SPARK: Adapting Keyword Query to Semantic Search

Qi Zhou, Chong Wang, Miao Xiong, Haofen Wang, and Yong Yu

Apex Data and Knowledge Management Lab*
Department of Computer Science and Engineering,
Shanghai JiaoTong University, 200240, Shanghai, P.R. China
{jackson,wangchong,xiongmiao,whfcarter,yyu}@apex.sjtu.edu.cn

Abstract. Semantic search promises to provide more accurate result than present-day keyword search. However, progress with semantic search has been delayed due to the complexity of its query languages. In this paper, we explore a novel approach of adapting keywords to querying the semantic web: the approach automatically translates keyword queries into formal logic queries so that end users can use familiar keywords to perform semantic search. A prototype system named 'SPARK' has been implemented in light of this approach. Given a keyword query, SPARK outputs a ranked list of SPARQL queries as the translation result. The translation in SPARK consists of three major steps: term mapping, query graph construction and query ranking. Specifically, a probabilistic query ranking model is proposed to select the most likely SPARQL query. In the experiment, SPARK achieved an encouraging translation result.

1 Introduction

In the next stage of web revolution, termed the semantic web, web resources will be made available with various kinds of metadata described in ontologies[1]. Correspondingly, many semantic query languages (e.g. RQL [1], RDQL [2], SquishQL [2] and SPARQL [3]) have been proposed for querying these ontologies. However, in order to use these semantic query languages, end users have to master complex formal logic representations and be familiar with the underlying ontologies. This has become a critical gap between semantic search and end users [3][4]. Meanwhile, most users have been accustomed to the traditional keyword search for years. Therefore, it is valuable to enable the users to carry out semantic search by inputting keyword query. However, keyword query is very different from semantic search. To adapt keyword query to semantic search, we have to overcome the following obstacles: 1) *Vocabulary Gap*: Casual web users usually

* This work is funded by IBM China Research Lab.

[1] In this paper, ontology refers to a knowledge base (KB) that includes concepts, relations, instances and instance relations that together model a domain.

[2] http://www.w3.org/Submission/RDQL/

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

K. Aberer et al. (Eds.): ISWC/ASWC 2007, LNCS 4825, pp. 694–707, 2007.

have no knowledge of the underlying ontology, so the words in their queries may be quite different from those in the ontology. 2) *Lack of Relation*: Relations between concepts/instances are required to be explicitly stated in formal logic queries, which are often missing in keyword queries [5]. How to automatically discover these missing relations becomes a big problem. 3) *Query Ranking*: Due to the ambiguity of keyword query, there may be multiple formal queries produced from one keyword query. How to rank these queries is a big challenge [6].

Faced with these difficulties, we present our novel approach in SPARK system. SPARK can automatically translate keyword queries into corresponding SPARQL queries under the domain ontology, with the aim of adapting keyword query to semantic search. The main translation steps are: *term mapping*, *query graph construction* and *query ranking*. *Term mapping* maps the terms of a keyword query to the resources of the knowledge base to narrow the vocabulary gap. After that, *query graph construction* links the mapped resources so that the missing relations and concepts can be obtained and a complete query graph can be constructed. Finally, the probabilistic *query ranking* model estimates the most likely SPARQL query among all the candidate queries. In this way, the end users can keep the habit of typing keywords and querying the semantic web data transparently, which increases the social utilization of semantic search. From the evaluation of 750 various keyword queries over three ontologies, SPARK achieved an encouraging MRR [4] score of 0.677.

The rest of this paper is organized as follows: Section 2 defines the problem of formal query construction. Section 3 details the main steps of formal query construction in SPARK. The implementation and experimental result are presented in Section 4. Section 5 outlines the related work. We give the conclusion and future work in the last section.

2 From Keywords to Formal Query

Answering many kinds of semantic query can be formulated as a problem of finding a group of objects which are connected by certain relationships and restrictions. In ontology, a semantic query is equal to a query graph with constrained object nodes and property arcs. An example query is *"Find all the states that the mississippi river runs through and border texas"* querying the geography ontology (Fig. 1). It can be rewritten as an equivalent conjunctive formal logic expression:

$$?x \leftarrow (?x, \mathbf{is}, State) \cap (Mississippi\ River, \mathbf{runThrough}, ?x) \cap (?x, \mathbf{border}, Texas)$$

where *class* '*State*', *instance* '*Mississippi River*' and '*Texas*' are ontological restrictions on nodes while '*border*' and '*runThrough*' are the required connecting arcs in the pattern. *'?x'* is the variable passing these restrictions on the query. So we can reduce the problem of translating keyword queries into formal queries to the problem of constructing equivalent query graphs from keywords. To clarify the problem, we give the formal definitions as follows:

[4] MRR: Mean Reciprocal Rank [16].

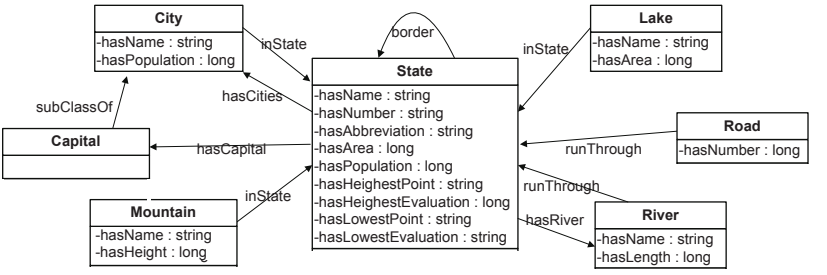


Fig. 1. Geography Ontology (Schema)

Knowledge base $D : \langle C, I, L, R, \tau \rangle$ is a directed graph G_D where: C and R define the sets of *class* and *relation*. I and L are the sets of *instance* and *literal* [5]. Function $\tau : (C \cup I) \times (C \cup I \cup L) \rightarrow R$ defines all the triples in D. Additionally, we use symbol resource $\{e\} : \{C \cup R \cup I \cup L\}$ to represent all *classes*, *relations*, *instances* and *literals*.

Keyword query K is a bag of terms $\{t\}$. In our assumption, the end users can have no knowledge of the underlying ontology and any arbitrary keyword queries can be issued by the users.

Formal query $F : \langle C^{'}, R^{'}, I^{'}, L^{'}, V, \tau^{'} \rangle$ over D is a graph G_F subsumed by G_D. V is the set of variable nodes which conjunct the relations and nodes. $\tau^{'} : (I^{'} \cup C^{'} \cup V) \times (I^{'} \cup C^{'} \cup V \cup L^{'}) \rightarrow R$ defines all the triples in F.

From the definitions above, the formal query construction problem can be modeled as:

> *Under knowledge base D, given a keyword query K, constructing and ranking candidate formal queries $\{F\}_{\preceq}$ such that the most likely query is among the highest ranked ones.*

For example, assume that a web user intends to express the information need "*Show me the states which the mississippi river runs through?*". He may type "*mississippi river state*" as his keyword query. In SPARK, the *formal query construction* process will map these keywords into knowledge base and complete the candidate formal query graphs under D. In resolving the ambiguity (term '*mississippi*' may refer to '*mississippi state*' or '*mississippi river*') and information loss (lack of relations among terms), there are many candidate formal queries with different senses as the construction results. The *query ranking* process then estimates a confidence score for each candidate formal query so that the more likely formal queries are ranked higher. Take the formal queries in Table 1 as an example, each one is assigned a ranking score (Column 'Total'). In the following section, we will illustrate the detailed approach in SPARK framework for *formal query construction*.

[5] To simplify the problem, we will treat all *literal* as *instance* in later sections.

Table 1. Some formal queries translated from keyword query *'mississippi river state'*

Formal Logic Representation	KQM	KBM	Total	Rank
$?x \leftarrow (?x, \mathbf{is}, \mathbf{State})$	0.160	0.495	0.0792	7
$?x \leftarrow (?x, \mathbf{is}, \mathbf{River})$	0.160	0.544	0.0870	6
$?x \leftarrow (?x, \mathbf{is}, \mathbf{State}) \cap (\textit{Mississippi River}, \mathbf{runThrough}, ?x)$	0.333	0.653	0.2898	1
$?x, \leftarrow (?x, \mathbf{is}, \mathbf{State}) \cap (?x, \mathbf{border}, \textit{Mississippi State})$	0.246	0.854	0.2100	3
$?x \leftarrow (?x, \mathbf{is}, \mathbf{River}) \cap (?x, \mathbf{runThrough}, \textit{Mississippi State})$	0.289	0.877	0.2534	2
$?x \leftarrow (?x, \mathbf{is}, \mathbf{State}) \cap (?x, \mathbf{hasCities}, \textit{Fall-River City})$	0.211	0.891	0.1880	4
$?x \leftarrow (?x, \mathbf{is}, \mathbf{State}) \cap (?x, \mathbf{hasCities}, \textit{Fall-River City})$ $\cap(\textit{Mississippi River}, \mathbf{runThrough}, ?x)$	0.129	0.979	0.1263	5

3 The SPARK Approach

The framework of SPARK consists of two modules (Fig. 2): *ontology processing* module and *formal query construction* module. When an ontology is selected as the underlying knowledge base, the *ontology processing* module automatically indexes its resources. The *formal query construction* module takes keywords as input, and returns a ranked list of SPARQL queries as output.

Once a user inputs a keyword query, the *term mapping* step uses a group of mapping methods to find the corresponding resources in the knowledge base according to user's keywords. Then, the *query graph construction* step enumerates all possible query combinations and applies *Minimum Spanning Tree* algorithm [7] to construct complete query graphs with different senses from the mapped resources, during which some missing relations or concepts will be made up in the query. Finally, the *query ranking* step evaluates the constructed formal queries from two perspectives: the keyword query model (KQM) and the knowledge base model (KBM). A ranked list of SPARQL queries will be given back to the end user. In the next three sections, we will detail the *term mapping* and *query graph construction* steps, and model the *query ranking* problem.

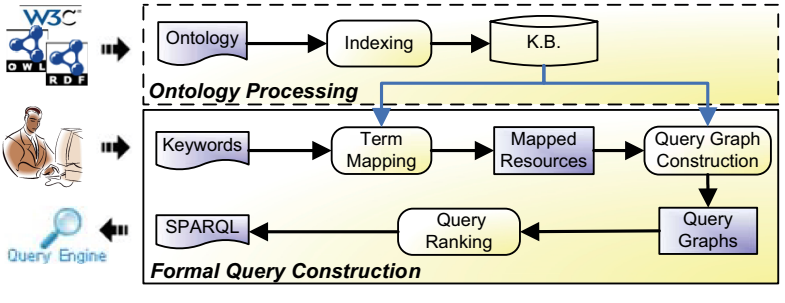


Fig. 2. SPARK Framework

3.1 Term Mapping

The purpose of *term mapping* is to find corresponding ontology resources (i.e. *classes*, *instances*, *properties* and *literals*) for each term in the keyword query. The names and labels of ontology resources are used for mapping. In our implementation, two types of mapping methods are employed: 1) *morphological mapping* employs string comparison techniques such as stemming, Sub-String, Edit-Distance, and I-Sub [8] to find morphologically similar words; 2) *semantic mapping* mainly utilizes general dictionaries like WordNet [11] to find semantically relevant words (e.g. synonyms). During a term mapping process, each term in K is matched against the knowledge base with different mapping methods, e.g. term *'river'* can be mapped to two ontology resources: *class 'River'* by direct mapping and *instance 'Fall-River City'* by Sub-String mapping. We assign a pre-defined confidence value $P(e|t)(P(e|t) \in (0, 1])$ to each mapping method to determine the mapping quality. Generally, the confidence value for a direct matching is higher than that for a synonym-based matching. Term mapping associates each term in keyword query with senses under the knowledge base. Therefore, after the term mapping process, a term is no longer a lexical string but represents a list of resources indicating what kinds of elements the user wants. In the next section, *query graph construction* process will construct candidate formal query graphs with different query senses from these mapped ontology resources.

3.2 Query Graph Construction

The *query graph construction* process builds up candidate query graphs with the ontology resources mapped above. Firstly, the mapped resources are split into different query sets. Then, *Minimum Spanning Tree* algorithm is applied to construct possible query graphs for each query set. Finally, each query graph is interpreted into a SPARQL query by conversion rules.

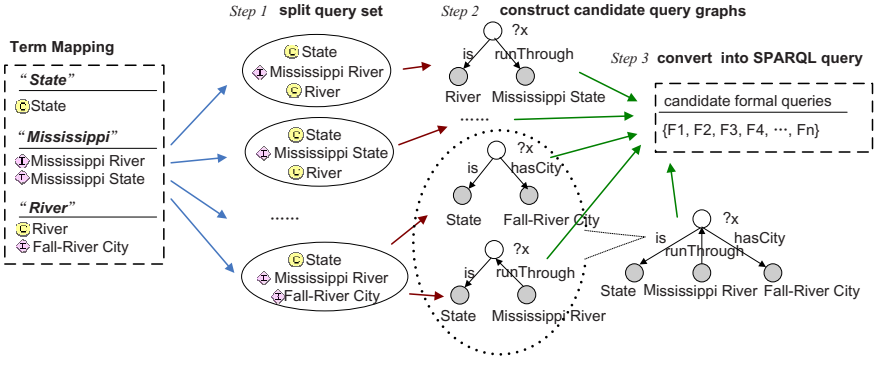


Fig. 3. Query Graph Construction

In the step of query set split: for each term t_i in K, if there is only one resource mapped from t_i, we directly add it into every F of query set $\{F\}$; otherwise we duplicate the original query set $\{F\}$ and add each mapped resource of t_i into every F in different sets. The purpose of query split is to assign a definite query sense to each formal query, which is a process of enumerating all possible combinations from different senses of each term.

After enumerating possible query set $\{F\}$, SPARK uses the *Kruskal's Minimum Spanning Tree* algorithm [7] to join these mapped resources of F into a complete query graph for each F in query set. Besides SPARK, the *Minimum Spanning Tree* has been introduced for inferring SQL query by DBExplorer [9]. In the *query graph construction*, SPARK makes up the missing relations and concepts for each query set by exploring its schema graph. Take the keyword query *'state mississippi river'* for example: there is no explicit relation between *'state'* and *'mississippi river'* expressed. From the schema graph of Geography Ontology (Fig. 1), one *relation 'runThrough'* entails between *instance 'Mississippi River'* and *class 'State'*. So this edge is introduced to construct our final query graph. We use these discovered nodes as variable nodes pointing to edges of query graph and discovered edges are added into the query graph. If a connected query graph can be joined with all resources in query set F, we directly regard it as the candidate formal query; otherwise we take the largest component instead.

Finally, we generate the query graph according to the following rules: 1) Resource *class* mapped by terms or discovered by graph exploration are regarded as variable nodes. 2) Resource *instance* and *literal* are regarded as end nodes. 3) Resource *property* are regarded as the edges of query graph. Since SPARQL is a graph pattern based query language, it is straightforward to convert the query graph into corresponding SPARQL query string.

3.3 Query Ranking

After the *term mapping* step and *query graph construction* step, multiple candidate formal queries will be produced from the original keyword query. There comes the problem: how to pick up the most likely formal query for the end users? In this section, *query ranking* is used to solve the problem. We model *query ranking* as: *"In knowledge base D, what is the probability of a constructed formal query F being a user issued query from the given keyword query K?"*. That is, we determine $P(F|D, K)$: the probability of generating event F under event D and K, which is the core idea of treating the query ranking problem as conditional probability event. The challenge turns to how to estimate this probability. Instead of estimating this probability directly, we apply *Bayes' Theorem* and obtain

$$P(F|D, K) = \frac{P(D, K|F)P(F)}{P(D, K)} \tag{1}$$

where $P(F)$ is the priori probability of formal query F. $P(D, K|F)$ is the probability of generating knowledge base and keyword from the constructed formal

query. We assume that the keyword query K and knowledge base D are independent events. Accordingly, $P(D, K|F)$ can be divided into two parts, and $P(F|D, K)$ can be written as:

$$P(F|D, K) = \frac{P(D|F)P(K|F)P(F)}{P(D)P(K)} \tag{2}$$

Formula 2 is not intuitive, so we apply *Bayes' Theorem* again into $P(D|F)$ and $P(K|F)$ in Formula 2)and obtain a more intuitive formula:

$$P(F|D, K) = \frac{P(F|D)P(F|K)}{P(F)} \propto P(F|D)P(F|K) \tag{3}$$

$P(F|D)$ is the probability of generating F from D. $P(F|K)$ is the probability of generating F from K. $P(F)$ is the priori probability of F. In this paper, we assume all the formal queries are in uniform distribution and $P(F)$ is equal among all candidate formal queries. Thus, the query ranking model is proportional to two sub probabilistic models: the keyword query model $P(F|K)$ and the knowledge base model $P(F|D)$. The next two sub-sections will illustrate the estimation of the two models in SPARK.

Keyword Query Model (KQM). The keyword query model reflects the probability of generating a formal query from a keyword query. Generally, there are two intuitive properties.

- *Mapping Proximity*: The ranking function should take keyword mapping proximity into account. In the keyword mapping, for every term t_i in K, the mapping probability is introduced to indicate the similarity between terms of K and resources of D. A formal query with resources of higher mapping scores should be given a higher priority than those with lower mapping score.
- *Relevance to Keyword Query*: The ranking function should rank the formal queries higher which are more relevant to user's keyword. The concepts mentioned in formal query with more keywords from user should be given a higher score.

Take the constructed queries in Table 1 for example, the third and the fourth queries are more reasonable from the perspective of the keyword query while the rest queries either have too few of user's expressed resources or add too many resources from the knowledge base. We define the captured information need of K by the distance measurement between K and F. To calculate the distance, we resolve the distance into two features: keyword mapping proximity $proximity(F, K)$ and query relevance $relevance(F, K)$:

Keyword mapping proximity score $proximity(F, K)$ bases on the average *term mapping* proximity $P(e|t)$ score. A higher $proximity(F, K)$ indicates that F is more relevant to user's terms.

$$proximity(F, K) = \frac{\sum_{e_i \in F} p(e_i|t)}{|t \in K|} \tag{4}$$

Query relevance score $relevance(K, F)$ is determined by the proportion of common resources in F and K. The proportion of common resources for K reflects how many resources in F are directly mapped from K. And the proportion of common resources for F reflects how many resources mapped from K contained in F.

$$relevance(F, K) = \frac{|(e \in F) \cap (P(e|t) > 0)|}{|t \in K|} \cdot \frac{|(e \in F) \cap (P(e|t) > 0)|}{|e \in F|} \tag{5}$$

Thus, the overall keyword query model is interpreted as:

$$P(F|K) \propto proximity(F, K) \cdot relevance(F, K) \tag{6}$$

Knowledge Base Model (KBM). We measure $P(F|D)$ for each formal query with its information content. Information content has been used by [10][12] for ranking the semantics of query. In information theory, information contained in an event is measured by the negative logarithm of the occurrence probability of the event. For example, given that $\{x_i\}$ is a discrete random event set with probabilities (p_1,. . . ,p_n and $\sum p_i$=1) of their occurrence, the information content of event x_i is given by $information(X = x_i) = -\ln p_i$.

In query graph, we measure the information content by its query graph patterns. Take the query graphs in Fig. 4 for example, the right one employs a more complex structure than that of the left one. Its question node ?x is not only constrained by the *instance 'Fall-River City'* but also the *instance 'Mississippi River'*. Obviously, its query result contains more information than those of the left query graph. In the query graph F_G, question node (?x) is modeled as the overall event

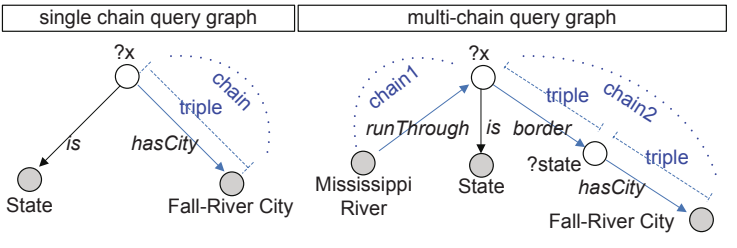


Fig. 4. Triple chain in query graph

and the path from it to end *instance* is viewed as a chain of events. Each event is a statement triple τ' like (?state, hasCity, *Fall-River City*). The right query graph in Fig. 4 consists of two chains. *chain1*: (?x, hasCity, *Fall-River City*) and *chain2*: (?x, border, ?state) (?state, hasCity, *Fall-River City*). Each τ' states an isolated event: given a resource (*Fall-River City*) or a variable node (?state), the probability of choosing the relation stated in τ' among all other relations is modeled as an event. The lower probability of choosing certain relation, the more information contained in τ'. Assuming a resource has a set of relations $\{R'\}$, the probability

of choosing relation $r^{'} \in \{R^{'}\}$ to construct $\tau^{'}$ is modeled as the probability of the event $\tau^{'}$. To determine the weight of relation, we assign the probability of choosing $r^{'}$ as its frequency in D. Frequency reflects the importance of the relation in knowledge base: the more triples share the relation, the more common the relation is. Therefore, the probability of event $\tau^{'}$ is:

$$P(\tau^{'}) = \frac{freq(r^{'})}{\sum_{r_i \in \{R'\}} freq(r_i)} \tag{7}$$

Given a chain with triples ($\tau^{'} \in chain$) from the question node to the end node. We assign the probability of the chain as its triple with lowest probability $P(chain) = min\{P(\tau^{'})\}_{\tau' \in chain}$. If multi-chain exists in F (Fig. 4), the event for the join variable node is determined by the overall probability of these separated event chain: $P(Event_F) = P(chain1) \cdot P(chain2)$. Thus, the information content can be computed by:

$$I(F) = -\ln(P(Event_F)) = -\ln \prod_{chain_i \in F} P(chain_i) \tag{8}$$

After computing the information content score of the query, we use *sigmod function* to adjust the information content score and estimate its probability $P(F|D)$ by.

$$P(F|D) \propto |\alpha - \frac{1}{1 + e^{-I(F)}}| \tag{9}$$

Using information content biases the formal queries with lower-probability. To balance the information content, we use a parameter $\alpha(\alpha \in (0, 1))$ for adjusting the ranking threshold. If the user wants the frequently-asked query, he can adjust α with corresponding slider in SPARK's web page. In the end, the *query ranking* process sorts these candidate formal queries by its overall probability score $P(F|K, D)$ from two sub-models.

4 Implementation and Experiment

SPARK is implemented in Java and Jena[6] API. It has a web interface[7] for online users. A user can choose the domain ontology, type his keywords and get a ranked list of SPARQL queries. These translated SPARQL queries can also be directly sent to ARQ[8] search engine to find related resources. Additionally, each SPARQL query has an automatically translated natural language query to clarify their information need.

In the rest of this section, we describe the experiments to validate the performance of SPARK and exploit its usability in querying various ontologies. Our goal is to observe the performance of the query ranking model as well as the query construction capability under different ontologies with various keyword queries. Additionally, we will also discuss the usability of SPARK from the experiences of end users.

[6] http://jena.sourceforge.net/
[7] http://spark.apexlab.org
[8] http://jena.sourceforge.net/ARQ/

4.1 Experiment Setup

Our experiment was performed on a PC with 3.2GHz Pentium(D) CPU and 2GB Memory. As far as we know, there is no test data specially designed for translating keyword queries on ontology. Therefore, we manually constructed these test knowledge base and keyword queries from Mooney Natural Language Learning Data [14], which has been used by many database and ontology-based querying experiments [3][13][15]. Firstly, we converted the test dataset (*geography*, *job* and *restaurant*) into RDFS ontologies. Then, we manually translated its natural language query into keyword queries for two purposes: to set up a test data set for similar applications and to make the evaluation more realistic. In the translating process, each natural language query was re-written as multiple short keywords query according to the understanding on it. For example: the keyword query '*state ohio river flow*' was translated from the natural language query *'show me all the states that river ohio runs through?'*. Then, these translated keyword queries were sent to SPARK and the constructed SPARQL queries were evaluated with the gold-standard SPARQL query created by its original natural language query. If the constructed SPARQL is not semantically equivalent to the gold-standard one, it is considered wrong.

We took two metrics in the evaluation: Recall and MRR. Recall indicates whether SPARK can construct at least one proper SPARQL query in its candidate queries while MRR focuses on the overall performance of SPARK. For each set of the constructed SPARQL queries by SPARK, we computed the performance by Reciprocal Rank (RR) of the first correct answer. If none of the SPARQL queries is correct, a score of 0 is given. Otherwise, the score is equal to the reciprocal of the rank of the first correct one. We calculated the mean reciprocal rank of all test keyword queries as MRR which is widely used in evaluating question-answering tasks [16].

4.2 Experiment Result

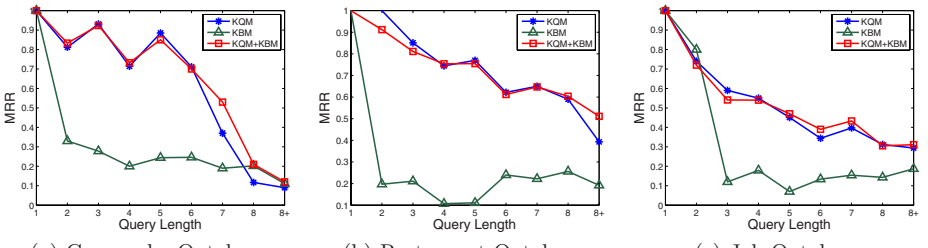
We used 250 keyword queries for each ontology. From the result in Table 2, the recall is 0.846, 0,824, 0.711 and the MRR is 0.755, 0.764, 0.513 for geography, restaurant and job ontology respectively, which is rather encouraging. The right constructed query is ranked as the first (MRR=1) or the second query (MRR=0.5) on average. From the evaluation result, SPARK can process most of the keyword queries entailing conjunctive relations such as '*city in virginia*', '*capital of the states, border new mexico*'. However, due to the limitation of keyword query, some of the semantics in keyword query can not be handled. For example: given *'area'* and *'population'* in ontology, we can't find a correct formal query for the keyword query *'state of smallest population density'*. Negation and superlative forms in keyword queries are inscrutable to be understood in current implementation, which is the main cause to loss in recall.

The ambiguity of keyword queries has been a big challenge since the invention of keyword search. To exploit the ability of processing ambiguous keyword query, we made an analysis of our test keywords and picked up those with ambiguous

Table 2. Evaluation Result over Geography, Restaurant, Job Ontology. (Query Total: count of all keyword query; AMB: count of ambiguous keyword query; AVL: average keyword query length.)

	Knowledge Base				Query			Result		Time
Ontology	Concept	Relation	Instance	Triple	Total	AMB	AVL	Recall	MRR	Avg(sec)
Geography	7	16	1018	3748	250	60	3.204	0.846	0.755	0.191
Restaurant	7	10	4315	108817	250	75	6.79	0.824	0.764	0.235
Job	12	19	11018	56868	250	49	6.24	0.711	0.513	0.249
Average	-	-	-	-	-	-	-	0.793	0.677	0.225

terms. There are 60(24%), 75(30%), 79(31.6%) ambiguous keyword queries in the evaluation set respectively. From the result, there is no obvious differences between unambiguous and ambiguous keyword queries. This is due to the tight coupling between the resources in query graph: the triple relation fit in with the intuition that resources with strong semantic meanings have other resources to reinforce one another. For the keyword query: *'mississippi city'*, term *'Mississippi'* may refer to the *instance 'Mississippi River'* or *instance 'Mississippi State'*. In the knowledge base, *' Mississippi State'* has a relation *property 'hasCity'* while *'Mississippi River'* does not. Therefore, the final formal query with *instance 'Mississippi State'* is ranked with higher overall score.



(a) Geography Ontology (b) Restaurant Ontology (c) Job Ontology

Fig. 5. MRR score for keyword query model (KQM), knowledge base model (KBM) and integration of two models (KQM+KBM)

In Fig. 5 we present comparison of ranking performance of keyword query model (KQM) and knowledge base model (KBM). As we can see, based on the integration of the two models, SPARK outperforms the best. Compared to the knowledge base model, the keyword query model contributes to the ranking performance more. However, with the increase in query length, the performance of keyword query model decreases. In that case, the knowledge base query can smooth the performance of the overall ranking score. The overall performance on keyword query of different length indicates that our query ranking model works best on the medium and short sized keyword queries (2-6 terms), which are the most frequently-used in today's web search engine. From the statistics

on time cost (Table 2), the time to process a keyword query takes 0.225 second on average, which is efficient for the implementation.

Besides the evaluation on test data set, we also made a study on the user experience of SPARK. From the feedback of 50 online users in two months, SPARK can construct correct SPARQL queries from their keywords most of the time. They thought that it is very easy for them to locate their desired information in ontologies quickly. The query ranking is effective: almost all of the right SPARQL queries can be found within the first three queries. Additionally, SPARK can handle abbreviated query. For example, query *'hill of nm'* issued by user can be easily translated into a formal query representing *'Show me all the mountains in the state of New Mexico'* in SPARK.

5 Related Work

In the last few years, there has been increasing interest in applying keyword query to structured data such as XML [17] and Relational Database [9][18][19]. The attractions are that users can keep the habit in traditional web search and do not need to know about the data schema. Banks [18] works by starting shortest-path search on each matching element in order to find out a join tree. DBExplorer [9] and Discover [19] take advantage of data schema to compute a set of possible join networks. In the context of semantic web, there has been little work on inferring candidate query graphs from keywords, which is very important to semantic search.

Faced with the gap between the formal logic based semantic query and the end users, some communities have proposed various solutions to narrow it: Bernstein et al. explore providing controlled language [15] and guided natural language interface [3]. From the perspective of query refinement [12] [20], the gap between users' information needs and its semantic querying is quantified by measuring several types of query ambiguities through incremental interaction. Graphical based search [21] also contributes a way, by building graph queries through browsing and selection on ontology.

Compared to these semantic search methodologies, using keywords lowers the formal scaffolding of semantic search. SemSearch [6] has a little-structured keyword query interface to hide the complexity of semantic search. Avatar Semantic Search [22] is a prototype search engine that exploits annotations in the context of classical keyword search. Another representative keyword based semantic search application is OntoLook [5]: a prototype relation-based search engine. OntoLook mentioned the weakness of keyword query in the context of semantic web and inferred possible relations among keywords to improve the precision of the search. Compared to these applications, SPARK not only covers the relation and concept missing problem but also gives solutions to the problems of query ranking and semantic matching proposed in SemSearch.

Many methodologies on deciding the best query have been proposed before SPARK: Ontologer [23] builds a query mechanism by recording user's behaviors in an ontology and recall it for ranking. Banks [18] incorporates different weight

on vectors into the relevance score. SemRank [10] uses information gain theory to rank the discovered path between two resources. To the best of our knowledge, most of the ranking approaches are feature based and there is no work today addressing the query ranking problem systematically. In this paper, we give definition of the problem and present an effective query ranking implementation.

6 Conclusion and Future Work

In this paper, we formalize the formal query construction problem and present an effective approach in SPARK to resolve it. SPARK aims at translating keyword queries into SPARQL queries to narrow the gap between formal logic based semantic search and end users. Additionally, the probabilistic ranking model implemented in SPARK can well explain what the desired properties are for a likely constructed formal query translated from keywords.

The main contributions of this paper are: 1) it puts forward the problem of translating keywords into formal logic based queries for semantic search; 2) it presents a novel solution to this problem which is implemented as SPARK that translates keywords into SPARQL queries; 3) it provides a probabilistic query ranking model for picking most likely translated formal queries from keywords. From the evaluation of 750 keyword queries over three ontologies, SPARK achieved an encouraging translation result.

In future work, we consider to enhance SPARK in two ways: 1) enlarge its query scope by introducing some structured operators (e.g. NOT, OR, etc) and improving human interaction support to translate more complicated information needs. 2) extend SPARK's approach to multiple ontologies for web-scaled usage.

References

1. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: A Declarative Query Language for RDF. In: WWW 2002. Proceedings of the 11th international conference on World Wide Web, Honolulu, Hawaii, USA, pp. 592–603. ACM Press, New York (2002)
2. Miller, L., Seaborne, A., Reggiori, A.: Three Implementations of SquishQL, a Simple RDF Query Language. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, Springer, Heidelberg (2002)
3. Bernstein, A., Kaufmann, E.: GINO - A Guided Input Natural Language Ontology Editor. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. (eds.) ISWC 2006. LNCS, vol. 4273, Springer, Heidelberg (2006)
4. Chakrabarti1, S.: Breaking Through the Syntax Barrier: Searching with Entities and Relations. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) PKDD 2004. LNCS (LNAI), vol. 3202, pp. 9–16. Springer, Heidelberg (2004)
5. Li, Y., Wang, Y., Huang, X.: A Relation-Based Search Engine in Semantic Web. Proceedings of IEEE Transactions on Knowledge and Data Engineering 19(2), 273–282 (2007)
6. Lei, Y., Uren, V., Motta, E.: SemSearch: A Search Engine for the Semantic Web. In: Proceedings of EKAW 2006, pp. 238–245 (2006)

7. Kruskal, J.: On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In: Amer. Math. Soc. (1956)
8. Stoilos, G., Stamou, G., Kollias, S.: A String Metric for Ontology Alignment. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 623–637. Springer, Heidelberg (2005)
9. Chaudhuri, S., Das, G., Narasayya, V.: DBExplorer: A System for Keyword Search over Relational Databases. In: Proceedings of Data Engineering 2002 (2002)
10. Anyanwu, K., Maduko, A., Sheth, A.: SemRank: ranking complex relationship search results on the semantic web. In: Proceedings of WWW 2005, Chiba, Japan, pp. 117–127. ACM Press, New York (2005)
11. Miller, G.A.: Wordnet: a lexical database for english. Commun. ACM 38(11), 39–41 (1995)
12. Stojanovic, N., Stojanovic, L.: A Logic-based Approach for Query Refinement in Ontology-based Information Retrieval Systems. In: Proceedings of the 16th IEEE Int. Conf. on Tools with Artifical Intelligence, Illinois USA, IEEE Computer Society Press, Los Alamitos (2004)
13. Popescu, A.M., Etzioni, O., Kautz, H.A.: Towards a theory of natural language interfaces to databases. In: IUI, pp. 149–157 (2003)
14. Tang, L.R., Mooney, R.J.: Using Multiple Clause Constructors in Inductive Logic Programming for Semantic Parsing. In: Flach, P.A., De Raedt, L. (eds.) ECML 2001. LNCS (LNAI), vol. 2167, Springer, Heidelberg (2001)
15. Bernstein, A., Kaufmann, E., Gohring, A., Kiefer, C.: Querying ontologies: A controlled English interface for end-users. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, Springer, Heidelberg (2005)
16. Voorhees, E.: Overview of the TREC 2001 Question Answering Track. In: Proceedings of TREC-X, Gaithersburg, Maryland, pp. 157–165 (2001)
17. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: XRANK: Ranked Keyword Search over XML Documents. In: Proceedings of SIGMOD 2003, San Diego, California, pp. 16–27. ACM Press, New York (2003)
18. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S.: Keyword Searching and Browsing in Databases Using BANKS. In: Proceedings of ICDE 2002, Illinois USA, ACM Press, New York (2002)
19. Hristidis, V., Papakonstantinou, Y.: DISCOVER: Keyword Search in Relational Databases. In: Proceedings of VLDB 2002 (2002)
20. Carlos A. Hurtado, A.P., Wood, P.T.: A Relaxed Approach to RDF Querying. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. (eds.) ISWC 2006. LNCS, vol. 4273, Springer, Heidelberg (2006)
21. N Athanasis, V.C., Kotzinos, D.: Generating On the Fly Queries for the Semantic Web: The ICS-FORTH Graphical RQL Interface (GRQL). In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, Springer, Heidelberg (2004)
22. Kandogan, E., Krishnamurthy, R., Raghavan, S., Vaithyanathan, S., Zhu, H.: Avatar Semantic Search: A Database Approach to Information Retrieval. In: Proceedings of SIGMOD 2006, pp. 790–792. ACM Press, New York (2006)
23. Stojanovic, N., Gonzalez, J., Stojanovic, L.: Ontologer: A System for Usage-Driven Management of Ontology-Based Information Portals. In: Proceedings of L-CAP 2003 (2003)