HyKSS: Hybrid Keyword and Semantic Search

Andrew J. Zitzelberger[1], David W. Embley[1],
Stephen W. Liddle[2], and Del T. Scott[3]

[1] Department of Computer Science,
[2] Information Systems Department,
[3] Department of Statistics
Brigham Young University, Provo, Utah 84602, U.S.A.

Abstract. Keyword search suffers from a number of issues: ambiguity, synonymy, and an 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 paper we describe and evaluate HyKSS, a hybrid search system driven by extraction ontologies for both annotation creation and query interpretation. For displaying results, HyKSS uses a dynamic ranking algorithm. We show that over data sets of short topical documents, the HyKSS ranking algorithm outperforms both keyword and semantic search in isolation, as well as a number of other non-HyKSS hybrid approaches to ranking.

1 Introduction

Keyword search for documents on the web works well—often surprisingly well. Can semantic search, added to keyword search, make the search for relevant documents even better? Clearly, the answer should be yes, and researchers are pursuing this initiative (e.g., [1]). The real question, however, is not whether adding semantic search might help, but rather how can we, in a cost-effective way, identify the semantics both in documents in the search space and in the free-form queries users wish to ask.

Keyword search has a number of limitations: (1) *Polysemy*: Ambiguous keywords may result in the retrieval of irrelevant documents. (2) *Synonymy*: Document publishers may use words that are synonymous with, but not identical to, terms in user queries causing relevant documents to be missed. (3) *Constraint satisfaction*: Keyword search is incapable of recognizing semantic constraints. If a query specifies "Hondas for 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 likely do not contain any of these words—not even "Hondas" since the plural is relatively rare in relevant documents.

Semantic search can resolve polysemy by placing words in context, synonymy by allowing for alternatives, and constraint satisfaction by recognizing specified conditions. Thus, for example, semantic search can interpret the query "Hondas for under 12 grand" as indicating interest in retrieving documents advertising a Honda whose selling price is less than $12,000. Unfortunately, correctly interpreting semantics in target documents and user queries is nontrivial, and misinterpretations negatively impact performance.

In this paper, we describe and evaluate *HyKSS* (pronounced "hikes"), a *Hy*brid *K*eyword and *S*emantic *S*earch system that can mitigate the weaknesses of both keyword and semantic search and can capitalize on their strengths.[4] The key component of HyKSS is a collection of conceptualizations called extraction ontologies [3]. An *extraction ontology* binds linguistic recognizers to a conceptual-model instance enabling the model instance to recognize concepts in text—both in target documents and in user free-form queries. Using extraction ontologies to facilitate semantic search and combining semantic search results with keyword search results as we do in HyKSS enables us to make the following contributions:

1. We define HyKSS—a hybrid keyword and semantic search system—and describe our proof-of-concept implementation of a HyKSS prototype.
2. We explain some interesting capabilities of HyKSS, including
   (a) its ability to cross ontology boundaries to satisfy queries and
   (b) its built-in advanced form mechanism to generate forms from ontological descriptions that help users understand what it knows about a domain of discourse and to give users a simple interface to specify more complex queries involving negations and disjunctions.
3. Over data sets of short topical documents that includes pages from craigslist[5] and Wikipedia,[6] we show that HyKSS
   (a) outperforms both keyword and semantic search in isolation,
   (b) is the best performing approach of those we studied for hybrid keyword and semantic search, and
   (c) performs well even with minimal semantic recognition and better as semantic recognition increases.

With respect to item 3(c), a limitation of HyKSS is that it relies on extraction ontologies that perform best over data-rich documents with ontologically narrow domains. Thus, HyKSS tends to reduce to keyword-search performance when search domains are ontologically broad or extraction ontologies are otherwise inadequate to provide good semantic recognition over the target documents.

This paper presents research in the area of data semantics on the web. Stuckenschmidt [4] identifies three major topics within the broader research area of data semantics on the web: namely, (1) extraction of semantics from web data,

[4] In [2], we used HyKSS as our query system to explore cross-language query and search. Here, we focus on HyKSS itself, describing all its features and providing an in-depth statistical evaluation of its performance relative to alternative systems. Except for the necessary background about extraction ontologies and some parts of the running example and its figures, which serve to illustrate the ideas in HyKSS, this paper is new, describing HyKSS and its features in depth and providing a statistical evaluation substantiating our claimed contributions.

[5] `http://www.craigslist.org`

[6] `http://www.wikipedia.org`

(2) representation of semantic information on the web, and (3) use of semantic information to process data on the web. The HyKSS project builds on our earlier work in the arena of extraction [5] (the first theme), uses semantic web technologies to store a semantic index representing structured aspects of source documents (the second theme), and provides a good deal of processing of semantic information to provide enhanced search facilities (the third theme). The bulk of the contributions of HyKSS relate to semantic data processing and thus fit within the third theme.

We present the details of these contributions as follows. In Section 2 we define extraction ontologies. In Section 3 we explain how extraction ontologies interpret queries, including context discovery, the indexing architecture of HyKSS that enables rapid query processing, and the hybrid results ranking algorithm. In Section 4 we describe the experiments we conducted and the results obtained. We compare HyKSS with related work in Section 5 and draw conclusions about the viability of a HyKSS-like system in Section 6.

2 Extraction Ontologies

An *extraction ontology* is a conceptual model augmented linguistically to enable information extraction. The primary components of the model are object sets, relationships sets, constraints, and linguistic recognizers. Figure 1 shows an example of a conceptual model instance for a car-ad application with its object and relationship sets and its constraints, and Figure 2 shows part of two linguistic recognizers—one for *Price* and one for *Make.*

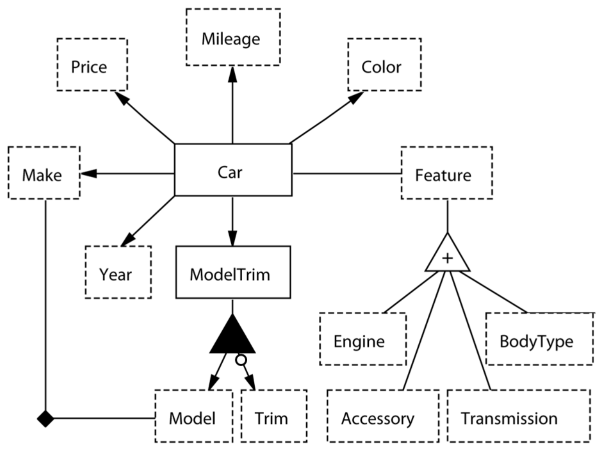


Fig. 1. Car Ad Conceptual Model Instance.

```
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

Make
    ...
    external representation: CarMake.lexicon
    ...
```

Fig. 2. Car Ad Linguistic Recognizer for *Price*.

The conceptual foundation for an extraction ontology is a restricted fragment of first-order logic. Each object set (denoted graphically in Figure 1 by a rectangle) is a one-place predicate. Each predicate has a *lexical* or a *nonlexical* designation: lexical predicates (denoted by dashed-border rectangles) restrict domain-value substitutions to be literals in domain sets, like *String* and *Integer*, or like *Price*, *Year*, *Make*, etc., in Figure 1; nonlexical predicates (denoted by solid-border rectangles) restrict substitutions to be object identifiers that represent real-world objects like people, geopolitical entities, and cars. Each n-ary relationship set (denoted graphically by a line with n object-set connections) is an n-place predicate. In Figure 1, for example, *Car-Color* and *Car-Feature* are binary relationship sets. Notationally, black triangles denote aggregation groupings of relationship sets in an is-part-of hierarchy. In Figure 1, a *ModelTrim* object relates to, and comprises, a *Model* specifier (e.g., "Accord") and a *Trim* designator (e.g., "LX"). Black diamonds on relationship sets denote prepopulated, fixed relationship sets. The *Model-Make* relationship set in Figure 1, for example, associates a *Make* with every *Model*, so that the predicate facts assert, among others, that an "Accord" is a "Honda" and a "Taurus" is a "Ford." Constraints are of three types: (1) Referential integrity requires objects in relationships to be objects in connected object sets. (2) Min-max participation restricts an object's participation in a relationship set. Although actual *min-max* designations may be specified, common participation constraints have graphical notations of two types: (a) arrows for 1-max participation of domain-side object sets making, for example, *Year* functionally dependent on *Car*, and (b) connecting small circles for 0-min participation, making *Trim* optional for a *ModelTrim* object. (3) Generalization/specialization constraints (denoted by a white triangle) form superset/subset relationships among object sets in an is-a hierarchy. Disjoint (+), complete ($\cup$), and partition ($\uplus$) constraints add additional restrictions among specializations. Thus, in Figure 1 *Engine*, *BodyType*, *Accessory*, and *Transmission* are disjoint sets of some, but not necessarily all, features of a car.

Similar to the work by Buitelaar, et al. [6], we linguistically ground a conceptual-model instance, turning an ontological specification into an extraction ontology. Each object set has a *data frame* [7], which is an abstract data type augmented with linguistic recognizers that specify textual patterns for recognizing instance values, applicable operators, and operator parameters. Figure 2 shows part of the data frames for the object sets *Price* and *Make*. Although any kind of textual pattern recognizer is in principle possible, our implementation supports regular expressions as exemplified in *Price* and lexicons as exemplified in *Make*, or combinations of regular expressions and lexicons. Each relationship set may also have a data-frame recognizer. Relationship-set recognizers reference and depend on data-frame recognizers for each of their connected object sets. In addition, relationship sets may be prepopulated with a fixed set of relationships that can provide additional context to aid in linguistic grounding. Thus, in Figure 1, the *Make* "Honda" would be additional context information for *Model* "Accord" in the *Car Ad* ontology.

In a data frame, the **internal representation** clause indicates how the system stores extracted values internally, and the **external representation** clause specifies the instance recognizers. The textual distance of matches from *context keywords* helps determine which match to choose for potentially ambiguous concepts within an ontology. The string "25K", for example, could be a *Mileage* or a *Price* but would be interpreted as a *Price* when in close context to words such as *asking* or *negotiable*. The **units** clause expresses units of measure or value qualifications that help quantify extracted values. 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." The **external representation** clause within a method declaration in Figure 2, for example, recognizes phrases in queries like "under 15 grand". Here, the *p2* within curly braces indicates the expected appearance of a *Price* parameter *p2* for the *LessThan* operator. The **output method** is responsible for displaying internally-stored values to the user in a readable format.

3 Query Processing

Figure 3 illustrates the HyKSS architecture. Before query processing begins, HyKSS preprocesses a document collection and creates a keyword index and a semantic index. For the keyword index, HyKSS uses Lucene,[7] and for the seman-

[7] `http://lucene.apache.org`

tic index, HyKSS uses OntoES (**Onto**logy **E**xtraction **S**ystem).[8] OntoES applies extraction ontologies to text documents to find instance values in the documents with respect to the object and relationship sets in the ontology. The extraction process uses the linguistic recognizers in data frames and the constraints of the conceptual-model structure along with several heuristics to extract instance values. 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 [5]. OntoES returns its semantic index as RDF triples that contain, in addition to the extracted data values, internal values converted to appropriate internal representations by data-frame canonicalization methods, standardized external string representations obtained by data-frame output methods, and information identifying the document and location within the document of extracted text.

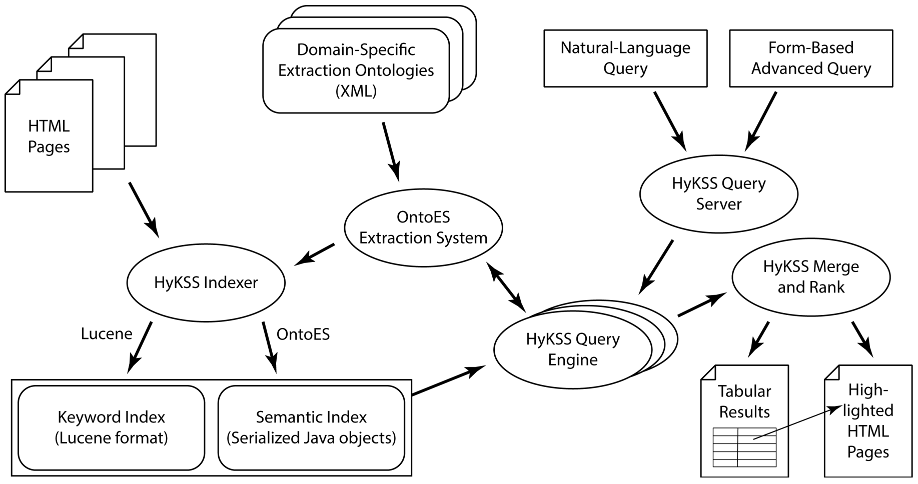


Fig. 3. HyKSS Architecture Overview.

When HyKSS processes queries, it first finds the extraction ontology or set of extraction ontologies that best matches the query content. This provides a context for "understanding" the query. HyKSS then distinguishes between semantic text within the query and keyword text, and runs semantic text through the semantic index and keywords through the keyword index. Finally, HyKSS combines the results to produce a hybrid document ranking.[9] To explain and illustrate the process, we use as a running example the query "Hondas in 'excellent condition' in Orem for under 12 grand".

[8] `http://deg.byu.edu` (in the OntologyWorkbench tool)
[9] See the HyKSS demo at `http://deg.byu.edu/demos`

3.1 Context Discovery

HyKSS discovers context by executing OntoES—applying each extraction ontology in its collection to the query text. As a result HyKSS discovers zero or more extraction ontologies to use for generating formal SPARQL queries over the RDF triples in the semantic index. If zero, none match well enough, and HyKSS treats the query as a keyword query.

To discover applicable extraction ontologies, HyKSS computes a score for an ontology's match with a query. The score for an ontology o is a weighted measure of the number of words, phrases, and symbols the data frames of o recognize in the query. In our implementation, HyKSS gives an ontology one point for each **external representation** match and each **context keyword** match, 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 (e.g., *Car* in Figure 1). HyKSS initializes its set of applicable extraction ontologies with the highest-scoring ontology and proceeds from highest to lowest adding any subsequent extraction ontologies that match a part of the query not matched by an ontology already in the set.

For the running example query, "Hondas in 'excellent condition' in Orem for under 12 grand", the data frames in the *Car* ontology in Figure 1 would recognize "Hondas" and "under 12 grand" as constraints on *Make* and *Price* respectively. A second ontology for U.S. cities would recognize "Orem" and would be included in the contextual-ontology set since it augments the coverage of the *Car* ontology with respect to the query. Other ontologies in the collection would likely rank lower or be subsumed by these two ontologies. Suppose, for example, the collection contained another ontology for contractual services that recognizes "under 12 grand" as a constraint. Since the car ontology would score higher and subsume the other ontology, it would be rejected. If the contractual-services ontology also included U.S. cities—being contractual services available in various locations—it could potentially score as high as the car ontology for the query. However, keywords for the nonlexical primary object set *Car* in Figure 1 should include keywords such as the common makes and models of cars that would tip the balance in its favor. Further, whenever there is doubt, HyKSS could return a top-k list of possibilities from which the user could select appropriate contextual ontologies for the query.

We limited the scope of the present study to exclude the evaluation and refinement of the HyKSS context discovery mechanism, leaving this additional investigation for future work. The prototype implementation is based on the heuristics described here, and works well in the relatively narrow scenario we tested for this study, including ontologies for vehicles, mountains, roller coasters, and video games. The HyKSS online demo (`http://deg.byu.edu/demos`) uses 12 ontologies, including date/time, distance, number, price, speed, U.S. location, vehicle type, vehicle, German vehicle, hybrid vehicle, Japanese vehicle, and sports vehicle.

3.2 Semantic Query Processing

The HyKSS semantic query processor takes the output from the context-discovery step and generates a SPARQL query. Figure 4 shows the essence of the result for our running example—the "essence" because HyKSS also generates query components to retrieve information about the extracted text in the web page. This includes (1) information about the web page from which OntoES extracted the text (its URL as well as a reference to a cached copy of the web page), (2) the text strings actually matched (as opposed to their converted internal representations used for query processing), (3) the locations of each text string in the web page (making the semantic index an actual index into its known web pages), and (4) the string's canonical display value (for use in communicating extracted values to a user).

```
PREFIX ann:<http://dithers.cs.byu.edu/owl/ontologies/annotation#>
PREFIX Car:<http://dithers.cs.byu.edu/owl/ontologies/Car#>
PREFIX US_Location:<http://dithers.cs.byu.edu/owl/ontologies/US_Location#>

SELECT ?Car ?CarResource ?US_Location ?US_LocationResource
         ?Make ?MakeValue ?Price ?PriceValue ?US_City ?US_CityValue
WHERE
{
  ?Car ann:inResource ?CarResource .
  ?US_Location ann:inResource ?US_LocationResource .
  FILTER (?US_LocationResource = ?CarResource)

  OPTIONAL{?Car Car:Car-Make ?Make .
     ?Make Car:MakeValue ?MakeValue .}
  FILTER (!bound(?MakeValue) || regex(str(?MakeValue), "Honda")) .

  OPTIONAL{?Car Car:Car-Price ?Price .
     ?Price Car:PriceValue ?PriceValue .}
  FILTER (!bound(?PriceValue) || ?PriceValue < 12000) .

  OPTIONAL{
     ?US_Location US_Location:US_Location-US_City ?US_City .
     ?US_City US_Location:US_CityValue ?US_CityValue .}
  FILTER (!bound(?US_CityValue) || regex(str(?US_CityValue), "Orem")) .
}
```

Fig. 4. Generated SPARQL Query (partial).

The context-discovery step provides the ontologies for the generated SPARQL query—*Car* and *US_Location* in the second and third *PREFIX* statements in Figure 4 for the running example. The first *PREFIX* statement references the annotation schema. In part, it allows for checking that the information referenced by the two ontologies comes from the same resource (web page), as guaranteed by the condition *?US_LocationResource = ?CarResource* in the first *FILTER* statement in Figure 4.

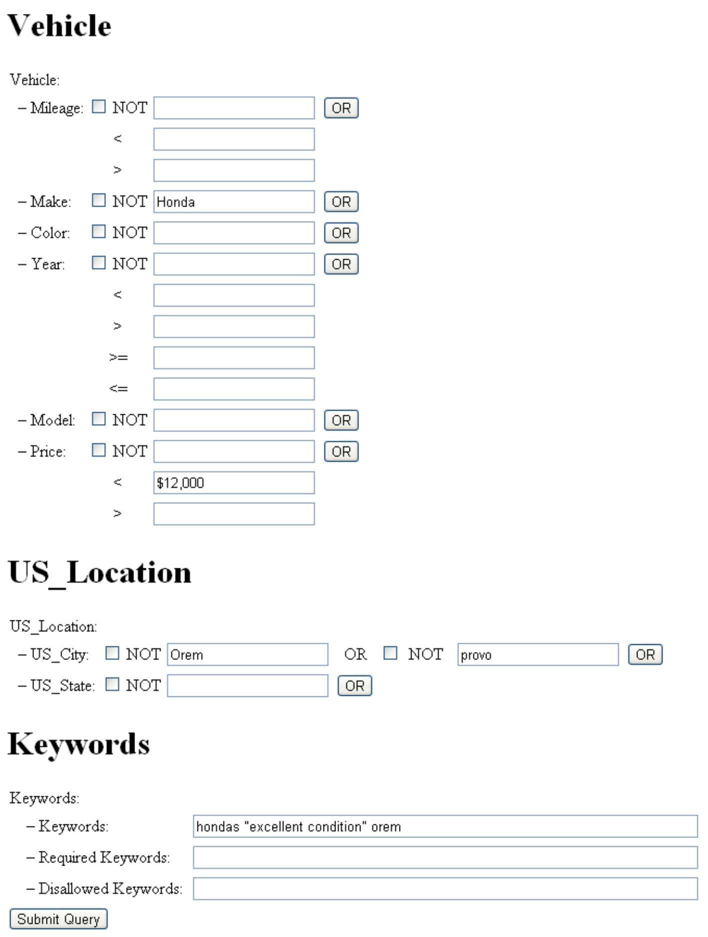
The ontologies from the context-discovery step also provide structure for the query. The nodes of the particular subgraph selected from each ontology for the query depend on which object sets contain data frames whose recognizers match portions of the query text. In our running example, "Hondas" matches with *Make* in the *Car* ontology in Figure 1, "under 12 grand" matches with the *LessThan* operator in *Price*, and "Orem" matches with *US_City* in the *US_Location* ontology. HyKSS also selects as a node the primary object set of each ontology—*Car* in Figure 1 and *US_Location* the US cities and states ontology.

Given the nodes, query generation is straightforward for the running example. As Figure 4 shows, HyKSS generates joins over the edges *Car-Make* and *Car-Price* in the *Car* ontology in Figure 1 and the edge *US_Location-US_City* in the *US_Location* ontology. HyKSS also generates appropriate selection conditions in the *FILTER* statements in Figure 4 so that if the values are bound (i.e., found and extracted from the web page), they satisfy the conditions that the advertised car is a Honda, its price is less than $12,000, and it is in the US city named Orem. The RDF property names (e.g., *Car:MakeValue* and *Car:PriceValue*) in the SPARQL query result from the way we map conceptual-model instances to OWL/RDF. We map each object set to a class C, and for lexical object sets we add a property $CValue$, the concatenation of the object-set name and the literal string "Value".

In general, query generation is straightforward for the types of queries and extraction ontologies we expect to use with HyKSS. Like typical keyword-only queries in which users query for pages that satisfy as many of the keywords as possible, we expect that for hybrid queries users wish to have as many of the semantic constraints satisfied as possible. Thus, we generate only conjunctive queries, and we always allow constraint-satisfaction to be *OPTIONAL* as Figure 4 shows. Further, since we expect most ontologies for HyKSS applications to have a simple structure that is acyclic (or acyclic after excluding prepopulated fixed relationship sets like *Model-Make* in Figure 1), HyKSS can generate queries like it does for the running example in a straightforward way: join across ontologies in the context discovery set by ensuring that the extracted information comes from the same web page, join over edges in the ontologies that connect identified nodes, and filter conjunctively on identified conditions. For cycles that lead to multiple paths between the same two identified nodes, such as if the conceptual-model instance in Figure 1 had a second edge between *Car* and *Feature* denoting features that could be added to a car (as opposed to features or options the car already has), we expect that relationship-set data-frame recognizers would be able to identify clues in queries that would select one path or another, or in the absence of identified clues, interact with the user to disambiguate the query.

Also, like typical keyword-oriented search engines, we provide advanced search capabilities for users who wish to pose queries that involve disjunctions and negations.[10] Thus, a user could pose the query "Find me the Hondas in 'excellent condition' in Orem and Provo for under 12 grand". Then, after selecting *Advanced Search*, a partially filled in form would appear, including the *Make* "Honda"

[10] To see a sample of a generated form, enter some query like the running example query to provide context and then click on *Advanced Search* in the HyKSS demo (`http://deg.byu.edu/demos`).



Vehicle

Vehicle:
- Mileage: ☐ NOT [ ] OR
  - < [ ]
  - > [ ]
- Make: ☐ NOT [Honda] OR
- Color: ☐ NOT [ ] OR
- Year: ☐ NOT [ ] OR
  - < [ ]
  - > [ ]
  - >= [ ]
  - <= [ ]
- Model: ☐ NOT [ ] OR
- Price: ☐ NOT [ ] OR
  - < [$12,000]
  - > [ ]

US_Location

US_Location:
- US_City: ☐ NOT [Orem] OR ☐ NOT [provo] OR
- US_State: ☐ NOT [ ] OR

Keywords

Keywords:
- Keywords: [hondas "excellent condition" orem]
- Required Keywords: [ ]
- Disallowed Keywords: [ ]

Submit Query

Fig. 5. Disjunctively Expanded Advanced Search Form

(having been canonicalized from "Hondas"), the *Price* "$12,000" (written in its output form and as a parameter for the less-than operator declared in its data frame), and the first *US_City* "Orem" but not the second city. To add the second city, a user would click on $\boxed{\text{OR}}$ and enter "Provo". Figure 5 shows the result. Given the filled-in form in Figure 5, generation of the Boolean condition for the query is straightforward.[11]

HyKSS creates the form itself by traversing the conceptual-model graphs of the selected ontologies in the context discovery set. Available conditional oper-

[11] In general, logic form identification in natural language is difficult (e.g., see [8]). In our query here, for example, note that "and" denotes disjunction rather than its more common conjunctive denotation.

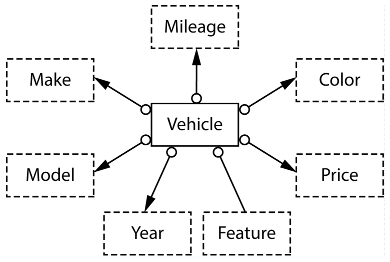


Fig. 6. Vehicle Ontology

ators come from those declared in an object set's data frame. HyKSS also adds typical advanced keyword capabilities as Figure 5 shows. Note that in Figure 5 the *Vehicle* ontology is similar to, but does not correspond to, the ontology in Figure 1; instead, as Figure 6 shows, it corresponds to one of the extraction ontologies we use in the experimental section below. In general, there exists a one-to-one mapping between forms and the conceptual-model hypergraphs we use for extraction ontologies [9]. Thus, HyKSS can both generate query forms as it does for Figure 5 and match filled-in query forms to conceptual structures as it does to generate SPARQL queries. Although not yet integrated into the HyKSS implementation, forms for conceptual-model hypergraphs also allow for nesting [9]—nestings that a user can unfold to allow for bounded queries over recursive cycles in conceptual-model hypergraphs.

3.3 Keyword Query Processing

For keyword queries to work well, it is necessary to remove extraneous words and symbols found in a query but not intended for matching. For hybrid semantic and keyword matching, words and phrases intended to convey non-equality comparison matching should also be removed. This is because such words are intended as endpoints for search constraints rather than artifacts to be matched directly, as we now illustrate.

HyKSS begins its preprocessing of hybrid queries with the removal of comparison constraints. A comparison constraint is a phrase such as "under 12 grand" that a data frame in an extraction ontology recognizes as matching one of its **comparison methods**. This removal prevents the terms "under", "12", and "grand", which likely constitute noise, from matching irrelevant tokens in documents. As an exception, however, the keyword query processor leaves recognized equality-comparison parameters as keywords, so that in database-like query phrases such as "City = Orem" or "Make is equal to Honda", HyKSS only removes the equality symbol and "is equal to" respectively. The motivation here is that words such as "Hondas" and "Orem" in the running example query are useful keywords even when they are part of a phrase recognized as declaring the use of a comparison method. Removal of comparison constraints reduces the running example query to “Hondas in ‘excellent condition’ in Orem”. Note, by the way, that it is the plural “Hondas” left as a keyword, even though the *Make* data-frame recognizer converts the plural to a singular for its use in semantic equality comparisons.

The remaining types of words and symbols to be removed include common punctuation characters, Lucene special characters such as wildcard characters that might unknowingly trigger unexpected functionality, and stopwords from non-quoted phrases. HyKSS passes quoted phrases, such as ‘excellent condition’, for Lucene to process as single-phrase keywords. As is typical in keyword searches, HyKSS retains stopwords within quoted strings so that phrases such as “no dings” remain intact. Since “in” is a stopword and not within a quoted string, the final keyword query for the running example is “Hondas ‘excellent condition’ Orem”.

3.4 Hybrid Query Processing

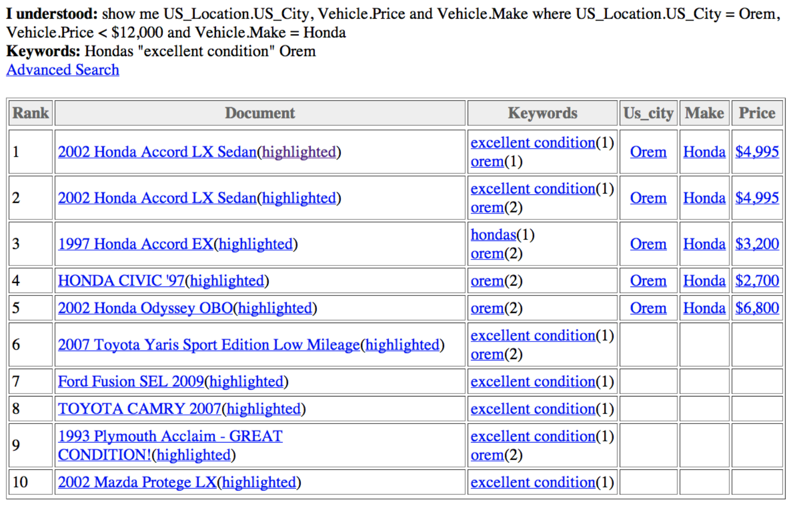
In our implementation, Lucene does the keyword processing, Sesame[12] executes the generated SPARQL query over the semantic store HyKSS previously created for the target documents, and HyKSS provides the ranking algorithm and results display (see Figure 3). Figure 7 shows the ranked results for our running example query, and Figure 8 shows the results of clicking on “highlighted” for the highest ranked document. Upon the click, HyKSS retrieves the cached page from which it extracted information, highlights the information extracted that matches with the user query, and displays the page.

We now give the details of the HyKSS merge and rank process. HyKSS ranks documents by the linear interpolation formula: $keyword\_score \times keyword\_weight + semantic\_score \times semantic\_weight$. For *keyword_score*, HyKSS uses the ranking score returned by Lucene—a number that measures query-document similarity with a combination of the vector space and Boolean models.[13] The ranking of semantic results is still an open area of research (see [10] for a good discussion of the issues). Since there is not a standard semantic ranking algorithm we can “pull off the shelf” as with Lucene for keyword search, HyKSS uses a ranking process based on the amount of requested information that a resulting document includes. After semantic query execution filters out documents that violate a condition specified in the query, the semantic post-query-processor in HyKSS counts for each document the number of lexical object sets matched by the query for which the document returns at least one value. HyKSS then normalizes the resulting counts by dividing by the highest count for the document set, yielding a *semantic_score* for each document. Finally, HyKSS weights keyword and semantic scores according to the contribution of each with respect to the query: $k/(k + s)$ for *keyword_weight* and $s/(k + s)$ for *semantic_weight*, where k is the number of words that remain in the query after removing (1) stopwords (except those in quoted phrases) and

[12] `http://www.openrdf.org`

[13] See `http://lucene.apache.org/core/3_6_1/scoring.html` for details.

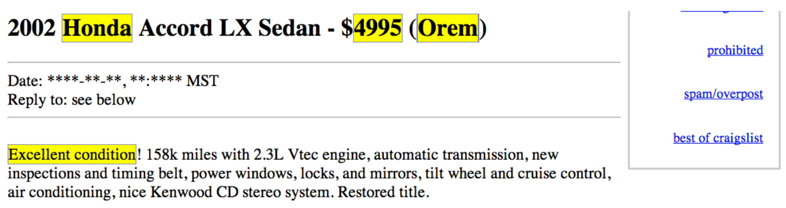
(2) words recognized as denoting semantic inequality comparisons, and s is one-half of the sum of (1) the number of lexical object sets in the ontologies in the context-discovery set matched by the user query and (2) the number of equality and inequality conditions generated for the user query. For our sample query, $k = 4$ ("Hondas", "excellent", "condition", "Orem") and $s = 3$ (one half of three plus three—the three matched lexical object sets in the two ontologies in context-discovery set: *US_Location:US_City*, *Vehicle:Make*, and *Vehicle:Price*, plus the three conditions: *Vehicle:Price* $<$ *\$12,000*, *Vehicle:Make* $=$ *Honda*, and *US_Location:US_City* $=$ *Orem*). To obtain the final ranking, HyKSS sorts the documents by rank score.



I understood: show me US_Location.US_City, Vehicle.Price and Vehicle.Make where US_Location.US_City = Orem, Vehicle.Price < $12,000 and Vehicle.Make = Honda
Keywords: Hondas "excellent condition" Orem
Advanced Search

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	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)			

Fig. 7. Ranked Results for "Hondas in 'excellent condition' in Orem for under 12 grand".



2002 Honda Accord LX Sedan - $4995 (Orem)

Date: ****-**-**, **:**** MST
Reply to: see below

Excellent condition! 158k miles with 2.3L Vtec engine, automatic transmission, new inspections and timing belt, power windows, locks, and mirrors, tilt wheel and cruise control, air conditioning, nice Kenwood CD stereo system. Restored title.

Fig. 8. Highlighted Display of Highest Ranked Document for the Running Example Query.

Observe that the weights in the HyKSS dynamic ranking scheme depend on the query being asked, whereas the keyword and semantic scores depend on the keyword and semantic matches in a document. If HyKSS fails to recognize any semantic information, it uses only keyword ranking. Similarly, HyKSS uses only semantic ranking if no keywords remain after preprocessing the query. The weights adjust on a spectrum 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.

The top five results in Figure 7 have values in the semantic columns (*Us_city*, *Make* and *Price*) indicating that these documents do not violate any semantic constraints. Further, each of the five results has a value for each column, indicating that the semantic ranking algorithm would assign them all the same score. The bottom five results all violate at least one semantic constraint, and thus they receive a semantic score of zero. If HyKSS were limited to only using its semantic ranking algorithm, it would return the top five documents in an arbitrary order. Using the keyword score in combination with semantic score, however, enables HyKSS to move the more query-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 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.

4 Experimental Results

We wish to determine how well HyKSS performs comparatively against keywords alone, semantics alone, and against other approaches to hybrid search. Because the performance of HyKSS depends on the specific ontologies in its collection and their quality together with the document and query sets, the space of comparison possibilities is huge. We therefore sample only a small subset of the possibilities, but we explore thoughtfully so as to learn as much as we can about the space in general. With a pay-as-you-go notion in mind, we also explore the space with respect to the costs and benefits of making extraction ontologies highly domain dependent and tuning them to recognize semantic facts in a domain as accurately as possible.

In subsequent sections, we describe the comparison space we explored. In Section 4.1 we describe the sliding scale of ontological conceptual richness we considered—from a simple extraction ontology that is only able to recognize numbers to a rich extraction ontology with instance recognizers for a full array of domain-dependent concepts. In Section 4.2 we describe the document set we considered, which consists either of documents all in a single application domain or documents in multiple application domains, providing noise for each other's queries. We also describe in Section 4.2 the query sets and how we obtained them, and we explain how we divided them into queries whose semantics we would be sure to recognize (training set) and queries whose semantics we may or may not recognize (validation and test sets). In Section 4.3 we describe the problems we faced in creating a gold standard for evaluation and how we resolved them. In Section 4.4, we describe the query processing and ranking techniques we compared against HyKSS. Finally, in Section 4.5 we present the results of our experiments and discuss their implications.

4.1 Ontology Collections

As a domain of choice for our experimentation, we consider vehicle advertisements. Within the domain, we experiment with five levels of ontological richness: (1) numbers, (2) generic units, (3) vehicle units, (4) vehicle, and (5) vehicle plus special categories of vehicles. The first two are general and apply to many domains (including vehicles), but they are devoid of specific vehicle domain concepts. The last three specifically target the vehicle domain, but at various levels of sophistication.

Level 1: Numbers. The lowest level of semantic modeling consists of a single extraction ontology with a single lexical object set that recognizes only numbers. It recognizes cardinal numbers such as 37, 2,000, and 10.5 but not ordinal or nominal numbers. The *Number* ontology is intentionally designed to be extremely limited. It includes no unit designators, not even *K* or *M* to denote thousands or millions. It does, however, know a few keywords that often refer to numbers such as "price", "mileage", and "year", but does not use keywords to differentiate among different kinds of numbers since there is only one object set for the instances—the numbers themselves. Its only operators are the comparison operators, which also include keywords such as "less than" to provide textual identifiers for operator applicability.

Level 2: Generic Units. This next level of semantic modeling uses three simple ontologies: *DateTime*, *Distance*, and *Price*. The *DateTime* ontology recognizes common date-time formats and stores values in seconds since 1970 for comparison. It 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. We store recognized distances canonically as measurements in meters. The *Price* ontology only recognizes US dollar amounts. It 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*, "longer" for *Distance*, and "under" for *Price*.

Level 3: Vehicle Units. Moving closer to the vehicle domain, the next level of semantic modeling includes three simple ontologies: *VehiclePrice*, *VehicleMileage* and *VehicleYear*. Having knowledge of the domain lets the ontologies recognize values that are applicable to vehicles. For example, the *DateTime* ontology recognizes the string "1836" but the *VehicleYear* ontology does not, since "1836" cannot be the year for an automobile make and model.

Level 4: Vehicle. The next level of semantic modeling is domain specific. The ontology collection supporting vehicles consists of a single ontology with common concepts of interest, including *Color*, *Make*, *Mileage*, *Model*, *Price*, and *Year*, as Figure 6 shows. It does not attempt to model every possible concept in the entire vehicle domain, or even all the concepts in Figure 1. With a higher level of semantic modeling, the *Vehicle* ontology collection is able to consider more context during extraction. For example, the *Year* object set in the *Vehicle* ontology can take advantage of the fact that a *Year* value often precedes a *Make* or *Model* value, whereas *VehicleYear* cannot because it has no concept of *Make*. Also, whereas lexical object sets in less semantically rich collections have no constraint on the number of occurrences they can extract from a document, the functionally dependent lexical object set in the *Vehicle* ontology has a limit of one extracted instance, meaning that HyKSS extracts only one *Color*, *Make*, *Mileage*, *Model*, *Price*, and *Year* from a document.

Level 5: Vehicle+. The final level of semantic modeling consists of the *Vehicle* ontology, together with five new ontologies: *GermanVehicle*, *HybridVehicle*, *JapaneseVehicle*, *SportsVehicle*, and *VehicleType*. These additional ontologies allow for more fine tuned extraction and querying in the vehicle domain. The *GermanVehicle* and *JapaneseVehicle* ontologies determine whether vehicle makes are German or Japanese. They also recognize terms such as "German car" and "Japanese car". The *HybridVehicle* and *SportsVehicle* ontologies function similarly but make use of vehicle models rather than makes. The *VehicleType* ontology also relies on vehicle models and categorizes vehicles as "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".

4.2 Document and Query Sets

As Fernandez, et al. point out, the semantic-search community has no benchmarks to judge the quality of semantic-search methods [11]. One of the difficulties in collecting queries is to explain to subjects what semantic-search queries are. Fortunately, we had built a rudimentary semantic-search engine for the vehicle domain and had constructed an online demo.[14] We asked students in two senior-level database classes 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 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. We removed a total of

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

24 queries, leaving 113 queries in the final set. These 113 queries constitute our training query set.

We posed a similar task to a class of students in computational linguistics. We did not present the online demo to these students, but rather showed them a few example queries and asked them to submit hand-written queries they felt a semantic system like HyKSS could handle. This resulted in 71 unique queries from which we removed 11 using the same previously mentioned criteria. The remaining 60 queries constitute our blind query set.

For document sets we chose to use vehicle advertisements posted on local craigslist sites.[15,16] Craigslist lets users 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 250 vehicle advertisements from the "for sale by owner" section 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 validation and test sets are from the Salt Lake City craigslist site.[17]

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[18], 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.[19] In creating the subset we attempted to avoid pages that referred to a mountain range and chose a selection of pages about single mountains. We downloaded the roller coaster pages using the links from the list-of-roller-coaster-rankings page.[20] 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. We included pages with the same URL only once.

As it turned out, only 76 of the 113 queries in the training set corresponded to valid results in the document sets. Rather than report vacuous results, we excluded the non-applicable queries from the training set. Likewise, only 15 of the 60 queries in the blind set were applicable. This left us with a total of 91 useful queries between the training and blind query sets that were applicable to our chosen document sets.

[15] `http://provo.craigslist.org`
[16] `http://saltlakecity.craigslist.org`
[17] The document sets may have contained duplicate or similar advertisements due to cross posting and re-posting tendencies of craigslist users. We did not check for this, but rather just took the ads as posted.
[18] `http://www.wikipedia.org`
[19] `http://en.wikipedia.org/wiki/List_of_mountains`
[20] `http://en.wikipedia.org/wiki/List_of_roller_coaster_rankings`

4.3 Annotation and Tuning

The decision to create our own document and query sets also required that we create gold standard annotations for the data. To avoid being influenced by the blind sets of documents and queries, we first tuned our extraction-ontologies on the training sets of documents and queries. By "tuning" we mean revising the ontologies to improve their extraction performance, for example, by refining existing extraction regular expressions, adding entries to lexicons, or otherwise editing associated data frames. We then locked them from further tuning and proceeded with the creation of our annotations for the gold standard.

We began our tuning by ensuring that each ontology collection extracted properly from and generated proper interpretations for the queries in the query training set. For these queries, we were able to tune our *Numbers*, *Vehicle*, and *Vehicle+* ontology collections up to 100% and our *Generic Units* and *Vehicle Units* ontology collections up to 98% accuracy. We continued the tuning process by ensuring that the ontology collections extracted properly from documents in the training document set. We tuned only for text; an image, for example, may show the color of a vehicle, but OntoES does not extract this information without textual cues. The tuned precision of the ontology collections over the training documents ranged from 97% to 100%, and recall ranged from 94% to 100%. This is typical of the type of performance we expect from OntoES in this type of domain [5]. The only ontology collection to achieve 100% in either metric was the *Numbers* collection.

We next annotated the validation document set. To see what drop-off we should expect when we turned to the blind document set, we measured precision and recall for the ontology collections. For the *Vehicle+* ontology collection, for example, we found that recall fell from 99% for the training document set to 94% for the validation document set and that precision fell from 98% to 90%. We therefore anticipated a similar decline in extraction quality for the blind document set and had some idea of what to expect for the blind query set as well.

With ontology tuning complete, we next constructed query-document relevance annotations for the training query set relative to the blind document set. Assigning query-document relevance is inherently subjective. To remain as objective as possible, we decided to use a closed-world assumption in assigning relevance: a document is relevant to a query if it explicitly satisfies all constraints in the query and has some response for each concept (e.g., *Price*, *Make*, *US_City*) mentioned in the query. When determining relevance, we used all information available in the advertisements, including 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. We used all of this information in determining query-document relevance. We did not, however, attempt to infer information not available in the advertisement from information present. For example, if an 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* (as per the fixed *Make-Model* relationship set in Figure 1) 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.

4.4 Query Processing and Ranking Techniques

We evaluated the retrieval quality of HyKSS by comparing it and two of its variations with keyword-only techniques, semantics-only techniques, and generic hybrid techniques, which we describe and label as follows.

– **Keyword** (K): Lucene ranks results of processing queries after removing non-phrase stopwords.
– **Keyword - Pre-processing** (K_p): Lucene ranks results of processing queries after removing comparison-constraint words and non-phrase stopwords.
– **HyKSS - Set Weights** (H_s): HyKSS processes queries and ranks results using set weights for interpolating keyword and semantic scores.[21]
– **HyKSS - Dynamic Weights** (H_d): HyKSS processes queries and ranks results using its dynamic ranking scheme described in Section 3.4.
– **HyKSS - Single Ontology** (H_o): HyKSS processes queries and ranks results using its dynamic ranking scheme, but instead of considering ontology sets, HyKSS uses only the single best matching ontology.
– **Keyword - Soft Semantics** (G_s): Lucene ranks results as for K_p, but only for results that satisfy a soft semantic filter that eliminates documents that explicitly violate a semantic constraint (open world assumption).
– **Keyword - Hard Semantics** (G_h): Lucene ranks results as for K_p, but only for results that satisfy a hard semantic filter that eliminates documents that do not satisfy all semantic constraints (closed world assumption).
– **Soft Semantic Ranking** (S_r): uses a HyKSS-style dynamic ranking scheme, but only for documents that pass the soft semantic filter.
– **Soft Semantics** (S_s): This process retrieves all results that satisfy the soft semantic filter, but does no ranking, leaving results in an arbitrary order.
– **Hard Semantics** (S_h): This process retrieves all results that satisfy the hard semantic filter, but does no ranking, leaving results in an arbitrary order.

4.5 Experiments and Results

Since the primary focus of HyKSS is to retrieve relevant documents, we use Mean Average Precision (MAP) [12] to evaluate its quality.[22] In essence, MAP

[21] We established the keyword and semantic weights as those that maximized mean average precision over the 50 validation documents. To find the keyword weight k, we executed the 100 queries in the training query set using 101 weight combinations (0.0–1.0) and chose the best. The semantic weight s became $1 - k$.

[22] See [13] for a full discussion of the principles of ranking in information retrieval.

declares a retrieval system to be perfect for a query q if it returns all N relevant documents for q in the top N ranking positions. MAP penalizes a retrieval system if it ranks irrelevant documents before all relevant documents. Precision for the kth relevant document in the ranking for query q, $Pr(D_{qk})$, is computed as $k/(n+1)$ where n is the number of documents that precede the kth relevant document in the ranking. Thus, the average precision for a single query q is $\frac{1}{n_q}\sum_{k=1}^{n_q} Pr(D_{qk})$, where n_q is the number of relevant documents for query q. Mean average precision for a set Q of queries is the mean of each average precision across all queries:

$$MAP(Q) = \frac{1}{|Q|}\sum_{q=1}^{|Q|}\frac{1}{n_q}\sum_{k=1}^{n_q} Pr(D_{qk}).$$

We computed MAP scores for our training and blind query sets over documents in our clean and noisy document sets across all ontology collections (*Numbers*, *Generic Units*, *Vehicle Units*, *Vehicle*, and *Vehicle+*) and for each of our experimental techniques identified in Section 4.4 (K, K_p, H_s, H_d, H_o, G_s, G_h, S_r, S_s, S_h). Figure 9 illustrates the type of results we measured; it shows the average MAP scores that result from applying all ten techniques to the training query set over the clean document set. As should be expected, we found that MAP scores for the training query set were higher than corresponding MAP scores for the blind query set, and MAP scores for the clean document set were higher than corresponding scores for the noisy document set.

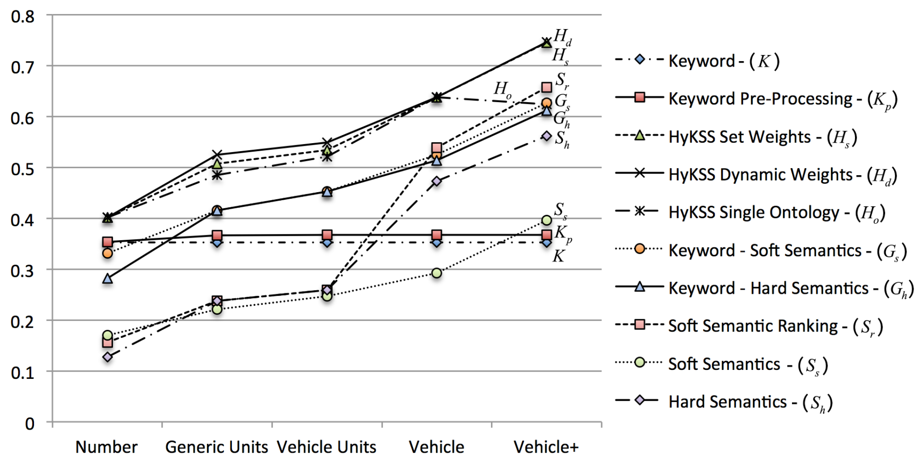


Fig. 9. MAP Scores for Training Query Set and Clean Document Set

Because we applied ten different techniques to the same queries and documents using a variety of ontology collections, our MAP-score responses are correlated. Thus, the most suitable methodology for interpreting the data is multivariate analysis of variance (see [14], especially Chapters 6 and 8). The need to account for the inherent differences between the 91 applicable queries in our training and blind query sets led us to use a block statistical model. The factors in our model are document set (clean and noisy) and ontology collection (*Numbers*, *Generic Units*, *Vehicle Units*, *Vehicle*, and *Vehicle+*). We studied the main effects of document set and ontology collection, and possible interactions between those factors to determine differences.[23]

After examining the multivariate analysis of variance for our data, we discovered that the ten techniques behave similarly within each family. That is, the HyKSS (H_s, H_d, and H_o), generic-hybrid (G_s, G_h), keyword-only (K and K_p), and semantics-only (S_r, S_s, S_h) technique families all behaved in a statistically similar manner within each family type. Thus, to simplify the presentation, we have chosen to report the results of only the four techniques that best represent each family. We chose H_d, G_s, K, and S_r as the representative techniques for the four families.

Table 1 is the multivariate analysis of variance for our block statistical model over the four representative techniques. We have allowed all treatments to have their own sources of variability. *Error1* is the interaction between query and document set, and *Error2* is the interaction between query and ontology collection. The test statistic we used is Wilks' Λ. We found that document set, collection, and the interaction between document set and collection are all statistically significant, with p-values less than .0078, .0001, and .0001, respectively. Because the interaction of document set and ontology collection turned out to be statistically significant, we then looked into decomposing the differences among the various techniques. We are interested in the relative performance of HyKSS, so we examined the differences between H_d and the other three representative techniques. We found that H_d and K are distinct ($p < .0018$), as are H_d and S_r ($p < .0001$). However, H_d and G_s are statistically indistinguishable at the level of the document-set by ontology-collection interaction. Therefore we examined the main effect, ontology collection, and found that at the level of ontology collection, H_d and G_s are indeed different ($p < .0001$).

Table 2 gives average MAP scores for the four representative techniques, grouped by ontology collection and document set, across the combined population of training and blind queries. Figure 10 shows graphs of the MAP scores from Table 2. The differences in performance between the various techniques are significant within the $p < .05$ threshold that we had established at the outset of our experimentation, meaning that the differences are not merely random.

Figure 11 shows average MAP scores across the combination of clean and noisy document sets—the resulting data if we conclude that the interaction between document set and ontology collection is unimportant. Because we found the magnitude of the interaction to be small, and upon further examination discovered that the graphs of the various sub-cases of factor combinations always

[23] We began our analysis using a split plot design, but when we examined the interactions among our factors and found that the interactions were not major sources of response variability, we simplified to a block statistical model.

Source	DF	Wilks' Λ	F Value	Pr > λ	Pr > F
Query	90				
Document Set	1	0.8745		<.0078	
Error1	90				
Ontology Collection	4	0.4279		<.0001	
H_d vs. K			48.79		.0001
H_d vs. S_r			4.19		.0025
H_d vs. G_s			21.87		.0001
Error2	360				
Doc. Set*Collection	4	0.7569		<.0001	
H_d vs. K			4.37		.0018
H_d vs. S_r			18.50		.0001
H_d vs. G_s			0.31		.8696
Error	360				

Error1 is for Query*Document Set interaction

Error2 is for Query*Ontology Collection interaction

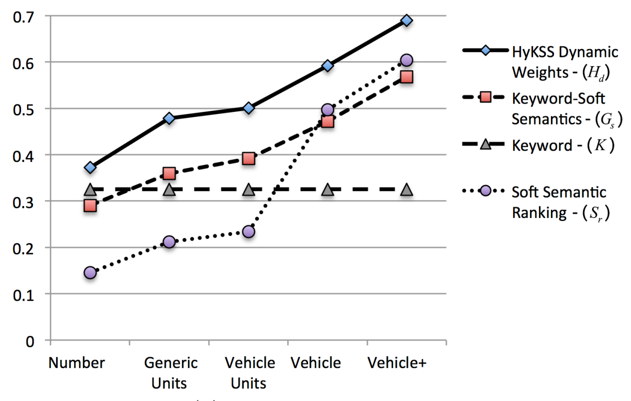
Table 1. Multivariate Analysis of Variance

Doc. Set	Collection	H_d	S_s	K	G_r
Clean	Number	0.3720	0.2906	0.3255	0.1450
	GenericUnits	0.4783	0.3591	0.3255	0.2116
	VehicleUnits	0.5009	0.3917	0.3255	0.2341
	Vehicle	0.5925	0.4725	0.3255	0.4975
	Vehicle+	0.6896	0.5677	0.3255	0.6043
Noisy	Number	0.2682	0.2157	0.2762	0.0332
	GenericUnits	0.4060	0.3000	0.2762	0.1207
	VehicleUnits	0.4603	0.3594	0.2762	0.1672
	Vehicle	0.5218	0.3342	0.2762	0.4187
	Vehicle+	0.6079	0.4157	0.2762	0.5101

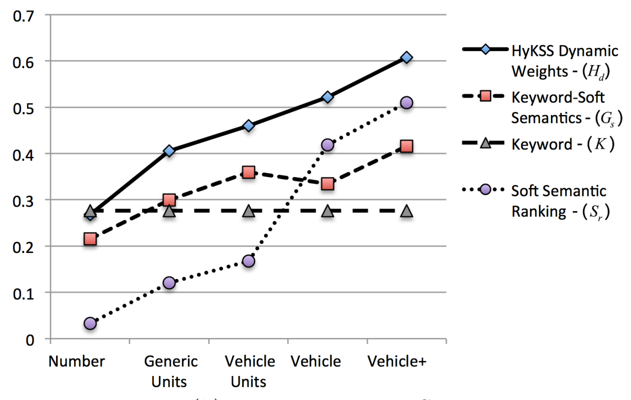
Table 2. MAP Scores by Representative Technique for Ontology Collections Grouped by Document Set

yielded a pattern similar to Figure 11, we concluded that the interaction between document set and ontology collection is not practically important even though it is statistically significant.

Next we performed several post-hoc tests to determine whether the differences in MAP scores that we see in Figure 11 are significant. We narrowed our multivariate analysis of variance to just the *Vehicle* and *Vehicle+* cases and found that the three techniques that use semantics—H_d, G_s, and S_r— behave in a statistically indistinguishable pattern. To visualize why this is so, notice how the slope of the lines for these techniques in Figure 11 for the *Vehicle* and *Vehicle+* cases is similar. However, we found that the difference between these three and the keyword technique, K, is indeed significant as we should expect. We further drilled down into the data and compared the average MAP scores for H_d across *Vehicle* and *Vehicle+* and found the difference to be statistically



(a) Clean Document Set



(b) Noisy Document Set

Fig. 10. Representative MAP Scores

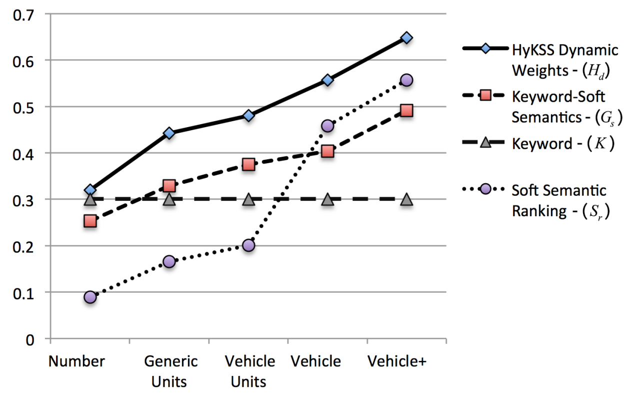


Fig. 11. Representative MAP Scores Averaged over Document Set

significant (F value 24.17, $p < .0001$). We also found that the difference between average MAP scores for H_d and S_r on *Vehicle+* is statistically significant (F value 28.24, $p < .0001$). In other words, we are satisfied that Figure 11 summarizes the data in a statistically reliable manner.

The graph in Figure 11 suggests at least five conclusions. (1) Keyword-only techniques are relatively constant in their performance[24] regardless of the sophistication of the available semantics. This is what we should expect, because these techniques do not use any semantics in calculating query results. (2) Conversely, all of the techniques that take advantage of semantics (HyKSS, semantics-only, and generic-hybrid techniques) yield improved results when the available semantics become more sophisticated. Again, this is intuitively what we should expect from the nature of the task. (3) The semantics-only approach is inferior to hybrid and keyword techniques for simple ontology collections where there is not much semantic support. However, semantics-only can be quite good when the ontology collection is rich. (4) The generic-hybrid approach trends in the same direction as HyKSS, but at a lower level of overall performance. (5) HyKSS performs better on average than the other techniques (significant at the level $p < .05$) across a range of conditions (query set, document set, and ontology collection). As we would expect, the difference between HyKSS and keyword-only performance is minimal when the available semantics are simple, but becomes much more pronounced as the available semantics increase. For simplicity we have only presented the analysis for the representative techniques, but we performed the full analysis for all ten techniques and found similar results, so these conclusions hold for all the techniques we tested.

We have also tried HyKSS within the domains of mountains, roller coasters, and video games. Anecdotally, the performance seemed to be similar to the car ads domain, but we did not perform extensive experiments with a careful statistical analysis of those results and so we cannot make any formal claims with respect to other domains. But informally, at least, we saw encouraging results and believe the approach will generalize to other domains of this type.

5 Related Work

HyKSS processes free-form queries in a hybrid fashion, recognizing both keywords and semantic constraints. Looking for similar systems, we found little work in efforts to handle comparison constraints in free-form queries, and none in the hybrid search arena. We did see, however, that Microsoft announced that its search engine Bing[25] has natural language capabilities that make use of comparison constraints on price in the search process. Being proprietary, we are unable to make direct comparisons with HyKSS, but we did enter our running

[24] In our experiment, the keyword results in the representative case turned out to be literally constant, but in general they need not be, and indeed the Keyword - Pre-processing approach (K_p) yielded results that were not quite constant.

[25] `http://www.bing.com`

query on the Bing home page.[26] Bing failed to semantically process the comparison constraint "under 12 grand" but, at least, returned results in the right domain, whereas another widely used search engine not only failed to recognize the constraint, but also returned in its top results an ad for a grand piano in Orem in "excellent condition".

The hybrid search systems of which we are aware either all require structured queries rather than free-form queries [1, 15–17], require a "keyword-based structured query language" [18], or require queries that are an extension of formal conjunctive queries [19, 20]. The structured query hybrid systems do, however, have retrieval and ranking mechanisms similar to those in HyKSS. The authors of [15] present an adaption of the vector space model for hybrid search that weights annotated concepts in documents according to a TF/IDF scheme. The form-based query system of K-Search [1] is similar to the advanced form interface of HyKSS, but does not include negations. By default, results must satisfy a hard semantic filter and are ranked according to keyword score, which is comparable in concept to the *Keyword - Hard Semantics* technique in our experiments. GoNTogle [1] allows users to return the intersection or union of keyword and semantic search results. For semantic ranking, GoNTogle 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 involved in query execution. Like HyKSS, GoNTogle combines the semantic score with the keyword with linear interpolation, but with pre-set weights.

The authors of [16] leverage semantic annotations in the context of existing search engines. They use a generalization of the PageRank algorithm, *Object-Rank*, 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 then transforms formal structured queries into a sequence of standard web search queries and combines results. Experiments show that the approach performs well in terms of precision and recall.

In addition to hybrid search, CE^2 [19] and Semplore [20] show viability for large-scale applications. CE has a unified framework that includes both RDF data and documents. Preliminary results show increased precision over keyword and semantic search while maintaining acceptable response times with millions of triples.

Although not a hybrid system, PowerAqua [21, 22] has a number of similarities with HyKSS. It accepts a natural language query and returns annotations from various public structured knowledge repositories as answers. In its search for relevant annotations, PowerAqua considers multiple ontologies and maps ontologies together, relying on syntactic label similarities, semantic similarities in ontology hierarchies and WordNet, and some heuristics. PowerAqua, by itself, is a semantic search system, although Fernandez, et al. [11] discuss combining PowerAqua with the work of Castells, et al. [15] 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

[26] January, 2012

combines ontologies dynamically at runtime 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 ontological context. PowerAqua relies instead on labels and the structure of ontologies without data frames to discover similarities. However, a significant strength of the PowerAqua approach is its open domain strategy. Whereas HyKSS relies on ontologies and data frames that have been crafted for specific search domains that extraction-ontology developers deem to be of interest, PowerAqua generally is able to build its cross-ontology queries from a much larger pool of ontologies. HyKSS would certainly be a more viable solution if it likewise could draw from a larger set of underlying extraction ontologies.

Based on our work we believe that it is fruitful to explore blending the best of both the semantic and keyword worlds in a hybrid approach like HyKSS. Since there is increasing interest in semantic search techniques, we expect this to be the topic for continued future study as researchers continue to improve the available non-hybrid approaches. For example, two non-hybrid semantic search systems of note are FREyA [23] and MORAG [24]. FREyA transforms natural-language questions into SPARQL queries by mapping words from the natural-language question to concepts within an underlying ontology. FREyA uses a combination of syntactic parsing, ontology-based reasoning, and user interaction to establish and refine its interpretation of natural-language queries. MORAG is a framework that uses explicit semantic analysis (ESA) to map (and match) query concepts to document concepts through the lens of some semantic concept space derived from such knowledge sources as Wikipedia. There are numerous other approaches, and Lopez et al. [25] provide a nice survey of question-answering systems designed for the semantic web.

There is also a considerable amount of work in keyword-oriented search over relational databases or other structured and semi-structured stores such as RDF graphs. Blinks [26] and SearchWebDB [27] are exemplars of such systems. A more recent example that provides improved results is Yaanii [28], a system that supports keyword search over RDF data sets using linear, monotonic, and combined linear/monotonic strategies. Because there is a massive and growing amount of data available on the semantic web, these types of approaches will continue to be important areas of ongoing research.

6 Conclusion

HyKSS processes hybrid keyword and semantic search queries in accord with five principles: (1) semantic indexing, along with standard keyword indexing, (2) semantic analysis as well as keyword analysis of user queries, (3) result ranking based on both semantic and keyword match, (4) identification and use of overlapping ontological contexts, and (5) advanced form search when conjunctive free-form queries are insufficient. Semantic indexing for HyKSS is similar to standard keyword indexing in the sense that HyKSS precrawls web sites (e.g., craigslist.org), indexes the semantics with extraction ontologies, and leaves a pointer to the original text semantically identified by OntoES. Semantic query analysis matches OntoES-identified semantics in the query with known ontologies and removes semantic-only query words, as well as stop words, before processing keywords. The HyKSS result ranking algorithm dynamically apportions weights to keywords and semantics depending on the proportion of keywords and semantics in the query. HyKSS can process queries that span ontologies, both for conjunctive free-form queries and advanced form queries (e.g., cars and US locations in our running example query and in our example of a form query in Figure 5).

HyKSS performs hybrid keyword and semantic free-form queries well. With a chosen alpha level of $p < .05$ we found for our experiments with car ads from craigslist.org that HyKSS outperforms keyword search in isolation, semantic search in isolation, and two generic hybrid search protocols (*Keyword - Soft Semantics* and *Keyword - Hard Semantics*). Our experiments also show that as semantic recognition within a domain increases, ranking results can improve: For the *HyKSS - Dynamic Weights* technique in the car-ads domain, *Vehicle+* performs better than *Vehicle* ($p < .0001$), and more generally as Figure 11 shows, all of the techniques that leverage semantic recognition tend to perform better when more semantic recognition is available. Across the domains we considered, even a small amount of semantics (e.g. *Generic Units*) appears to perform better than no semantics at all as Figure 11 shows. We also observed that too little semantics combined with keyword search performs worse than keyword search in isolation for some techniques (e.g., consider *Numbers* for S_r and G_s in Figure 11). However, as the performance of HyKSS demonstrates, there are significant advantages to hybridizing keyword and semantic search techniques, and the HyKSS approach itself appears promising.

As with its underlying extraction technology, HyKSS has the limitation that it will work best within domains that are relatively narrow in ontological breadth. The extraction ontologies upon which HyKSS relies generally need to be created by specialists who are trained in ontology development. A useful area for future research is to study ways to speed up the ontology development process. We have made some initial studies (e.g. [29]), but much more could be done. As we observed in Section 5, a potentially fruitful way to address this need is also to explore an open domain strategy as others have done, and thus be able to rely upon the work of a much broader community of ontology developers.

Another area needing further investigation is how HyKSS would scale in the presence of a larger foundation of extraction ontologies. In particular, the question of context discovery (see Section 3.1) becomes more challenging in the presence of a larger set of ontologies than the dozen or so used in the current implementation of HyKSS.

References

1. R. Bhagdev, S. Chapman, F. Ciravegna, V. Lanfranchi, and D. Petrelli. Hybrid search: Effectively combining keywords and ontology-based searches. In *Proceed-*

ings of the 5th European Semantic Web Conference (ESWC'08), pages 554–568, Tenerife, Canary Islands, Spain, June 2008.

2. D.W. Embley, S.W. Liddle, D.W. Lonsdale, J.S. Park, B.-J. Shin, and A. Zitzelberger. Cross-language hybrid keyword and semantic search. In *Proceedings of the 31st International Conference on Conceptual Modeling (ER 2012)*, pages 190–203, Florence, Italy, October 2012.
3. D.W. Embley and A. 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, Sophia, Bulgaria, February 2010.
4. H. Stuckenschmidt. Data semantics on the web. *Journal on Data Semantics*, 1(1):1–9, 2012.
5. D.W. Embley, D.M. Campbell, Y.S. Jiang, S.W. Liddle, D.W. Lonsdale, Y.-K. Ng, and R.D. Smith. Conceptual-model-based data extraction from multiple-record web pages. *Data & Knowledge Engineering*, 31(3):227–251, 1999.
6. P. Buitelaar, P. Cimiano, P. Haase, and M. Sintek. Towards linguistically grounded ontologies. In *Proceedings of the 6th European Semantic Web Conference (ESWC'09)*, pages 111–125, Heraklion, Greece, May/June 2009.
7. D.W. Embley. Programming with data frames for everyday data items. In *Proceedings of the 1980 National Computer Conference*, pages 301–305, Anaheim, California, May 1980.
8. V. Rus. A first evaluation of logic form identification systems. In R. Mihalcea and P. Edmonds, editors, *Senseval-3: Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 37–40, Barcelona, Spain, March 2004.
9. C. Tao, D.W. Embley, and S.W. Liddle. FOCIH: Form-based ontology creation and information harvesting. In *Proceedings of the 28th International Conference on Conceptual Modeling (ER2009)*, pages 346–359, Gramado, Brazil, November 2009.
10. V. Lopez, A. Nikolov, M. Fernández, M. Sabou, V.S. Uren, and E. Motta. Merging and ranking answers in the semantic web: The wisdom of crowds. In A. Gómez-Pérez, Y. Yu, and Y. Ding, editors, *ASWC*, volume 5926 of *Lecture Notes in Computer Science*, pages 135–152. Springer, 2009.
11. M. Fernandez, V. Lopez, M. Sabou, V. Uren, D. Vallet, E. Motta, and P. 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.
12. C.D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval.* Cambridge University Press, New York, New York, July 2008.
13. T.-Y. Liu. *Learning to Rank for Information Retrieval.* Springer, Berlin, Germany, 1st edition, 2011.
14. A.C. Rencher and W.F. Christensen. *Methods of Multivariate Analysis.* Wiley, 3rd edition, July, 2012.
15. P. Castells, M. Fernandez, and D. 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.
16. B. Fazzinga, G. Gianforme, G. Gottlob, and T. Lukasiewicz. Semantic web search based on ontological conjunctive queries. In *Proceedings of the Sixth International Symposium on Foundations of Information and Knowledge Systems (FoIKS10)*, pages 153–172, Sophia, Bulgaria, February 2010.

17. G. Giannopoulos, N. Bikakis, T. Dalamagas, and T.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.
18. J. Pound, I.F. Ilyas, and G. Weddell. Expressive and flexible access to web-extracted data: A keyword-based structured query language. In *Proceedings of the 2010 International Conference on Management of Data (SIGMOD'10)*, pages 423–434, Indianapolis, Indiana, June 2010.
19. H. Wang, T. Tran, and C. 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.
20. L. Zhang, Q. Liu, J. Zhang, H. Wang, Y. Pan, and Y. 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/ASWC'07)*, pages 652–665, Busan, Korea, November 2007.
21. V. Lopez, V. Uren, M.R. Sabou, and E. 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.
22. V. Lopez, M. Fernández, E. Motta, and N. Stieler. Poweraqua: Supporting users in querying and exploring the semantic web. *Semantic Web*, 3(3):249–265, 2012.
23. D. Damljanovic, M. Agatonovic, and H. Cunningham. Natural language interfaces to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC10)*, pages 106–120, Heraklion, Greece, May/June 2010.
24. O. Egozi, S. Markovitch, and E. Gabrilovich. Concept-based information retrieval using explicit semantic analysis. *ACM Transactions on Information Systems*, 29(2):1–34, April 2011.
25. V. Lopez, V.S. Uren, M. Sabou, and E. Motta. Is question answering fit for the semantic web?: A survey. *Semantic Web*, 2(2):125–155, 2011.
26. H. He, H. Wang, J. Yang, and P.S. Yu. Blinks: ranked keyword searches on graphs. In C.Y. Chan, B.C. Ooi, and A. Zhou, editors, *SIGMOD Conference*, pages 305–316. ACM, 2007.
27. T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In Y.E. Ioannidis, D.L. Lee, and R.T. Ng, editors, *ICDE*, pages 405–416. IEEE, 2009.
28. R. De Virgilio, A. Maccioni, and P. Cappellari. A linear and monotonic strategy to keyword search over rdf data. In F. Daniel, P. Dolog, and Q. Li, editors, *ICWE*, volume 7977 of *Lecture Notes in Computer Science*, pages 338–353. Springer, 2013.
29. D.W. Lonsdale, D.W. Embley, Y. Ding, L. Xu, and M. Hepp. Reusing ontologies and language components for ontology generation. *Data & Knowledge Engineering*, 2009.