to produce the appropriate value. In some cases the ontology may not successfully extract a value from a field. In this event, HyKSS generates a projection but no selection. It is reasonable to assume that if users attempt to enter a value for a field, even a nonsensical value, they must have interest in that field.

The SPARQL query generated from the structured query is similar to the one presented in Figure 3.3 except that it may contain disjunctions and negations as specified by the user. Query execution and ranking proceeds in the same manner as for the basic search mechanism. Figure 4.3 shows the results for the advanced query. The top three results of this query are identical to those for the example free-form query results shown in Figure 3.4. Further, the top five results for the example free-form query occur in the top six ranking positions for the results of this form query. However, the remainder of the results differ because the advanced query also allows for "provo" to be a *US_City*. Note that "provo" does not appear as a keyword because it is not included as a keyword in the form query. The advanced search interface is currently implemented in a proof-of-concept stage and we leave the implementation

of more advanced and more user friendly features such as displaying top ranked ontologies or allowing users to browse ontologies to select for form generation to future work.

**I understood**: show me US_Location.US_City, Vehicle.Price and Vehicle.Make where US_Location.US_City = Provo|Orem, Vehicle.Price < $12,000 and Vehicle.Make = Honda
**Keywords**: hondas "excellent condition" orem

| Rank | Document | Keywords | Us_city | Make | Price |
|---|---|---|---|---|---|
| 1 | 2002 Honda Accord LX Sedan(highlighted) | excellent condition(1) orem(1) | Orem | Honda | $4,995 |
| 2 | 2002 Honda Accord LX Sedan(highlighted) | excellent condition(1) orem(2) | Orem | Honda | $4,995 |
| 3 | 1997 Honda Accord EX(highlighted) | hondas(1) orem(2) | Orem | Honda | $3,200 |
| 4 | 1996 Honda Accord (Tan)(highlighted) | orem(2) | Provo | Honda | $2,200 |
| 5 | HONDA CIVIC '97(highlighted) | orem(2) | Orem | Honda | $2,700 |
| 6 | 2002 Honda Odyssey OBO(highlighted) | orem(2) | Orem | Honda | $6,800 |
| 7 | 2002 Honda Accord!!! 106k 5,995(highlighted) | | Provo | Honda | $5,995 |
| 8 | Honda Civic EX Coupe(highlighted) | | Provo | Honda | $9,000 |
| 9 | 2002 Honda Civic(highlighted) | | Provo | Honda | $8,199 |
| 10 | 1995 Honda Accord(highlighted) | | Provo | Honda | $1,250 |

Figure 4.3: The results page for the advanced search query.

## 4.3 Results Display

HyKSS displays results somewhat differently from typical search engines. The goal of the results page for any search engine is twofold. First and foremost, a search engine attempts to place the most relevant documents at the top of the rankings. Second, a search engine aims to allow users to quickly determine if a document is relevant or not. Many search engines use page snippets, snippet highlighting, and page previews to accomplish this second goal. HyKSS, however, leverages its knowledge of the ontologies used to answer the query in order to generate a table of results that presents the extracted values for object sets the user

inquired about. Figure 4.3 shows the HyKSS results page for the example advanced search query. The current demo system only returns and displays the top ten results to users.

The results page begins by informing the user what HyKSS "understood" from the query. To demonstrate semantic understanding, the page documents the projections (the "show me"clause) and selections (the "where" clause). This is followed by a declaration of the words in the query that HyKSS considers to be true keywords. These keywords directly correspond to the state of the original query after keyword query pre-processing. For queries executed with the basic search interface, a link to the advanced search interface appears directly beneath the keywords.

The page presents the actual search results in a table format. The first column indicates the ranking assigned by HyKSS, the second contains a link to the returned document, and the third is a list of relevant keywords and their corresponding occurrence counts in the document. A matched phrase is considered a single keyword for display purposes. The remaining columns are generated dynamically based on the projections constructed from the query. The results page in Figure 4.3 contains columns for $Make$, $Price$, and $US\_City$ because projections were generated for these object sets. The cells in these semantic columns contain the relevant extracted information for the document. Only documents that satisfy the selections in the semantic query contain values in the semantic columns. To ensure consistent formatting the display values generated using the object set's *output method* (generated at indexing time) are shown to the user.

The values in the keyword and semantic cells are hyperlinks. Clicking on these links opens the relevant document and highlights the relevant value within the document. There is also a "highlighted" link next to the document link in the "Document" column that highlights all relevant keyword and semantic values in the document. Figure 4.4 shows the top result for the example query highlighted via the "highlighted" link. Notice that although the values in the semantic cells of Figure 4.3 are shown using the output value, the document is highlighted using the original extracted text. Thus, even though the results table lists "$4,955" as the

price, the interface correctly highlights "4955" in the document (our ontology uses "$" as context information and not part of the actual extracted value). The current highlighting mechanism works by highlighting all textual matches to the original text that are surrounded by word boundaries. We leave more valuable highlighting methods to future work.

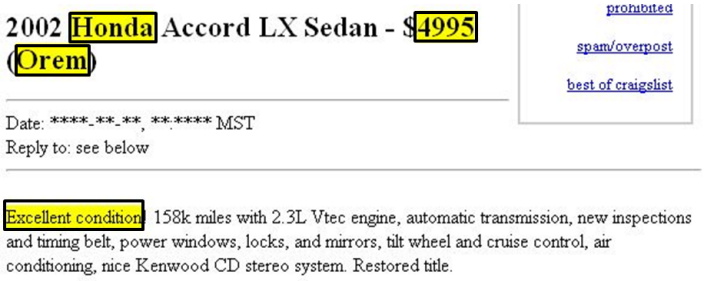


Figure 4.4: The highlighted version of the top result for the example query. (The highlighted values are surrounded by black boxes for extra emphasis and to facilitate black and white printing. Personal information has been removed from the advertisement to help protect user privacy.)

To assist in readability for object sets with non-functional constraints, such as for *Feature* in Figure 2.1, semantic cells are limited to containing three extracted values. In the event that more than three values are extracted a link is generated stating "See $N$ more" where $N$ is the number of additional values. Clicking this link will create a pop-up window containing those additional values. The values in the pop-up window are also clickable and will highlight the value in the returned document. The "highlighted" link in the "Document" column highlights all relevant values including those that appear only in the pop-up window. The three value limit is not applied to the cells in the "Keywords" column as user queries are typically short and every keyword is valuable.

For usability purposes, each of the semantic columns is also sortable.[2] The table stores the canonical value for each extracted value and uses it as a sort key to enable this functionality. Thus, sorting the "Price" column will arrange rows correctly using the numeric amount despite that fact that the output value contains the "$" symbol. String values are

[2]Sorttable javascript library used for sorting. See http://www.kryogenix.org/code/browser/sorttable/

sorted alphabetically. If a cell contains more than one value, only the first value is used when sorting the rows. The "Rank" column is also sortable and is included primarily to allow users to easily return to the original rankings after sorting other columns.

# Chapter 5

# Experimental Results

Our work on HyKSS makes three claims. First, we claim that combining keyword and semantic search capabilities in a hybrid manner outperforms using either approach in isolation when annotations are imprecise and incomplete. Second, we claim that the ranking approach used in HyKSS outperforms a variety of approaches to ranking for hybrid search. Third, we claim that HyKSS allows for pay-as-you-go improvements of the system and that even small ontologies can improve the retrieval performance of the system. We test each of these claims in a series of experiments.

In this chapter, we proceed by first describing the libraries of ontologies used for experimentation. Next, we describe the retrieval metric used to validate our results. This is followed by a description of the document and query sets used in the experiments. We conclude with the presentation and a discussion of our experimental results.

## 5.1 Ontology Libraries

Our experiments make use of a sliding scale of ontology libraries. The most basic library includes only very simple and general semantic modeling, and the highest level specifically models certain aspects, but not the entirety, of a target domain. (We target our experimentation towards the vehicle advertisement domain.) We use this sliding scale of ontology libraries to demonstrate the pay-as-you-go nature of HyKSS. We provide descriptions of each ontology library below. These descriptions are intentionally brief as all ontologies used for

experimentation can be accessed online.[1] The regular expressions for each of the ontologies in the libraries are not intended to be foolproof, but are designed to extract relevant concepts with high precision and recall according to the training set used to develop the recognizers.

**Level 1: Numbers**. The lowest level of semantic modeling consists of a singleton ontology (it contains only a single lexical object set) that recognizes only numbers. We define a number to be text that starts with a digit one through nine and is followed by zero or more digits. We consider the digit zero to be a number as well. The ontology allows for comma groupings and can handle numbers with a single decimal point. The ontology will not pick up textual representations of numbers such as "one" or "two."

The *Number* ontology is intentionally designed to be extremely limited. The ontology does not understand unit designations because they are ambiguous without domain knowledge. Consider the unit "K". This unit could refer to measurements such as temperature in Kelvin, the karats of a diamond, or the multiplier 1000. As such, it would be improper for the *Number* ontology to handle units in a specified way. However, the ontology does know keywords that often refer to numbers such as "price", "mileage", and "year". The presence of these keywords in a query will generate a projection as discussed previously. This does not indicate that the ontology understands these terms, only that the ontology recognizes that the presence of these terms likely indicates the presence of, or interest in, a *Number*. The *Number* ontology handles comparison constraints such as "less than" and "greater than" in the expected manner.

**Level 2: Generic Units**. The Generic Units ontology library increases the level of semantic modeling and provides recognizers for common units. This library consists of three simple ontologies: *DateTime*, *Distance*, and *Price*. The *DateTime* ontology recognizes common date-time formats and canonicalizes values to seconds since 1970 for comparison. If a year is extracted without a date or a time, the assumed date-time is January 1 at midnight (00:00) of that year. Likewise, dates without a time are assumed to have the time of midnight

[1] http://dithers.cs.byu.edu/wok/hykss/thesis_experiments/ontology_library/

on that date. Our *DateTime* ontology does not adjust for time zones or daylight savings time (all times are assumed to be in GMT).

The *Distance* ontology recognizes generic distances such as height, width, and length using common units like kilometers, feet, and miles. Recognized distances are canonicalized to measurement in meters. Domain knowledge allows the ontology to recognize the unit "K" as the multiplier 1000. The *Price* ontology currently only recognizes US Dollar amounts and canonicalizes to the same. As such, ambiguous prices in strings such as "Price: 14000" are assumed to refer to US dollars. The *Price* ontology also recognizes and interprets units such as "K" and "grand" to indicate the multiplier 1000. Having more specific domain knowledge also allows methods to be invoked with more intuitive terms such as "after" for *DateTime* or "longer" for *Distance*.

**Level 3: Vehicle Units**. The Vehicle Units ontology library is similar to the generic units ontology library, but moves closer to modeling the target domain of vehicle advertisements. The library consists of three simple ontologies for $VehiclePrice$, $VehicleMileage$, and $VehicleYear$. Having knowledge of the domain allows the ontologies to recognize values that are applicable to vehicles. For example, the Generic Units *DateTime* ontology may recognize the string "1836" whereas the $VehicleYear$ ontology would not recognize this string because it is unlikely to refer to the year of a vehicle.

**Level 4: Vehicle**. The Vehicle ontology library consists of only a single ontology, $Vehicle$. This ontology does not attempt to model the entire vehicle domain but rather has object sets for common concepts of interest such as $Color$, $Make$, $Mileage$, $Model$, $Price$, and $Year$. The $Vehicle$ ontology is similar to the ontology in Figure 2.1, but does not include the $Feature$ object set or its connected relationship set.

The data frames for $Mileage$, $Price$, and $Year$ are similar, but not identical, to the data frames used in the $VehicleMileage$, $VehiclePrice$, and $VehicleYear$ ontologies in the Vehicle Units ontology library. The data frames differ slightly because the object sets in the $Vehicle$ ontology can take advantage of other conceptual knowledge within the ontology. For

example, the $Year$ object set in the $Vehicle$ ontology can take advantage of the fact that a $Year$ value often precedes a $Make$ value, whereas $VehicleYear$ cannot because it has no concept of $Make$.

With a higher level of semantic modeling, the Vehicle ontology library is able to consider larger amounts of context during extraction. As mentioned, having knowledge of multiple concepts within a single ontology can help during the extraction process. In addition to the comparison functionality provided in the Vehicle Units ontology library, the $Vehicle$ ontology provides *canonicalization methods* for common abbreviations (e.g., "chevy" = "chevrolet") and misspellings (e.g., "siverado" = "silverado"). Also, whereas each of the lexical object sets in the ontologies in the previous libraries have no constraint on the number of occurrences they can extract from a document, each lexical object set in the $Vehicle$ ontology has a limit of one extracted instance, meaning that only one $Color$, $Make$, $Mileage$, $Model$, $Price$, and $Year$ can be extracted from a document.

**Level 5: Vehicle+.** The final level of semantic modeling is the Vehicle+ ontology library. The Vehicle+ ontology library contains the $Vehicle$ ontology from the previous ontology library, as well as five new ontologies: $GermanVehicle$, $HyrbridVehicle$, $JapaneseVehicle$, $SportsVehicle$, and $VehicleType$. These additional ontologies are small simple ontologies allowing for more fine tuned extraction and querying in the vehicle domain. The $GermanVehicle$ and $JapaneseVehicle$ ontologies extract vehicle makes and use the *canonicalization method* to determine if they are of the type specified. They can also extract terms such as "german car" and "japanese car," respectively in order to make the determination. The $HybridVehicle$ and $SportsVehicle$ ontologies function similarly but make use of vehicle models rather than makes. The $VehicleType$ ontology also relies on vehicle models, but instead canonicalizes to "car", "truck", "suv", or "van". These additional ontologies are valuable in understanding higher level queries such as "find me a black truck" or "list Japanese cars".

## 5.2 Metric

The primary focus of HyKSS is to retrieve relevant documents. As such, we use mean average precision (MAP) [MRS08], a common information retrieval metric to evaluate the quality of the system. MAP is a commonly used metric that provides a single-figure measurement of retrieval quality with a focus on the ranking of documents.

Average precision is the key computation in MAP. In terms of average precision, a retrieval system is perfect for a given query if it returns all $N$ relevant documents for that query in the top $N$ ranking positions. If irrelevant documents occur in the ranking before a relevant document the score is penalized. Average precision is computed by starting at the top of the ranking and moving down, computing the precision level at each relevant document in the ranking, and then computing the average of these precision values. Any relevant documents that are not included in the ranking are considered to have a precision level of zero and are included in the average. Average precision includes both a precision component, because it averages precision levels, and a recall component, because precision with respect to the ranking is measured for every relevant document. Average precision can be seen as the estimation of the area under an uninterpolated precision-recall curve.

Mean average precision is the mean of each average precision computation across all queries. Mathematically, MAP is computed according the following formula: $MAP(Q) = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{m_j}\sum_{k=1}^{m_j} Precision(R_{jk})$, where $Q$ is the set of queries, $m_j$ is the number of relevant documents in the collection for query $j$, and $Precision(R_{jk})$ is the precision of the $k$th relevant document for query $j$. The $\frac{1}{m_j}\sum_{k=1}^{m_j} Precision(R_{jk})$ portion of the formula computes the average precision for a single query while the preceding portion is averaged across the queries. We exclude queries with no relevant results when calculating mean average precision.

### 5.3 Document and Query Sets

HyKSS is currently designed to work over topical documents where each ontology is applied once per document. In choosing document sets for experimentation we sought to locate document collections of this nature with corresponding query sets containing comparison constraints. In particular, queries over product pages appeared to be a natural application for HyKSS. However, many query sets in this domain are drawn from proprietary search engines (e.g., [LWA09, SPT10]) and do not appear to be publicly available.

In the end we were not able to locate ready-made document and corresponding query sets that met our needs. The problem of finding desirable document and query sets for systems involving semantic search components is not unique to this work. The authors of [FLS+08] point out that "the semantic web community is still a long way from defining standard evaluation benchmarks that comprise all the required information to judge the quality of current semantic search methods."

Given this difficulty we opted to construct our own document and query sets in the vehicle advertisement domain. We first obtained a query set from previous efforts involving AskOntos [Vic06]. These efforts presented an online demo[2] of the system to two different classes of database students. The students were asked to generate two queries they felt the system interpreted correctly and two queries they felt the system misinterpreted, but that the system should have handled correctly.

The students generated 137 syntactically unique queries (syntactic duplicates were removed). For our purposes, we then removed queries that the free-form query interface of HyKSS is not designed to handle. These queries included features such as negations, disjunctions, and aggregations. We also removed queries when the initial intent was ambiguous, the query could not be objectively evaluated, or the query could not have relevant results in the document set eventually used in our experiments. A total of 24 queries were removed[3]

[2] http://www.deg.byu.edu/demos/askontos/

[3] See http://dithers.cs.byu.edu/wok/hykss/thesis_experiments/queries/removed_vehicle_training_queries.txt

leaving the final query set with 113 queries[4]. These 113 queries constitute our training query set.

We posed a similar task to a class of computational linguistic students. These students were not presented the online demo, but were shown a few example queries and asked to submit hand written queries they felt a semantic system like AskOntos could handle. This exercise resulted in 71 unique queries from which we removed 11 queries[5] using the previously mentioned criteria for removal.[6] This left us with a final blind query set of 60 queries.[7]

For document sets we chose to use vehicle advertisements posted on local craigslist sites.[8] Craigslist allows users to post classified advertisements that consist of free-text descriptions and a small number of semantic fields. However, users occasionally misuse these fields and enter incorrect or irrelevant information. We gathered a total of 250 vehicle advertisements from the "for sale by owner" sections under the "car+trucks" headings of the craigslist sites. We then divided these advertisements into training (100), validation (50), and test (100) document sets. The training set includes documents from the Provo craigslist site while the the validation and test sets are from the Salt Lake City craigslist site. The document sets may contain duplicate or similar advertisements due to cross posting and re-posting tendencies of craigslist users.

We also gathered additional topical documents not in the vehicle advertisement domain to use as noise in the experiments. These additional documents include 318 mountain pages and 66 roller coaster pages from Wikipedia,[9] and 88 video game advertisements from Provo's craigslist site. We gathered the mountain pages manually by downloading a subset of pages linked from the list-of-mountains page.[10] In creating the subset we attempted to avoid pages

[4]See http://dithers.cs.byu.edu/wok/hykss/thesis_experiments/queries/vehicle_training_queries.txt

[5]See http://dithers.cs.byu.edu/wok/hykss/thesis_experiments/queries/removed_vehicle_blind_queries.txt

[6]We removed one of the eleven queries because it resulted in fatal query execution at experimentation time. However, this query did not have any relevant documents in the test document set and thus does not affect MAP scores.

[7]See http://dithers.cs.byu.edu/wok/hykss/thesis_experiments/queries/vehicle_blind_queries.txt

[8]http://provo.craigslist.org/ and http://saltlakecity.craigslist.org/

[9]http://www.wikipedia.org/

[10]http://en.wikipedia.org/wiki/List_of_mountains

that referred to a mountain range and chose a selection of pages about single mountains. The roller coaster pages were downloaded using the links from the list-of-roller-coaster-rankings page.[11] One of roller coasters, Hades, linked to page about the Greek god rather than the roller coaster, but we chose to leave it in the document set because it represents real-world noise. Pages with the same URL were only included once.

### 5.4 Annotation and Tuning

The decision to create our own document and query sets also required that we create gold standard annotations for the data. A single annotator created these annotations for expected extraction from documents and queries, as well as for query-document relevance. For the former task, the annotator often semi-automatically generated the annotations using extraction ontologies, and then checked and altered the annotations manually where needed.

We began the tuning process by ensuring that each ontology library extracted properly from and generated proper interpretations for the queries in the training query set. Specifically, for each query we ensured that each ontology library, as a whole, recognized expected semantic constraints and generated the expected generic semantic query. This required ontology libraries with multiple ontologies, such as the Vehicle Units ontology library, to generate appropriate cross-ontology queries where necessary. We measured query interpretation accuracy in a binary manner, claiming that an interpretation was correct if it was an exact match and incorrect otherwise. The Generic Units and Vehicle Units ontology libraries achieved 98% interpretation accuracy and the other libraries achieved 100% query accuracy over the training query set.

We continued the tuning process by ensuring that the ontology libraries extracted properly from documents in the training document set. The gold standard annotations for extraction from documents do not form a true gold standard over all ontological information, but rather form a standard for the information we expect OntoES to be able to extract. For

[11] http://en.wikipedia.org/wiki/List_of_roller_coaster_rankings

example, an image in a advertisement may indicate that the color of a vehicle is green, but we do not expect OntoES to extract this information without textual cues, and thus the color green is not included in the expected extraction annotations for that document. Precision of the extraction processes for each ontology library ranged from 97% to 100%, and recall ranged from 94% to 100% according to our expected annotations. The only ontology library to achieve 100% in either metric was the Number ontology library.

After tuning the ontology libraries with respect to the training data we locked them from further modification (with a small exception described hereafter) before continuing with the necessary annotations for non-training data. By doing so we hoped to avoid validation and test data having an influence on our ontologies in HyKSS. Query processing algorithm development continued beyond this point, but not with respect to the test document set. We proceeded next by creating expected extraction annotations for the validation document set with respect to the Vehicle+ ontology library. We measured extraction precision and recall and found that recall fell from 99% in the training document set to 94% in the validation document set and that precision fell from 98% to 90%. Therefore, we expect a similar decline in extraction quality when using the test document set.

With ontology tuning complete, we next constructed query-document relevance annotations for the training query set relative to the blind document set, and later did the same for the blind query set relative to the blind document set. These annotations are the annotations used to evaluate the actual retrieval quality of HyKSS. The construction of query-document relevance annotations is a difficult task that can often be subjective. In order to remain as objective as possible, and to model the domain to the best of our ability, we used a closed-world assumption in assigning relevance; a document is only relevant to a query if it explicitly satisfies all constraints in the query. The extracted semantics for a document must satisfy all selections and have some value for each projection in order to be considered relevant. While this relevance annotation approach may seem contrary to our ranking approach, we believe that this approach is appropriate for this domain for two

reasons. First, users are less likely to respond to a product advertisement if some of their constraints are not met in the product page. Second, used vehicles are sold in a wide array of conditions, and even those features that come standard with the vehicle are not guaranteed to work as originally constituted.

In constructing these annotations we used all information available in the advertisements. This includes visual information that OntoES is currently incapable of extracting. For example, many vehicle advertisements do not specify the number of doors on or the color of a vehicle, but include a picture of the vehicle indicating the values for these attributes. All of this information is used in determining query-document relevance.

Due to the difficulty of annotation we did not attempt to infer information not available in the advertisement from information present. For example, if a advertisement listed the trim of the vehicle we did not try to infer the number of doors. One exception to this is that we did infer the Make of a vehicle given a Model due to its relative ease. We also took advantage of human intuition to infer the meaning of phrases like "fully loaded" to indicate that it had standard features, such as air conditioning and a CD player, occasionally requested in user queries.

In order to experiment with queries over a non-target domain we next expanded the extraction capabilities of our Generic Units ontology library. We selected a couple of pages from the mountain, roller coaster, and video game documents and used them to tune the ontologies in the Generic Units ontology library. The tuning process for these extractions was much less formal and systematic than the tuning process for ontology libraries over the vehicle advertisements. During this exercise we discovered need to handle non-ASCII spaces in some documents, and updated all ontologies in all the libraries to do so (comparison constraints were not updated because we did not anticipate the need to handle non-ASCII spaces in queries). Other than this change, all modifications during this phase of tuning were limited to the Generic Units ontology library. These modifications are the exception to the ontology locking referred to previously.

In developing the HyKSS algorithms we attempted to avoid being influenced by the test document set. We did this first, as mentioned previously, by locking the ontologies, with the one exception, before annotation of the blind document set. The test document set was reserved for experimentation until after the completion of algorithmic development. However, when initial experimentation began we discovered a number of issues resulting in fatal errors to query execution. We also discovered that our original semantic ranking algorithm was conceptually infeasible for large document sets. As necessity required, we made modifications to our algorithms in order to resolve these issues. However, in so doing, we did not tune performance to the document test set in terms of mean average precision. We simply made the modifications and ran the experiments. In this manner we attempted to keep the document test set as blind as possible given the circumstances. All of these modifications were made prior to any experimentation using the blind test query set.

The dynamic query weighting algorithm presented in this thesis was also added during this algorithm modification stage. We initially developed two variants of this algorithm which performed similarly on the validation data. There were tradeoffs in performance depending on the ontology set being used. We could have used both of these variants in all experiments, but instead opted to only report on the method we felt performed best when using both the training query set and test document set at that point in the experimentation.

### 5.5 Experiments and Results

We used the constructed query-document annotations to conduct a number of experiments and evaluate the retrieval quality of HyKSS. For each experiment we tested a number of different processing and ranking approaches for comparison. We describe these differing approaches below.

- **Keyword**: Lucene processes queries after non-phrase stopwords are removed and ranks the results.

- **Keyword - Pre-processing**: Lucene processes queries after comparison constraints and non-phrase stopwords are removed and ranks the results.
- **HyKSS - Set Weights**: HyKSS processes queries and ranks results using set weights for interpolating keyword and semantic scores. These set weights are determined using the validation data as described later.
- **HyKSS - Dynamic Weights**: HyKSS processes queries and ranks results using the dynamic ranking scheme discussed earlier.
- **HyKSS - Single Ontology**: HyKSS processes queries and ranks results using the dynamic ranking scheme discussed earlier except that instead of considering ontology sets HyKSS only uses the single best matching ontology.
- **Keyword - Soft Semantics**: Lucene uses keyword search (after keyword query pre-processing) to rank the results returned using a soft semantic filter. A soft semantic filter uses the open world assumption.
- **Keyword - Hard Semantics**: Lucene uses keyword search (after keyword query pre-processing) to rank the results returned using a hard semantic filter. A hard semantic filter uses the closed world assumption.
- **Soft Semantic Ranking**: The semantic ranking mechanism of HyKSS ranks results that pass a soft semantic filter.
- **Soft Semantics**: This approach retrieves all results that pass a soft semantic filter. No ranking mechanism is provided, and an arbitrary ordering of the results is used in the MAP calculation.
- **Hard Semantics**: This approach retrieves all results that pass a hard semantic filter. No ranking mechanism is provided, and an arbitrary ordering of the results is used in the MAP calculation.

For the "HyKSS - Set Weights" ranking strategy we estimated the keyword and semantic weights that maximized mean average precision over the validation data. To find

| | Number | Generic Units | Vehicle Units | Vehicle | Vehicle+ |
|---|---|---|---|---|---|
| Keyword Weight | 0.71 | 0.71 | 0.71 | 0.56 | 0.56 |
| MAP(Set Weights) | 0.44782 | 0.60031 | 0.62594 | 0.70894 | 0.74966 |
| MAP(Dynamic Weights) | 0.44782 | 0.60031 | 0.62594 | 0.70889 | 0.74960 |

Table 5.1: The results of computing the best preset weights for HyKSS on the validation data. The MAP is also shown for HyKSS dynamic approach.

weight values, we executed each query in the training query set using 100 weight combinations and chose the best. The keyword weight started at 1.0 and dropped in increments of 0.01 until the keyword weight reached 0.0. The semantic weight was one minus the keyword weight. After each weight change the MAP was computed using the returned results. In the event that two MAP scores tied, we preferred the weight combination with the higher keyword score in hopes that it would generalize better to unseen data. Table 5.1 shows the results of the weight tuning. The table includes the MAP scores when using the "HyKSS - Dynamic Weights" scheme as well.

With the approaches for experimentation fully established we moved on to the experiments. The first two experiments were restricted entirely to the vehicle advertisement domain. In each of these experiments, we executed the previously described ranking approaches over the test set of vehicle advertisements. For queries, the first experiment used the training query set while the second experiment used the blind test query set. Tables 5.2 and 5.3 show the results of these experiments, respectively.[12] To assist in readability, each column shows the best MAP score in **bold** and the second best score in *italics*. Of the 113 queries in the training query set, 76 queries had relevant documents in the test document collection. Only 13 of the 60 queries in the blind test query set had relevant documents in the test document collection.

To further test the capabilities of HyKSS we added noise to the document set used in the original experiments. This additional noise consisted of the Wikipedia and craigslist

[12]Due to the arbitrary manner in which the ranking approaches break ties, running the experiments on different hardware may result in different MAP scores. However, in our experience the general trends in the results remain the same.

| Ranking/Library | Number | Generic Units | Vehicle Units | Vehicle | Vehicle+ |
|---|---|---|---|---|---|
| Keyword | 0.35251 | 0.35251 | 0.35251 | 0.35251 | 0.35251 |
| Keyword - Pre-processing | 0.35304 | 0.36651 | 0.36736 | 0.36736 | 0.36736 |
| HyKSS - Set Weights | *0.40126* | *0.50700* | *0.53420* | **0.63764** | *0.74562* |
| HyKSS - Dynamic Weights | **0.40138** | **0.52445** | **0.54893** | *0.63730* | **0.74568** |
| HyKSS - Single Ontology | **0.40138** | 0.48453 | 0.52112 | *0.63730* | 0.62422 |
| Keyword - Soft Semantics | 0.33105 | 0.41544 | 0.45212 | 0.52434 | 0.62651 |
| Keyword - Hard Semantics | 0.28177 | 0.41544 | 0.45212 | 0.51363 | 0.61164 |
| Soft Semantic Ranking | 0.15626 | 0.23804 | 0.25876 | 0.53875 | 0.65750 |
| Soft Semantics | 0.16997 | 0.22059 | 0.24733 | 0.29182 | 0.39631 |
| Hard Semantics | 0.12748 | 0.23804 | 0.25876 | 0.47287 | 0.56212 |

Table 5.2: The results of the various ranking approaches using the training query set and test document set for the vehicle advertisement domain.

| Ranking/Library | Number | Generic Units | Vehicle Units | Vehicle | Vehicle+ |
|---|---|---|---|---|---|
| Keyword | 0.18886 | 0.18886 | 0.18886 | 0.18886 | 0.18886 |
| Keyword - Pre-processing | *0.18902* | *0.19032* | *0.19032* | 0.19032 | 0.19032 |
| HyKSS - Set Weights | **0.22366** | **0.24453** | **0.25741** | **0.36676** | **0.40642** |
| HyKSS - Dynamic Weights | **0.22366** | **0.24453** | **0.25741** | *0.36542* | *0.40530* |
| HyKSS - Single Ontology | **0.22366** | **0.24453** | **0.25741** | *0.36542* | 0.39609 |
| Keyword - Soft Semantics | 0.08563 | 0.07337 | 0.08541 | 0.21000 | 0.26961 |
| Keyword - Hard Semantics | 0.02858 | 0.07337 | 0.08541 | 0.26281 | 0.32101 |
| Soft Semantic Ranking | 0.08815 | 0.07751 | 0.10899 | 0.28820 | 0.33472 |
| Soft Semantics | 0.07499 | 0.07015 | 0.09515 | 0.14895 | 0.18998 |
| Hard Semantics | 0.04525 | 0.07751 | 0.10899 | 0.26791 | 0.30681 |

Table 5.3: The results of the various ranking approaches using the blind test query set and test document set for the vehicle advertisement domain.

video game advertisements discussed earlier. The experiments used the same query sets and query-document relevance annotations as in the first two experiments. Some of the queries, such as "<100K miles", did not actually contain enough information to restrict relevant results to the vehicle domain, but we were aware of the context in which the query was originally posed and thus restricted relevant results to those in the vehicle domain. Tables 5.4 and 5.5 shows the results of these two additional experiments. Again, the **bolded** value in each column indicates the method with the highest MAP score and the *italicized* value indicates the method that performed second best.

| Ranking/Library | Number | Generic Units | Vehicle Units | Vehicle | Vehicle+ |
|---|---|---|---|---|---|
| Keyword | *0.29786* | 0.29786 | 0.29786 | 0.29786 | 0.29786 |
| Keyword - Pre-processing | **0.30113** | 0.31619 | 0.31691 | 0.31691 | 0.31691 |
| HyKSS - Set Weights | 0.28716 | *0.42531* | *0.48898* | **0.57338** | *0.66432* |
| HyKSS - Dynamic Weights | 0.28743 | **0.44852** | **0.51148** | *0.57155* | **0.66832** |
| HyKSS - Single Ontology | 0.28743 | 0.39027 | 0.46500 | *0.57155* | 0.53705 |
| Keyword - Soft Semantics | 0.24660 | 0.35210 | 0.42125 | 0.37945 | 0.46692 |
| Keyword - Hard Semantics | 0.20115 | 0.35210 | 0.42125 | 0.48915 | 0.58546 |
| Soft Semantic Ranking | 0.03680 | 0.13720 | 0.18698 | 0.46078 | 0.56523 |
| Soft Semantics | 0.04272 | 0.11664 | 0.17556 | 0.04568 | 0.10199 |
| Hard Semantics | 0.02625 | 0.13720 | 0.18698 | 0.39721 | 0.48507 |

Table 5.4: The results of the various ranking approaches using the training query set and test document set for vehicle advertisement along with additional document noise.

| Ranking/Library | Number | Generic Units | Vehicle Units | Vehicle | Vehicle+ |
|---|---|---|---|---|---|
| Keyword | 0.16657 | 0.16657 | 0.16657 | 0.16657 | 0.16657 |
| Keyword - Pre-processing | *0.16660* | *0.16935* | *0.16935* | 0.16935 | 0.16935 |
| HyKSS - Set Weights | **0.17071** | **0.19055** | **0.20089** | **0.27081** | *0.29899* |
| HyKSS - Dynamic Weights | **0.17071** | **0.19055** | **0.20089** | *0.26962* | **0.30194** |
| HyKSS - Single Ontology | **0.17071** | **0.19055** | **0.20089** | *0.26962* | 0.29729 |
| Keyword - Soft Semantics | 0.05886 | 0.03585 | 0.04581 | 0.10516 | 0.15641 |
| Keyword - Hard Semantics | 0.02459 | 0.03585 | 0.04581 | 0.19787 | 0.25229 |
| Soft Semantic Ranking | 0.01502 | 0.03740 | 0.06698 | 0.20562 | 0.23097 |
| Soft Semantics | 0.05479 | 0.04177 | 0.06501 | 0.02080 | 0.04350 |
| Hard Semantics | 0.00208 | 0.03506 | 0.06698 | 0.18904 | 0.21058 |

Table 5.5: The results of the various ranking approaches using the blind test query set and test document set for vehicle advertisement along with additional document noise.

For the final experiment we evaluated how well HyKSS performs when searching for documents not in the target domain. Our goal with this experiment was to provide an exploratory evaluation of how HyKSS functions when ontology libraries have not been explicitly tuned towards a domain of interest. We created five queries and ensured that the Generic Units ontology library could interpret them correctly. These queries were targeted towards the mountain, roller-coaster, and video-game domains in the noisy document collection. We designed these queries to be similar in construction to many of the queries we gathered from students. Figure 5.1 shows our queries for this experiment. We generated query-document relevance annotations for these queries using the list-of-mountains and list-

of-roller-coaster-rankings pages as guides. Only the query-document relevance annotations for the video-game query required us to look through the document collection. The document collection for this experiment consisted of both the noisy document set and the test document set for vehicle advertisements. Table 5.6 shows the results for this experiment.

1) mountains over 8000 meters in elevation
2) steel roller coasters over 122m tall
3) ps3 games under $30
4) coaster with a drop of more than 300 feet
5) coasters longer than 2000 meters

Figure 5.1: The queries used to test HyKSS in domains that semantic modeling does not target.

| Ranking/Library | Generic Units |
|---|---|
| Keyword | 0.22043 |
| Keyword - Pre-processing | 0.13632 |
| HyKSS - Set Weights | 0.24190 |
| HyKSS - Dynamic Weights | *0.24693* |
| HyKSS - Single Ontology | *0.24693* |
| Keyword - Soft Semantics | **0.24877** |
| Keyword - Hard Semantics | **0.24877** |
| Soft Semantic Ranking | 0.05686 |
| Soft Semantics | 0.10834 |
| Hard Semantics | 0.05686 |

Table 5.6: The results of using the Generic Units ontology with queries that do not target the vehicle-advertisement domain.

Although mean average precision, and not query execution speed, is the focus of this thesis, we present in Table 5.7 the query execution times for the first experiment (using the training query set over the blind document set without additional noise). When measuring MAP we would often break the experiment apart and use a subset of the ontology libraries to generate the semantic index for that experiment portion. This does not affect MAP scores as the queries generated by a specific ontology library can only bind to indexed annotations generated by that ontology library. However, the size of the semantic index does influence

| Ranking/Library | Number | Generic Units | Vehicle Units | Vehicle | Vehicle+ |
|---|---|---|---|---|---|
| Keyword | 0.0028 | 0.0009 | 0.0009 | 0.0008 | 0.0008 |
| Keyword - Pre-processing | 0.0015 | 0.0009 | 0.0008 | 0.0008 | 0.0008 |
| HyKSS - Set Weights | 0.1011 | 2.8681 | 2.0313 | 0.0762 | 1.4828 |
| HyKSS - Dynamic Weights | 0.0942 | 2.8744 | 2.0344 | 0.0610 | 1.5009 |
| HyKSS - Single Ontology | 0.0907 | 0.0524 | 0.0503 | 0.0605 | 0.0699 |
| Keyword - Soft Semantics | 0.0898 | 2.8864 | 2.0050 | 0.0601 | 1.5055 |
| Keyword - Hard Semantics | 0.0709 | 2.1264 | 1.4650 | 0.0585 | 0.7223 |
| Soft Semantic Ranking | 0.0919 | 2.8751 | 1.9892 | 0.0709 | 1.4771 |
| Soft Semantics | 0.0974 | 2.8573 | 2.0136 | 0.0651 | 1.4846 |
| Hard Semantics | 0.0703 | 2.1722 | 1.4969 | 0.0579 | 0.7208 |

Table 5.7: The execution times in seconds (rounded to the nearest millisecond) for each ranking approach for the experiment using the training query set over the blind document set. These execution times correspond to the MAP scores for the first experiment in Table 5.2.

query execution speed. When measuring the query execution times present in Table 5.7 our semantic index included the annotations generated by all five ontology libraries.

The table shows the average execution time in seconds for each query in the training query set. The average includes all queries in the query set, not just those with relevant documents in the document set. The execution time includes time required for query evaluation as well as any time required by the system for output messaging. Query execution times vary slightly across each running of the experiments but the general patterns remain the same.

## 5.6 Discussion

The results of the experiments are quite favorable towards HyKSS and hybrid search in general. Across all experiments, HyKSS generally outscored the other approaches in terms of mean average precision. The other hybrid approaches using keyword search ranking with semantic filters typically followed close behind.

In Table 5.1, we were quite surprised to find that the Number, Generic Units, and Vehicle Units ontology libraries all received the same estimation for keyword weight to maximize MAP. We expected this value to be highest for the Number ontology library and

to gradually decrease as the level of semantic modeling increases. This decrease occurs as the sliding scale moves from the Vehicle Units ontology library to the Vehicle ontology library. Thus, there is some indication that the importance of the keyword score decreases once semantic modeling passes certain thresholds. What those thresholds are, however, remains unclear.

As expected, Table 5.1 also shows that MAP scores increase as the level of semantic modeling increases. This helps demonstrate the pay-as-you-go nature of HyKSS. Users will likely see an increase in retrieval performance if they are willing to put in time and effort to create higher levels of semantic modeling. The other result of interest is that the dynamic weighting approach performs essentially the same as using the preset weights for each ontology library. This is significant as the dynamic approach does not require document annotation effort by the user to compute weights.

The results of the first experiment (Table 5.2), using the training query set over the blind document set, yields several interesting discussion points. First, it appears that removing comparison constraints from keyword queries provides only a small benefit in retrieval performance. Second, the pay-as-you-go nature of HyKSS is reaffirmed as all ranking methods with a semantic component improve as the level of semantic modeling increases. The most interesting result, however, is that hybrid search, and especially HyKSS, outperforms all the other methods. For each ontology library, either "HyKSS - Static Weights" or "HyKSS - Dynamic Weights" obtained the highest MAP score. In two instances, the "HyKSS - Single Ontology" approach obtained the same MAP score as the "HyKSS - Dynamic Weights" approach. However, these two instances are when using the Number and Vehicle ontology libraries which consist of a single ontology. In all other cases, using multiple ontologies outperforms choosing and using the single best ontology.

Beyond the HyKSS based methods, the hybrid approaches using keyword ranking with a semantic filter consistently outperformed other methods except at the lowest and highest levels of semantic modeling. At the lowest level of semantic modeling, the Number

ontology library, semantics do not appear to be very useful. Numbers are present in many documents and knowing that a document contains a number less than $N$ does not add much value to the search. At the highest level of semantic modeling, the Vehicle+ ontology library, semantics are extremely useful, and using semantics alone yields positive results, but still does not perform as well as HyKSS. However, using soft semantics with no ranking performs relatively poorly. This is because no ranking is provided and documents that passed the filter using the open world assumption can be in an arbitrary position in the ranking. Ranking the soft semantic results or using a hard semantic filter yields much higher MAP scores.

Using this same query set over the blind document set with additional noise (Table 5.4) confirms many of the results discovered in the first experiment. However, the addition of noise also results in consistently lower MAP scores. The noise had other affects on the results as well. At the lowest level of semantic modeling, the Number ontology library, the "Keyword - Pre-processing" approach outperformed all other ranking approaches. The addition of so many documents containing a wide variety of numerical values likely rendered the low level of semantics largely useless. As the semantic modeling increased the hybrid approaches began to outperform other approaches. Again, using soft semantics without ranking performed poorly relative to the other methods. The inclusion of additional noise made this approach unusable to the point that it was less effective than keyword search at even the highest level of semantic modeling.

The experiments involving the blind test query set (Tables 5.3 and 5.5) also reveal interesting results. The HyKSS approaches outperform all other approaches for all ontology libraries. However, when using the blind test query set over the test vehicle document set (Table 5.3) the HyKSS approach using a single ontology performed as well as the multiple ontology approach at all but the highest level of semantic modeling. As using a single ontology in HyKSS results in much faster query execution there may be trade-offs to consider in this regard for a real-world system.

The final experiment using vehicle-based queries, using the blind test query set over the test document set with noise (Table 5.5) confirms the same trends as the other experiments. This experiment has, by far, the lowest MAP scores of any of the vehicle query experiments. This is expected because both the query set and document set were blind to the ontology designers, and the document set intentionally contains additional noise. HyKSS once again outperforms all other methods although the margins are not as large as in the previous experiments.

Our last experiment, using five queries targeting domains in the noisy document set (Table 5.6), gives exploratory insights into using HyKSS for non-targeted domains. Hybrid methods once again outperformed using keyword or semantic search in isolation, but HyKSS did not receive the highest MAP score. The hybrid approaches using keyword ranking and semantic filters outperformed HyKSS. In fact, although not reported in the table, using keyword ranking without removing comparison constraints significantly outperformed all other methods (0.39433 for both soft and hard filters). We did not generally report on this approach because it was generally comparable to the approach removing constraints; almost always the same or a little better, a little worse in a few cases. Further inspection is needed to discover why leaving comparison constraints in these queries made such a dramatic difference. Intuitively, it seems that the presence of comparison-constraint words helped lead keyword search to relevant documents. (The keyword "over", for example, may commonly occur in mountain pages, and "longer than" in roller coaster pages. These words indicate bigger and better—possibly a common theme for mountains and roller coasters.) There are not enough queries in this experiment to be convincing, but it may indicate that keyword ranking with a filter may be superior for non-target domains. More experimentation is needed to draw any meaningful conclusions.

To further validate our claim that HyKSS outperforms the other ranking approaches we tested for statistically significant differences in their performances. Specifically we test for statistically significant differences between ranking approaches across the various document

sets, query sets, and semantic modeling levels used in our experiments. Our experiments supply us with 21 mean average precision data points for each ranking approach (five each for the first four experiments and one for the last experiment). We use the Student's paired t-test method (see [SAC07]) for computing statistical significance. This approach computes the probability that two test subjects, ranking approaches in our case, are statistically identical according to some normal distribution. If the probability of being the same is low enough, we consider the difference between the subjects to be statistically significant.

We found that there are statistically significant differences between the HyKSS-based ranking approaches and most other ranking approaches. Specifically, our results confirm that the three HyKSS methods outperform the "Keyword", "Keyword - Query Pre-processing", "Soft Semantic Ranking", "Hard Semantics", and "Soft Semantic" methods. The difference is statistically significant for $p < 0.005$. We also found that there is a statistically significant difference between the HyKSS methods and the "Keyword - Soft Semantics" and "Keyword - Hard Semantics" methods for $p < 0.05$. Our results also indicate that there is not a statistically significant difference between the three HyKSS methods. This certainly favors the dynamic weight approach to the set weight approach because it requires less effort from users. However, it also favors using a single ontology due to the runtime efficiency of not considering cross-ontology queries. It should be noted, however, that 4 of the 21 data points for each approach are using ontology libraries containing only a single ontology. This may or may not have affected the outcome of the significance testing. Also of note, we found that removing comparison constraints from keyword queries did not have a statistically significant impact on query performance in terms of mean average precision.

We conclude with a discussion of the query execution times[13] for HyKSS and the other ranking approaches in Table 5.7. As expected, the keyword search based methods significantly outperform the semantic methods in terms of runtime. Keyword search simply requires less processing, and the Lucene library has been constructed for fast keyword query

[13]Our analysis of query execution times only considers the times in Table 5.7 and those discussed in our analysis. We expect similar patterns to hold across the other experiments.

execution. When using ontology libraries containing a single ontology library the HyKSS and semantic methods all performed fairly well, maintaining execution times of under a second. Future work will need to determine how execution times are affected by larger document sets and larger ontology libraries, and how to scale the system to keep response times reasonable.

Methods that cross ontology boundaries, however, performed poorly when using ontology libraries containing multiple ontologies. Response times were as high as just under 3 seconds per query using the Generic Units ontology library. Query execution times were even slower when using the same query set but using the test document collection with additional noise. In this case the average query execution speed for cross-ontology query methods needed to be measured in minutes rather than seconds, even when using only a subset of the libraries to generate the annotations in the index. We also found that using ontologies containing a single lexical object set in cross-ontology queries quickly exhausts large amounts of memory. These are significant performance issues that will need to be resolved in future work if using cross-ontology queries over ontologies is to be viable.[14] This may require HyKSS to alter the structure of the semantic queries it generates. Using HyKSS with a single ontology, however, resulted in reasonable response times across the board. When designing systems of this nature it may be prudent to include larger ontologies that match domains of interest and simply select and use the best one for runtime performance.

[14]Optimization of SPARQL queries is an active area of research and this problem may resolve itself as progress is made. Further, other semantic storage options such as relational database management systems may be more suitable for this type of problem.

# Chapter 6

## Related Work

Search systems are increasingly turning to semantics to improve retrieval performance. Several systems leverage semantics to assist in transforming free-form textual queries into formal structured queries (e.g., [ZWX+07, TCL09]). Some systems take this a step further and execute generated structured queries over underlying annotations (e.g., [Vic06, AME07, LUM06, KKR+06]). The Avatar [KKR+06] system even allows for keyword matching, in addition to semantic matching, over the underlying annotations. Similar systems (e.g., [GMM03, RSA04]) retrieve relevant annotations using submitted keyword queries, but use a process that relies on retrieving instances that match keywords, and then retrieving semantically related instances to the original instance matches. Other systems also allow textual queries over semantic annotations, but require users to submit full natural language queries (e.g., [KBZ06, LPM05]). These systems are extremely useful but are semantic search systems, and are completely dependent on the quality and completeness of underlying annotations (although these annotations can be extracted from unstructured text).

Pure semantic search systems are not likely to perform as well as hybrid search when annotations are imprecise and incomplete. A number of hybrid search systems attempt to combat this weakness by including a textual keyword search component in the search process. OntoSearch [JT06] presents a system based on a spreading activation algorithm, which is an approach for searching networks. OntoSearch begins by executing an input query using keyword search to retrieve an initial set of documents. The underlying annotations for these documents constitute a seed set for the spreading activation algorithm that runs over the

network of annotations. This algorithm infers concepts of relevance from a user's query based upon the seed set. OntoSearch then constructs a query vector using both keyword and concept weights, and compares it against the document collection using the traditional vector space model. This system does not mention the ability to handle cross-ontology queries although it appears likely to be able to do so. Additionally, OntoSearch does not address the issue of handling comparison constraints in queries and does not appear to be equipped to do so. However, this approach does appear to be well suited to handling more traditional keyword-like queries that do not include comparison constraints.

A number of other hybrid search systems are more similar to the retrieval and ranking mechanisms used in HyKSS, but require structured queries rather than free-form queries. The authors of [CFV07] present an adaption of the vector space model for hybrid search. This system weights concepts (annotations) in documents according to a TF/IDF scheme. Users submit structured queries and the system retrieves an initial set of tuples in a Boolean manner. A document retrieval step expands the query using hierarchies and rules to discover all documents related to an initial query. The document vectors are compared to an extended query vector to determine the score and ranking of each document. To compensate for possibly incomplete annotations, a keyword query is generated from the structured input query and the keyword score for documents is combined with the semantic score using a linear interpolation. In this scheme, however, the weights for keywords and semantics are set at 0.5 rather than being dynamic (with special cases for when either of the scores is zero).

K-Search [BCC+08] and GoNTogle [GBDS10] are the two systems most similar to HyKSS. K-Search motivates the need for hybrid search with similar arguments to those in this thesis. The authors of K-Search discuss a generic hybrid search architecture and provide K-Search, a concrete implementation of that architecture. The K-Search interface provides a tree view of an ontology to use for generating structured queries in a form-like manner. These structured queries can include keywords either as a separate query component (as in HyKSS) or within semantic fields (called keyword-in-context). This form-based querying is similar

to the advanced form interface of HyKSS, but does not include negations, and the query is built entirely using the ontology tree view rather than starting with an initial free-form query. By default, results must satisfy a hard semantic filter and are ranked according to keyword score. This is comparable in concept to the "Keyword - Hard Semantics" approach used in our experiments. Users can also change initial rankings by focusing on specific annotation values. The results display of K-Search is similar to HyKSS but has additional features such as aggregation of data into graphs and multiple-color results highlighting.

GoNTogle is a framework for document annotation and retrieval. A hybrid search option is available for performing document retrieval over these annotations. GoNTogle allows users to return the intersection or union of the keyword and semantic search results. For semantic ranking, GoNTogle uses a scheme that considers the number of tokens in the document that the annotation covers, the total number of tokens in the document, and the number of ontology classes used during query execution. The semantic score is combined with the keyword score using a linear interpolation with pre-set weights. Like K-Search, GoNTogle requires a structured form-based interface for the semantic portion of the query. GoNTogle also provides a number of advanced semantic search options such as finding related documents and confining searches to higher or lower level concepts of an ontology.

A different approach to combining keyword and semantic search is presented in [FGGL10]. This work presents a means of leveraging semantic annotations with existing search engines. They present a generalization of the PageRank algorithm, ObjectRank, which allows the ranking process to include objects in addition to pages. In an offline step the system processes ontological reasoning and generates HTML pages for each object. The system can then transform formal structured queries into a sequence of standard web search queries and combine the various results into a single result set. Experiments show that this approach performs extremely well in terms of precision and recall.

HyKSS is different from these hybrid search systems in its ability to handle free-form textual queries in addition to structured queries. Although the authors of [CFV07] mention

that structured queries can be generated from free-form queries, we believe that this task is non-trivial and introduces further ambiguity into the system. Further, structured queries likely allow for the specification of comparison constraints and cross-ontology queries, but require users to explicitly specify necessary constraints. However, these methods do appear more readily equipped to handle operations over large ontologies. HyKSS appears to be the only hybrid search system that focuses on the ability to handle comparison constraints from free-form queries, and one of the few that claims the ability to query over multiple ontologies.

The QUICK [PIW10] system presents another interesting hybrid search approach. QUICK differs from other hybrid search systems because in addition to considering both keywords and semantics, it also allows users to submit queries in a hybrid query language. This query language, referred to as a keyword-based structured query language by the authors, maintains the flexibility of keyword search but allows for the specification of some constraints using a simple syntax. It will be interesting to see if untrained users can adapt to a query paradigm of this nature.

Additional efforts have been placed in making hybrid search systems viable for large-scale applications. One such architecture, $CE^2$ [WTL08], presents a unified framework to represent both RDF data and documents in an integrated way. Preliminary results indicate that hybrid search using this architecture (over millions of triples) shows increased precision over keyword and semantic search in isolation while maintaining acceptable response times. The Semplore [ZLZ+07] system is another effort towards a scalable architecture for hybrid search queries. However, both of these systems require users to submit hybrid queries that are an extension of formal conjunctive queries. Still, work of this nature indicates that hybrid search technology may be viable for large-scale applications.

Most of the hybrid systems discussed do not allow, or at least do not discuss, the ability to handle cross-ontology queries. One system capable of handling cross-ontologies queries is PowerAqua [LUSM09]. PowerAqua accepts a natural language query and can return annotations to answer that query from various public structured knowledge repositories.

In the search for relevant annotations PowerAqua considers multiple ontologies and maps ontologies together when deemed appropriate. The means for mapping ontologies together relies on syntactic label similarities, semantic similarities such as ontology hierarchies and WordNet[1] similarities, and a series of heuristics. PowerAqua, by itself, is a semantic search system. However, recent work [FLS+08] combines PowerAqua with the work of [CFV07] to construct a cross-ontology hybrid search system. The cross-ontology mechanism in HyKSS is, in many ways, simpler than the mechanism used in PowerAqua. Similar to PowerAqua, HyKSS combines ontologies dynamically at runtime in order to answer a user query. In contrast, HyKSS can rely on data frames within extraction ontologies, along with a series of simple heuristics, to determine which ontology sets to generate. PowerAqua is not as fortunate and must rely upon labels and the structure of ontologies without data frames to try to determine similarities. HyKSS also simplifies processing by only allowing recognized text to be claimed by a single ontology in an ontology set and thus eliminating joins. PowerAqua does have an advantage, however, in that it has access to far more ontologies because data frames are not required. Further, PowerAqua appears to be dealing with larger ontologies than HyKSS has been tested with.

We found little work in efforts to handle comparison constraints in free-form queries, and none in the hybrid search arena. Microsoft researchers have done some work on tagging product queries with structural information [LWA09, ML09]. However, these annotations map text to concepts and do not discover comparison constraints for concepts such as price. Further, these efforts focused on the query annotation process and did not experiment with the effect of such annotations on document retrieval. Recently, however, Microsoft announced that its search engine Bing[2] has natural language capabilities on their shopping site that interpret and make use of comparison constraints on price in the search process.[3] (The search

[1] http://wordnet.princeton.edu/

[2] http://www.bing.com/

[3] http://www.bing.com/community/site_blogs/b/search/archive/2011/03/01/bing-feature-update-searching-for-a-good-deal-new-natural-language-capabilities-in-bing-shopping-understand-prices.aspx

algorithm for Bing is proprietary, and we do not have the ability to make direct comparisons with HyKSS.)

We believe that hybrid search is an appropriate component of dataspace support platforms [FHM05, HFM06]. A dataspace, as opposed to a database, uses a data co-existence solution. This means that a dataspace support platform must accept all data and provide functionality over that data regardless of how well integrated the data sources are. A key aspect of this type of system is that the capabilities of the system improve over time. HyKSS provides a pay-as-you-go nature of improvement, but does not yet make improvements automatically or semi-automatically based on usage patterns. While HyKSS is certainly not a dataspace support platform, we expect hybrid search to be a key component of a successful dataspace system.

Faceted search is an increasingly common approach for combining keywords and semantics in the search process. A facet is another term for a semantic category or a dimension of the data. Users locate relevant information in a faceted search environment by using both keywords and facet values to drill down to relevant information. This approach differs from HyKSS and other hybrid search systems in that faceted search generally involves users iteratively drilling down to relevant information rather than executing a single query in a hybrid manner. Facted search interfaces are common on commerce web sites, and a research effort has produced a faceted search interface for Wikipedia [HBS+10].

Finally, it should be mentioned that major search engines are not based entirely upon keyword search. Even PageRank [BP98], though primarily a keyword search algorithm, takes advantage of link structures in hypertext, a form of semantic information. Major search engines are continually searching for new ways to obtain and integrate semantics into their search algorithms to improve the user search experience. Some semantics are discovered through automated processes, such as machine learning, and others are gleaned through annotations created by website providers. However, because these systems are proprietary we are not aware of how hybrid search algorithms are employed or to what extent they are used.

Just recently Google, Bing, and Yahoo[4] announced a combined effort to create and support a common vocabulary for annotating information on web pages.[5] This effort will allow web developers to use a single annotation format to provide valuable semantic information to search engines. While this effort requires annotation by a developer, and search engines will certainly still use automated means for discovering some semantics, their combined efforts demonstrate large-scale interest in integrating semantics into search and providing hybrid search systems.

[4]http://www.yahoo.com/

[5]Posted at http://googleblog.blogspot.com/2011/06/introducing-schemaorg-search-engines.html and elsewhere.

# Chapter 7

# Conclusions and Future Work

## 7.1 Conclusions and Contributions

This thesis introduced HyKSS, our system for *Hy*brid *K*eyword and *S*emantic *S*earch. HyKSS employs both keyword and semantic processing in order to improve retrieval ranking results. The use of keywords and semantics in a hybrid manner helps to mitigate the weaknesses of either approach used in isolation. HyKSS can process free-form textual queries, as well as queries presented through a form interface. Our work on HyKSS demonstrates the potential of hybrid search methods as well as the viability of extraction ontologies as a semantic basis for a hybrid search system.

HyKSS includes several novel techniques for hybrid query processing. The semantic processing and ranking methods are, to our knowledge, unique to HyKSS. We demonstrated a method for dynamically generating ontology sets using data frames available in extraction ontologies. HyKSS is then able to perform queries over multiple ontologies grouped together in an ontology set and thus is not limited to executing queries using only a single ontology. We also provided a simple approach for semantic result ranking based on the amount of requested semantic information a document can supply without violating constraints. Additionally, for hybrid query processing we presented a novel, query-based method for determining the keyword and semantic weights for a given query.

Our experimental results indicate that our hybrid search system outperforms keyword and semantic search used in isolation for most levels of our semantic modeling and for most of our data and query sets. The only exception to this was using a very low level of semantic

modeling over a noisy data set, in which case keyword search (with query pre-processing) outperformed all other methods. The results of hybrid search improved as the level of semantic modeling increased, which supports the pay-as-you-go-nature of HyKSS. Our results also indicate that HyKSS outperforms several other varieties of hybrid search over the data sets used. Moreover, the dynamic query driven weighting approach used by HyKSS performs similarly to using weights tuned with validation data, but does not require manual annotation.

We found that across the query sets, document sets and semantic modeling levels used in our experiments the difference in performance between HyKSS (with dynamic weights) and the non-HyKSS ranking approaches, including keyword, semantic, and a number of hybrid ranking methods, is statistically significant. However, we also found that using multiple ontologies does not hold a statistically significant advantage over selecting and using the best matching ontology. Due to the increase in query execution speed when selecting and using a single ontology the latter approach may hold an advantage for use in real-world systems.

### 7.2 Future Work

Our work on HyKSS provides only an introductory look at using hybrid keyword and semantic search to improve the search experience. There are many lingering questions and issues that should be addressed. Web users are accustomed to providing extremely short keyword queries [JSS00], whereas comparison constraints often require several terms to express. However, there is certainly a training effect that takes place with search engines—we tend not to ask queries we know the system cannot handle. Studies are needed to determine if users will ask queries containing comparison constraints when they know the system can handle them.

Future work could also focus on query response time and scalability. HyKSS needs to have reasonable query response times as both the size of the document collection and the number of ontologies in the library increase. The ability to distribute indexes and ontologies, and parallelize processing where possible is likely key to this effort. Our current semantic

processing, which relies on the open world assumption for semantic ranking, is intuitive, but may be computationally expensive. Other semantic ranking mechanisms may provide more reasonable response times while maintaining retrieval ranking quality. We also plan to explore other processes and heuristics for generating ontology sets. There are additional scalability and processing issues when it comes to handling large ontologies that will need to be explored as well.

Improving retrieval and ranking quality is a continuous effort. A key component of this effort is improving the quality of underlying annotations. Current and planned work on our extraction process should assist in this effort. This research includes data frames for relationship sets, rule execution, and the ability to generate more than one non-lexical object set per document. This last effort will help us test HyKSS on arbitrary web documents rather than restricting ourselves to topical documents. We also plan to explore integrating other extraction tools and knowledge bases. Other methods for weighting the keyword and semantic components at query time also need further exploration and experimentation.

Several usability updates are planned for our HyKSS interface. HyKSS is currently limited to using and displaying only the highest ranked ontology set. We can include a side panel on the results page that will display several top ranked ontology sets. Users can then select different ontology sets to execute the query against. A similar ranking of ontologies can also be added to the advanced search form, allowing users to add and remove ontologies, and thus forms, to and from the set they are working with. The search result page can also be augmented with a list of object sets from the current ontology set so that users can add and remove columns from the results table to more readily see the information they are interested in. These and other improvements can enhance the usability of HyKSS.

Finally, further validation is needed to more fully justify the claims made in this thesis. Larger query sets, preferably from real-world use of semantic or hybrid systems, will provide more insight into the effectiveness of HyKSS. Increasing the size and diversity of document sets and ontology libraries will also be useful in analyzing the use of HyKSS in a

larger system. Scaling up these types of experiments is necessary to determine if HyKSS is viable for real-world use.

## References


[AME07] Muhammed Al-Muhammed and David W. Embley. Ontology-Based Constraint Recognition for Free-Form Service Requests. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE'07)*, pages 366–375, Instanbul, Turkey, April 2007.

[BCC+08] Ravish Bhagdev, Sam Chapman, Fabio Ciravegna, Vitaveska Lanfranchi, and Daniela Petrelli. Hybrid Search: Effectively Combining Keywords and Ontology-Based Searches. In *Proceedings of the 5th European Semantic Web Conference (ESWC'08)*, pages 554–568, Tenerife, Canary Islands, Spain, June 2008.

[BCHS09] Paul Buitelaar, Philipp Cimiano, Peter Haase, and Michael Sintek. Towards Linguistically Grounded Ontologies. In *Proceedings of the 6th European Semantic Web Conference (ESWC'09)*, pages 111–125, Heraklion, Crete, Greece, May/June 2009.

[BP98] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of the 7th International World Wide Web Conference (WWW'98)*, pages 107–117, Brisbane, Australia, April 1998.

[CFV07] Pablo Castells, Miriam Fernandez, and David Vallet. An Adaptation of the Vector-Space Model for Ontology-Based Information Retrieval. *IEEE Transactions on Knowedge and Data Engineering*, 19(2):261–272, February 2007.

[ECJ+99] David W. Embley, Douglas M. Campbell, Yuan S. Jiang, Stephen W. Liddle, Deryle W. Lonsdale, Yiu-Kai Ng, and Randy D. Smith. Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages. *Data & Knowledge Engineering*, 31(3):227–251, November 1999.

[ECSL98] David W. Embley, Douglas M. Campbell, Randy D. Smith, and Stephen W. Liddle. Ontology-Based Extraction and Structuring of Information from Data-Rich Unstructured Documents. In *Proceedings of the Seventh International Conference on Information and Knowledge Management (CIKM'98)*, pages 52–59, Bethesda, Maryland, November 1998.

[EKW92] David W. Embley, Barry D. Kurtz, and Scott N. Woodfield. *Object-Oriented Systems Analysis: A Model-Driven Approach*. Yourdon Press, Upper Saddle River, New Jersey, 1992.

[Emb80] David W. Embley. Programming with Data Frames for Everyday Data Items. In *Proceedings of the American Federation of Information Processing Societies Records National Computer Conference (AFIPS'80)*, pages 301–305, Anaheim, California, May 1980.

[EZ10] David W. Embley and Andrew Zitzelberger. Theoretical Foundations for Enabling a Web of Knowledge. In *Proceedings of the Sixth International Symposium on Foundations of Information and Knowledge Systems (FoIKS'10)*, pages 211–229, Sofia, Bulgaria, February 2010.

[FGGL10] Bettina Fazzinga, Giorgio Gianforme, Georg Gottlob, and Thomas Lukasiewicz. Semantic Web Search Based on Ontological Conjunctive Queries. In *Proceedings of the Sixth International Symposium on Foundations of Information and Knowledge Systems (FoIKS'10)*, pages 153–172, Sofia, Bulgaria, February 2010.

[FHM05] Michael Franklin, Alon Halevy, and David Maier. From Databases to Dataspaces: A New Abstraction for Information Management. *SIGMOD Record*, 34(4):27–33, December 2005.

[FLS+08] Miriam Fernandez, Vanessa Lopez, Marta Sabou, Victoria Uren, David Vallet, Enrico Motta, and Pablo Castells. Semantic Search Meets the Web. In *Proceedings of the Second IEEE International Conference on Semantic Computing (ICSC'08)*, pages 253–260, Santa Clara, California, 2008.

[GBDS10] Giorgos Giannopoulos, Nikos Bikakis, Theodore Dalamagas, and Timos K. Sellis. GoNTogle: A Tool for Semantic Annotation and Search. In *Proceedings of the Seventh European Semantic Web Conference (ESWC'10)*, pages 376–380, May/June 2010.

[GMM03] Ramanathan V. Guha, Rob McCool, and Eric Miller. Semantic search. In *Proceedings of the 12th International World Wide Web Conference (WWW'06)*, pages 700–709, Budapest, Hungary, May 2003.

[GS05] Antonio Gulli and Alessio Signorini. The Indexable Web is More than 11.5 billion Pages. In *Proceedings of the 14th International World Wide Web Conference (WWW'05)*, pages 902–903, Chiba, Japan, May 2005.

[HBS+10] Rasmus Hahn, Christian Bizer, Christopher Sahnwaldt, Christian Herta, Scott Robinson, Michaela Bürgle, Holger Düwiger, and Ulrich Scheel. Faceted Wikipedia Search. In *Proceedings of the 13th Internation Conference on Business Information Systems (BIS'10)*, pages 1–11, Berlin, Germany, May 2010.

[HFM06] Alon Halevy, Michael Franklin, and David Maier. Principles of Dataspace Systems. In *Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'06)*, pages 1–9, Chicago, Illinois, June 2006.

[JSS00] Bernard J. Jansen, Amanda Spink, and Tefko Saracevic. Real Life, Real Users, and Real Needs: A Study and Analysis of User Queries on the Web. *Information Processing and Management*, 36:207–227, January 2000.

[JT06] Xing Jiang and Ah-Hwee Tan. OntoSearch: A Full-Text Search Engine for the Semantic Web. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, pages 1325–1330, Boston, Massachusetts, July 2006.

[KBZ06] Esther Kaufmann, Abraham Bernstein, and Renato Zumstein. Querix: A Natural Language Interface to Query Ontologies Based on Clarification Dialogs. In *5th International Semantic Web Conference (ISWC'06)*, pages 980–981, Athens, Georgia, November 2006.

[KKR+06] 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, Chicago, Illinois, June 2006.

[LPM05] Vanessa Lopez, Michele Pasin, and Enrico Motta. AquaLog: An Ontology-Portable Question Answering System for the Semantic Web. In *Proceedings of the Second European Semantic Web Conference (ESWC'05)*, pages 546–562, Heraklion, Greece, May/June 2005.

[LUM06] Yuangui Lei, Victoria Uren, and Enrico Motta. SemSearch: A Search Engine for the Semantic Web. In *Proceedings of the 15th International Conference on Knowledge Engineering and Knowledge Management Managing Knowledge Patterns (EKAW'06)*, pages 238–245, Podebrady, Czech Republic, October 2006.

[LUSM09] Vanessa Lopez, Victoria Uren, Marta Reka Sabou, and Enrico Motta. Cross Ontology Query Answering on the Semantic Web: An Initial Evaluation. In *Proceedings of the Fifth International Conference on Knowledge Capture (K-CAP'09)*, pages 17–24, Redondo Beach, California, September 2009.

[LWA09] Xiao Li, Ye-Yi Wang, and Alex Acero. Extracting Structured Information from User Queries with Semi-Supervised Conditional Random Fields. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Enformation Retrieval (SIGIR'09)*, pages 572–579, Boston, Massachusetts, 2009.

[ML09] Mehdi Manshadi and Xiao Li. Semantic Tagging of Web Search Queries. In *Proceedings 47th Annual Meeting of the Association of Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL/AFNLP'09)*, pages 861–869, Singapore, August 2009.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, New York, July 2008.

[PIW10] 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/PODS'10)*, pages 423–434, June 2010.

[RSA04] Cristiano Rocha, Daniel Schwabe, and Marcus Poggi Aragao. A Hybrid Approach for Searching in the Semantic Web. In *Proceedings of the 13th International World Wide Web Conference (WWW 2004)*, pages 374–383, New York, New York, May 2004.

[SAC07] Mark D. Smucker, James Allan, and Ben Carterette. A Comparison of Statistical Significance Tests for Information Retrieval Evaluation. In *Proceedings of the Sixteenth Conference on Information and Knowledge Management (CIKM'07)*, pages 623–632, Libson, Portugal, November 2007.

[SPT10] Nikos Sarkas, Stelios Paparizos, and Panayiotis Tsaparas. Structured Annotations of Web Queries. In *Proceedings of the 2010 International Conference on Management of Data (SIGMOD/PODS'10)*, pages 771–782, Indianapolis, Indiana, June 2010.

[TCL09] Aditya Telang, Sharma Chakravarthy, and Chengkai Li. Query-By-Keywords (QBK): Query Formulation Using Semantics and Feedback. In *Proceedings of the 28th International Conference on Conceptual Modeling (ER'09)*, pages 191–204, Gramado, Brazil, November 2009.

[Vic06] Mark Vickers. Ontology-Based Free-From Query Processing for the Semantic Web. Master's thesis, Brigham Young University, Provo, Utah, June 2006.

[WTL08] Haofen Wang, Thanh Tran, and Chang Liu. $CE^2$: 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, Napa Valley, California, October 2008.

[ZLZ+07] 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 *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC'07/ASWC'07)*, pages 652–665, Busan, Korea, November 2007.

[ZWX+07] Qi Zhou, Chong Wang, Miao Xiong, Haofen Wang, and Yong Yu. SPARK: Adapting Keyword Query to Semantic Search. In *Proceedings of the 6th International and 2nd Asian Semantic Web Conference (ISWC/ASWC'07)*, pages 694–707, Busan, Korea, November 2007.