Query-By-Keywords (QBK): Query Formulation Using Semantics and Feedback*

Aditya Telang, Sharma Chakravarthy, and Chengkai Li

Department of Computer Science & Engineering
The University of Texas at Arlington, Arlington, TX 76019
{aditya.telang,sharmac,cli}@uta.edu

Abstract. The staples of information retrieval have been *querying* and *search*, respectively, for structured and unstructured repositories. Processing queries over known, structured repositories (e.g., Databases) has been well-understood, and search has become ubiquitous when it comes to unstructured repositories (e.g., Web). Furthermore, searching structured repositories has been explored to a limited extent. However, there is not much work in querying unstructured sources. We argue that querying unstructured sources is the next step in performing focused retrievals. This paper proposed a new approach to generate queries from search-like inputs for unstructured repositories. Instead of burdening the user with schema details, we believe that pre-discovered semantic information in the form of taxonomies, relationship of keywords based on context, and attribute & operator compatibility can be used to generate query skeletons. Furthermore, progressive feedback from users can be used to improve the accuracy of query skeletons generated.

1 Motivation

Querying and search have served well, respectively, as mechanisms for retrieving desired information from structured and unstructured repositories. A query (e.g., SQL) is typically quite precise in expressing *what* information is desired and is usually formed with the prior knowledge of the data sources, the data model, and the operators of the data model. On the other hand, a search request (typically, in the form of keywords) is on text repositories (including HTML and XML) and does not assume the knowledge of sources or the structure of data. A search is likely to generate a large number of results (especially when the search corpus is large and the input, by definition, is imprecise with respect to the intent) and hence ranking or ordering the results to indicate their relevance or usefulness has received considerable attention. Queries, in contrast, generate more focused results and can also be ranked to generate top-k answers.

One of the significant differences between querying and search is that some forms of querying require training – in the query language, the data model on which the query is being expressed, and the syntax of the query language. It

* This work was supported, in part, by the NSF Grant IIS 0534611.

A.H.F. Laender et al. (Eds.): ER 2009, LNCS 5829, pp. 191–204, 2009.

also requires an understanding of the sources or schema. As a result, querying is the proclivity of those who are willing to spend time learning the intricacies of correct specification. In contrast, search is straightforward and easy, and as a result, has become extremely popular with the advent of the Web. The lack of a learning curve associated with search makes it useful to a large class of users. The proliferation of search and its popularity has lead researchers to apply it to structured corpus as well [1]; however, the main problem (from a users' viewpoint) being the post-processing effort needed to filter out irrelevant results.

Consider a user who wants to – "*Retrieve castles near London that can be reached in 2 hours by train*". Although all the information for answering the above request is available on the web, it is currently `not possible` to frame it as a *single query/search* and get meaningful results. Since this and similar queries (Section 3 provides examples of such queries) require combining information from multiple sources, search is not likely to be a preferred alternative. Although there are a number of systems [2], [3] that combine information from multiple sources belonging to the *same* domain (e.g., books, airlines etc.), they cannot answer the above class of queries.

The focus of this paper is to address the problems associated with querying unstructured data sources, especially the web. The *InfoMosaic* project [4] is investigating the integration of results from multiple heterogeneous Web sources to form meaningful answers to queries (such as the one shown above) that span multiple domains. As part of this project, we are investigating a mechanism for formulating a complete query from search-like keywords input (to avoid a learning curve) that can be processed over multiple unstructured domains to retrieve useful answers. Additionally, we are also investigating the generation and optimization of a query plan to answer the formulated structured query over the Web. More recently, this problem is also being addressed in the literature [5].

An intuitive approach to both search and querying would be to use natural language to express the query. This is certainly a preferred alternative as it frees the user from the data model and other considerations needed for expressing a structured query. We view structured queries being at one end of the spectrum and natural language queries being at the other end. Since the general capability to accept arbitrary natural language queries and convert them to structured queries is still not mature, our proposed approach (that lies somewhere in-between the two and even easier than a natural language) will provide a mechanism for querying unstructured repositories with intuitive inputs and minimal user interaction.

This paper advances the thesis that a query is essential to extract useful and relevant information from the web – especially when it involves integrating information that spans multiple domains and hence multiple disparate repositories. As most of the users are familiar and comfortable with search, we want to start with a search-like input and expand it to a complete query using different types of semantic information and minimal user feedback/interaction. For instance, in the case of the above query example, a user may provide his input as a set of the following keywords: *castle, train, London* in any order. For the same query,

another user may input: *train, 2 hours, London, castle*. These list of words may mean several different things to different users. For example, the user may be looking for a castle in London or a book written by London in which Castle and Train appear in the title. The semantic information (that is assumed to be separately discovered and collected for this purpose) will assist in formulating the correct complete query with minimal interactions with the user.

As alluded to above, our proposed approach uses different types of semantic information: taxonomies (for context information such as travel or publishing in the above example), attributes associated with concept nodes in the taxonomy, their types, and whether they can participate in join, spatial or temporal conditions, alternative meanings of words as they appear in multiple taxonomies, compatibility of attributes across concept meanings, dictionary meaning and priority of word semantics from the dictionary to the extent possible, and finally user feedback on past interactions. The remainder of the paper will elaborate on the proposed Query-By-Keywords (QBK) approach that uses different types of semantics and workload statistics to disambiguate and generate a partial query to capture the intent of the user.

1.1 Contributions and Roadmap

One of the contributions of this paper is the novelty of the approach proposed to generate a structured query from an input that is characteristic of search. Other important contributions include: the identification of appropriate semantic information (e.g., taxonomies and other associated data) for the transformation of keywords, algorithms for generating alternative query skeletons and ranking them using compatibility and other metrics. Finally, the role and use of feedback for improving the ranking and generating the complete structured query.

We would like to clarify that the scope of this paper does not include the discovery (or automatic acquisition) of information used for the proposed approach. The thrust of this paper is to identify what information is needed, establish the effectiveness of this information, and an approach for transforming input keywords into a complete query. The discovery of this information, an independent problem in itself, is being addressed separately.

The rest of the paper is organized as follows. Section 2 contains related work. Section 3 provides an overview of the proposed approach with motivating examples of user intent, keyword input, and alternative queries that are generated by our approach. Section 4 discusses the details of steps for transforming input keywords into a complete query including keyword resolution, ranking, and query template generation. Conclusions and future work are in Section 5.

2 Related Work

The work closest to ours in formulating queries using templates/skeletons with multiple interactions from the user is the popular Query-By-Example (or QBE) paradigm [6]. In addition, template-based query formulation using multiple interactions with the user has been developed for database systems such as SQL

Server, Oracle and Microsoft Access. Similarly, the CLIDE framework[7] adopts a multiple-interaction visual framework for allowing users to formulate queries. The primary motivation of the CLIDE architecture is to determine which queries would yield results versus those which produce a null result-set. However, formulating queries using these mechanisms requires the user to have knowledge about the types of queries supported by the underlying schema as well as a minimal understanding of the query language of the data model. Deep-web portals such as Expedia (www.expedia.com) or Amazon (www.amazon.com) support the QBE paradigm; however, the queries to these systems are restricted to the schema of a single domain such as travel, shopping, etc. and thus, lack the flexibility to support complex queries that span multiple domains. To the best of our knowledge, the problem of formulating arbitrary queries that span multiple domains has not been addressed.

Search engines (e.g., Google) and meta-search engines (e.g., Vivisimo [8]) use the keyword query paradigm and its success forms the motivation for this work. However, they do not convert the keywords into queries as their aim is not query processing. Although some search engines (e.g., Ask.com) accept natural language input, we believe that they do not transform them into structured queries. Deep Web portals, on the other hand, support search through templates, faceted searches and natural language questions (e.g., START [9]). However, since the underlying schemas in these systems are well-defined and user queries are purely based on these schemas, the need to support arbitrary queries/intents with varying conditions on multiple types of operators does not arise.

Frameworks that support queries on multiple sources support either a keyword query paradigm [2] or mediated query interfaces [3][4] for accepting user intents. Similarly, commercial systems such as Google Base [10] advocate the usage of keyword queries. Faceted-search systems [11] support query formulation using keywords in multiple navigational steps till the user gets the desired results. However, the focus of these frameworks is to perform a simple text/Web-search to obtain different types of data in response to the keywords (e.g., blogs, web-links, videos, etc.) instead of formulating a query where every keyword corresponds to a distinct entity.

3 Overview of QBK Approach

User queries that span across multiple domains (such as Travel, Literature, Shopping, Entertainment etc.) and involve different join conditions across sources in these domains can be complex to specify. For example, consider some representative queries that users would like to pose on the web:

Q1: *Retrieve castles near London that are reachable by train in less than 2 hours*
Q2: *Retrieve lowest airfare for flights from Dallas to VLDB 2009 conference*
Q3: *Obtain a list of 3-bedroom houses in Houston within 2 miles of exemplary schools and within 5 miles of a highway and priced under 250,000$*

Although all the information for answering the above (and similar) intents is available on the Web, it is currently not possible to pose such queries. Ideally, it

should be possible to accept minimal input that characterizes the above queries from the user, and refine it into a *complete* query (such as the one shown below in response to $Q1$) to reflect the user intent.

```
SELECT *
FROM
  SOURCES www.castles.org, www.national-rail.com
                  /* Using the travel domain */
  FOR ENTITIES castle, train
WHERE
   train.source = 'London' and
   train.destination = castle.location and
   train.start_date = 09/19/2008 and
   train.return_date = 09/19/2008 and
   train.duration < 02 hours /* temporal conditions */
```

The approach we are proposing is to generate the above complete query by accepting a set of keywords (can also be extracted using a natural language specification). It may be possible to derive the above query *completely* from the keywords given for $Q1$ by making some minimal default assumptions about the dates based on the context. However, if a set of keywords are input, the generation of a complete query may not always be possible. Hence, we have introduced the notion of a query skeleton in this paper.

Web users are comfortable expressing queries through keywords rather than a query language (as displayed by the popularity of search and meta-search engines). Furthermore, current language processing techniques do not posses the ability to process and translate any arbitrary natural language query into a structured query. Hence, it is preferable for the user to express a query using a set of representative words than in natural language. For instance, some of the possible keyword representations for $Q1$ could be:

$Q1_{K1}$: *castle, train, London*
$Q1_{K2}$: *train, from, London, to, castle*
...
$Q1_{Kn}$: *castle, reachable, train, from, London, 2 hours, or, less than*

The above can also be extended to specify phrases/conditions instead of only keywords. For example, "less than 2 hours" can be expressed together rather than separately. The phrase needs to be parsed with respect to a context. Irrespective of how the intent is mapped into keywords (e.g., alternatives $Q1_{K1}, Q1_{K2}, ..., Q1_{Kn}$ shown above), the final query formulated by the system in response to all these different inputs should correspond to $Q1$. Of course, this may not be possible without some interaction with the user once the domains and potential partial queries are identified and generated by the system. On the other hand, it is also possible that different user intents may result in the same set of keywords introducing ambiguity that needs to be identified and resolved systematically in order to arrive at the final query intended by the user. As an example, the following natural language queries can result in the *same* set of keywords from a user's perspective.

- *Retrieve Castles near London that are reachable by Train*
- *Retrieve Hotels near London that are Castles and can be reached by a Train*

Thus, for formulating query skeletons that converge to the actual user intent, it is necessary for the underlying system to intelligently correlate the specified keywords and generate alternative query skeletons based on the interpretation of the keywords in different domains. It is also possible that, within the same domain, multiple query skeletons can be generated by using alternative interpretations of keywords.

3.1 Specification, Precision, Usability, and Learning Tradeoffs

It is clear that there is a tradeoff between ease of query specification (or learning effort), its precision, and utility of results. Search is easy to specify but inherently vague and the result has to be sifted to obtain useful or meaningful answers (low utility). Although ranking helps quite a bit, as ranking is not always completely customized to an individual user, results need to be pruned by the user. On the other hand, a structured query is precise and the user does not have to interact with the system for obtaining meaningful answers (high utility). Of course, ranking can further help bring more meaningful answers to the top (or even avoid computing others).

Table 1. Specification Comparison Matrix

Specification	Learning Curve	Precision	Utility	Schema/Source Knowledge
SQL	High	High	Med-high	High
QBE	Low	High	Med-high	Medium
Templates	Low	High	Medium	Low
NL	Low-Med	Medium	Med-high	Low-Med
Search	Low	Low	Low	Low
QBK	**Low**	**High**	**High**	**Low**

Table 1 shows a back-of-the-envelope comparison of various search/query specifications across the dimensions of learning effort, precision, utility, and knowledge of schema/sources. The ultimate goal is to have a query specification mechanism that has low learning effort, precise, high utility, and does not require the knowledge of sources. QBK is an attempt in that direction as shown in the bottom row. The purpose of the table is to quickly understand a specification mechanism along a number of metrics and learn how it stacks up against other mechanisms. The table does not include specifications such as faceted search as it is navigational wit results refined at each step. The score of Medium-high utility for SQL and QBE is based on whether ranking is used or not. The Natural language (NL) row assumes ambiguities in the specification and hence the utility of results may not be high. This table can be used as a starting point to to compare different approaches used for search as well as for querying.

4 Details of the QBK Approach

Our approach to completing multi-domain queries is shown in Figure 1. In our approach, the user provides keywords deemed significant (e.g., $\{castle, train, London\}$ for *Q1*) instead of a query. The *Keyword Resolution* phase checks these keywords against the *Knowledge Base* for resolving each keyword to entities, attributes, values, or operators. For a keyword that occurs as a heteronym (i.e., same entity representing multiple meanings/contexts) in the *Knowledge Base*, all possible combinations for the different meanings of this keyword is generated. This occurs when the keyword occurs in different taxonomies corresponding to different domains/context. From these combinations, query skeletons and any other conditions (or attributes on which conditions are possible) are generated.

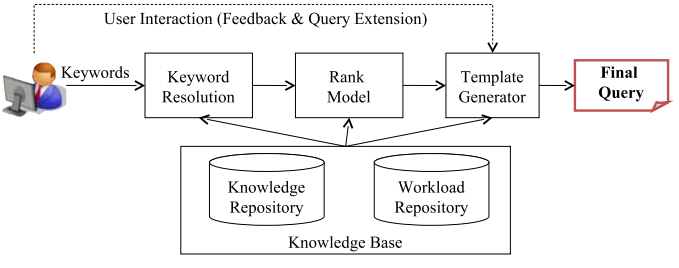

Fig. 1. Keywords to Query Formulation Steps

These query skeletons are ranked by the *Rank Model* in the order of relevancy (to the input and past usage) and shown to the user to choose one that corresponds to his/her intent. Both keyword resolution and ranking is based on the information in the *Knowledge Base*. Subsequently, a template is generated with all the details which can be further filled with additional conditions. The list of entities that are of interest, the domain and sources to which they map, and the possible list of simple conditions (e.g., $train.startTime < relationaloperator > value$) or attributes as well as join conditions (e.g., $castle.location < traditional/spatial/temporaloperator > train.startLocation$) is shown. Additionally, a list of attributes is displayed for the choice of result attributes. The user fills/modifies the template in a manner similar to filling a Web query interface so that an unambiguous and complete query can be generated for further processing.

4.1 Knowledge Base

The knowledge Base consists of a *Knowledge repository* that contains taxonomies organized by domains/context including meta-information about the entities, sources, operators, attributes and values. This is used in the keyword resolution phase and for constructing query skeletons. The knowledge base also consists of a *Workload repository* that organizes the past user of query skeleton for a

set of input keywords as well as the conditions provided and output attributes selected. Workload repository, when available and statistically significant, is used for ranking.

Knowledge Repository: This repository contains pre-discovered *semantic* information in the form of a collection of taxonomies associated with domains and are populated with appropriate types of meta-data. For instance, the domain of *Travel* can be represented using taxonomies for – *transportation*, *travel lodging*, and *tourist attractions*. Similarly, the domain for *Literature* may contain taxonomies such as *journal*, *book*, etc.. These represent the roots of different taxonomies within the given domain. Nodes in each taxonomy represent entities (e.g., castle, museum, church, etc.) associated with the domain corresponding to a *is-a* relationship[1].

In addition to the *is-a* hierarchy, supplementary meta-data is associated with each node in a taxonomy. For instance, an entity *castle* in the *tourist attractions* domain may have several types of associated meta-data: i) *Web sources* (e.g., www.castles.org) from which attribute values relevant to this entity can be extracted, ii) common *attributes* (e.g., name, age, country_location) that are associated with the entity, and iii) *semantics* representing linguistic meaning and semantic popularity. Additionally, each attribute of the entity has a number of meta-data: i) data type of the attribute (e.g., string, numerics, etc.), ii) attribute category (spatial, temporal, generic), iii) possible attribute value range, and iv) synonyms. For leaf-level entities in a taxonomy, the *values* for certain categorical attributes are associated. This is needed to resolve keywords that are values themselves (e.g., London) and infer the entity for which it is a value (e.g., city). As this set can be arbitrarily large, a way to infer them (using something similar to a WordNet) instead of storing all the values is going to be important. The list of relevant Web sources corresponding to an entity can be obtained using search engines and the information associated with Web directories. The *entity-linguistic-score* based on its linguistic meaning can be captured using WordNet [12].

In addition to meta-data, the knowledge repository also contains information about the *compatibility* between entities. Two entities are said to be *compatible* if a meaningful *join* condition can be formulated between (one or more) attributes of the participating entities. For instance, the entities *castle* and *train* are compatible since their respective attributes *location* and *startLocation* can be compared. The join could result in traditional, spatial, or temporal conditions based on the attribute types and the operators associated with them. This compatibility across entities can be identified by matching the respective attribute data types. A compatibility matrix can be used as the number of operators is not that large. Compatibility information of successfully formulated past queries from the workload repository can also be used for this purpose. Another component of this repository is a list of operators that are applicable to attributes.

[1] In this paper, we assume the availability of such taxonomies. Simple taxonomies can be generated using a combination of Web directories (e.g., Yahoo Dictionary) and dictionaries (e.g., Webster).

We assume – simple relational operators (==, ! =, <, <=, >, >=), Boolean operators, temporal operators (derived from Allen's Algebra [13]), and a few spatial operators (such as near, within, etc. [14]).

It is evident that building this comprehensive knowledge repository is a separate problem in itself and is beyond the scope of this paper. Given such a repository, its completeness and algorithmic usage in formulating queries is of concern here.

Workload Repository: Past statistics based on user interaction can play an important role in the query formulation process as it indicates significance of user preferences. Hence, we maintain a repository that is a collection of statistics and feedback associated with past queries. Specifically, it comprises of the information associated with users' preference of the query skeletons from those generated by the *Rank Model*. Additionally, statistics of the attributes of entities used for specifying conditions or output are also collected. This information is used for choosing widely preferred attributes for an entity in the generation of skeletons and templates. This repository is constructed using the feedback collected as a by-product of the query formulation process. For every keyword, the following user preferences are collected: i) context of the individual keyword (e.g., *castle* belonging to the travel domain frequently chosen over others) and ii) frequency of the attributes used for specifying conditions or chosen for output.

4.2 Keyword Resolution

The purpose of this phase is to map specified keywords into the domains that are stored as part of the knowledge base. Coverage of keywords in each domain is important as it indicates the relevance of the domain to the input. This phase performs matching for each of the input keywords with – *entity* names in the taxonomies, *attribute* names associated with each node (entity) in the taxonomies, and *values* associated with leaf-level entity attributes.

Since a domain comprises multiple taxonomies, it is possible (as shown in Figure 3) that the same keyword (*castle*) will belong to multiple taxonomies (*tourist attractions* and *travel accommodations*) in a single domain (*Travel*). In such cases, determining which intent the user has in mind is not possible. Hence, the resolution phase checks for multiple instances of the same keyword in different taxonomies, each giving rise to a separate entity set (and hence, separate query intents) for each occurrence of the entity. Additionally, it is also possible for the same keyword to occur in taxonomies in multiple domains (*castle* occurs in the domains of *Travel* and *Literature* as shown in Figure 3). Hence, the resolution phase analyzes all the domains independently to get the list of entity sets within each domain.

It is possible that the keyword may not always match to an entity in the taxonomy i.e., it may match to an attribute of an entity or the value of an entity's attribute. In such cases, the immediate parent entity (to which the attribute/value belongs) is chosen to represent the input keyword. Further, the keywords not finding a match to any of these categories are compared against a

keyword	occurs as	taxonomy	domain	entity-attribute-value association
castle	entity	tourist attractions	travel	-
	entity	travel accommodation	travel	-
	value	literature	literature	book.title
train	entity	transport mode	travel	-
	value	literature	literature	book.title
	value	movie	entertainment	movie.title
london	value	city	geography	city.name
	value	literature	literature	book.author
	value	business	industry	company.name
	value	movie	entertainment	movie.title

Fig. 2. Keyword Resolution: $Q1_{K1}$

Entity Set	Domain	Taxonomies
castle, train, city	Travel	Tourist attractions, transport mode, city
castle, train, city	Travel	Travel accommodations, transport mode, city
book	Literature	Literature
movie	Entertainment	Movie

Fig. 3. Query Space: $Q1_{K1}$

set of operators to determine their occurrences as a spatial, temporal, or generic operators and an operator-list. These operator keywords do not occur in any entity set generated. The keywords that do not find any match are ignored.

Thus, for a given set of input keyword, the resolution process generates a list of entity sets belonging to the same or multiple domains where every set comprises of entities that belong to the same domain (but might map to multiple taxonomies within that domain). For instance, the outcome of the resolution process for intent $Q1_{K1}$ (for $Q1$) (based on the keyword matching results shown in Figure 2) is shown in Figure 3. As the figure indicates, it is possible that the given input may generate multiple combinations (of entities). This would make the task of separating the relevant intents from the irrelevant ones extremely hard. Hence, in order to establish an order amongst these combinations, the output generated by the *resolution* phase is fed to the *Rank Model*.

4.3 Rank Model

Since the set of entity combinations generated by the resolution process can be large, we propose a ranking technique to order these combinations based on the different characteristics of the entities in the an entity-set namely – i) linguistics, ii) statistics, and ii) join compatibility. In the rest of the section, we elaborate on each of these parameters, and explain their importance in ranking entity sets and discuss the mechanism to fuse them in a single ranking function.

Linguistics: It is a common observation [15] that *linguistic meaning* of a keyword plays an important role when users specify their input. For instance, as per WordNet [12], the keyword *castle* is considered to have the following linguistic meanings: i) a large and stately mansion, and ii) a piece in the game of chess. However, as established by WordNet, users generally use an established language model [16] to formulate their natural language queries as well as search queries. That is, they choose the meaning which is linguistically more popular than the rest of the meanings for the same keyword (in this case the former meaning of *castle* will be chosen over the latter in most cases). Thus, there is reason to believe that when users express their keyword input to formulate queries over

multiple domains, they will use a similar language model of picking linguistically popular meanings for an entity.

Based on this observation, we assign a *entity-linguistic-score* to every entity in a taxonomy. The linguistic meanings for a given entity are obtained from WordNet, which gives the meanings for an entity in a ranked order such that the most popular linguistic meaning has a rank of 1. However, since we need to calculate a combined *linguistic score* for every entity set generated by the resolution process, we normalize WordNet's rank into a *entity-linguistic-score* (given by Equation 1). For an entity e_i whose WordNet rank is e_{rank}, and for which WordNet has a list of n distinct meanings:

$$entity\,linguistic\,score(e_i) \;=\; 1 \;-\; \frac{e_{rank} - 1}{n - 1} \tag{1}$$

Given an entity set $\{e_1, e_2, ..., e_n\}$, the ***linguistic-score*** for an entity set is calculated as a *product* of the individual *entity-linguistic-scores* using the independence assumption. That is, the entities in a set are not associated with each other in a linguistic context. For example, the *linguistic score* for the entity set $\{castle, train, andcity\}$ representing the input $Q1_{K1}$ is calculated as a product of the individual *entity-linguistic-scores* associated with *castle*, *train*, and *city*.

Statistics: Although linguistics can play an important role in ordering entity sets, it is based on a global language model which is static thus gives the same ordering for a given input. However, this order can be different as observed by the user behavior. Hence, in addition to linguistics, we also analyze the entity sets in terms of past user preferences i.e., we use the *usage statistics* associated with past user queries from the *workload repository* in our *Knowledge Base* as another component of the ranking model.

Based on the workload, an entity set that has been selected more number of times for the *same* input would rank higher. However, as is the case with ranking in general, it is not possible to have an exhaustive query workload which covers every possible query. In the absence of an exact query, we make the *independence assumption* i.e., we consider the statistics for individual entities of the combination and apply the product function to combine them. Hence, the statistics score for an entity set $\{e_1, e_2, ..., e_n\}$ is the product of the individual *entity-statistics-score*. For instance, the keywords input $Q1_{K1}$ may not exist in the query workload but entities *castle*, *train* and *London* may exist in different queries, either together with additional keywords or independently with other keywords. In this case, the *statistics score* for an entity set $\{castle, train, city\}$ representing the input $Q1_{K1}$ is calculated as outlined above.

Join Compatibility: Given an entity set $\{e_1, e_2, ..., e_n\}$, the user would most likely be interested in formulating query conditions that involve joining different entities based on the commonality of the attributes between the entities. Since, the exact query conditions at this stage cannot be determined, we believe that an entity set that allows the flexibility to join every entity to every other entity on some attribute is definitely desirable than a set that offers very less joins across entities. In addition, the flexibility to join any two entities on a larger

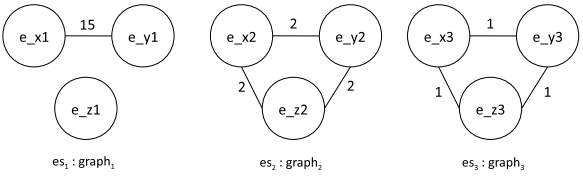

Fig. 4. Join compatibility for keyword combinations

number of attributes (instead of just one or two) is definitely desirable. Hence, for every entity-set, we define a ***join-compatibility-score***.

Consider the entity set $\{castle, train, city\}$ representing the input $Q1_{K1}$. It is possible to join the entities *castle* and *train* based on an attribute (e.g., location), the entities *train* and *city* based on an attribute (e.g., location) and the entities *castle* and *city* based on an attribute (e.g., name, location). On the other hand, an entity set $\{book, city\}$ representing the input $Q1_{K1}$ (considering the keywords 'castle' and 'train' refer to the name of a book i.e., its attribute value), will not allow any joins between the two entities and restrict a number of query conditions the user would like to formulate. Thus, it is clear that the first entity set should be ranked higher than the second.

Consider a list of entity sets $L = \{e_{s1}, e_{s2}, ..., e_{sm}\}$ where e_{si} represents a set of entities $\{e_1, e_2, ..., e_n\}$. We represent each entity set e_{si} by a graph G_i where the entities in the set represent the vertices of the graph. If any two entities in the set can be joined on an attribute, then an edge exists between the corresponding two vertices (representing the entities). The edge weight is represented by the total number of distinct attributes that can be used for the joining the two entities. For instance, consider the graphs representing the three entity sets shown in Figure 4. For the first graph, the vertices e_{x1}, e_{y1}, e_{z1} correspond to the entities in the entity set e_{s1}. The edge and the weight (15) between e_{x1} and e_{y1} indicates that the two entities can be joined on 15 different attributes.

Considering 'n' to be the maximum number vertices in a graph, the maximum number of edges is given as $m = n(n-1)/2$. We then sort the graphs (corresponding to the entity sets) based on the decreasing order of the number of edges. If two graphs have the same number of graphs, the tie is broken by running a maximum spanning tree (MST) that ranks the graphs based on the edge weights. From the above ordering of graphs, we achieve a final *join compatibility score* for every entity set in L by normalizing the rank of its graph (similar to method in Equation 1).

Putting it all Together: We have discussed three distinct components that play an important role in the ranking of entity sets. Since they are derived from linguistics, statistics, and join possibilities, we believe that when combined together, they form a comprehensive and suitable model to order the set of keyword combinations. We propose *logistic regression* to model our ranking function since it is a well-established and proven model used for predicting the outcome of an event when the parameters influencing the event are diverse and unrelated to

each other (as is the case with our components). Initially, we set uniform regression parameters i.e., equal weight to every parameter (linguistics, statistics and join compatibility) to rank the entity sets. As more and more workload is collected, we believe it will act as a major component in generating a significant training data to learn the parameter weights.

4.4 Query Completion

Based on the previous phases, our approach generates a template with a partially-filled query. Each keyword in the input is accounted for. For an entity, the template is populated by its corresponding domain and the underlying Web source to which it belongs. For instance, for $Q1_{K1} : Intent01$ selected by the user where *castle* represents a *tourist attraction*, *train* represents the *transportation mode*, and *London* is identified as a *city*. For this intent, the domains (tourist attractions, transportation) and sources (www.castles.org) are populated.

For an attribute, if the attribute has not been listed in the query template in the first step, it is analyzed for its type (spatial, temporal, generic) and the entity associated with it is obtained to formulate query conditions of the type: *entity.attribute* {*operator*} {*value*}. If this attribute can be formulated as an integration condition, then the corresponding conditions are formulated. Similarly, if the attribute is a popular choice for the output, then the `SELECT` clause is populated with it. For a value (e.g., London), the corresponding attribute and its parent entity are derived and condition of the type *city.name* == *London* is formed. For operators, if they are not listed in the template in the above steps, the possible conditions between the entities for which the operators are applicable are analyzed and modified accordingly. For instance, if an *operator* "near" is specified for the above intent, then the integration condition can be modified as: *castle.location near* {*train.startLocation*, *train.endLocation*, *city.Location*}. As the last stage of user interaction, the template is filled/modified by the user based on his/her preferences and the complete query (similar to $Complete_Q\ 1$) is formulated that captures the exact user intent in terms of constraints and conditions across multiple domains.

5 Conclusion

In this paper, we have presented a novel approach to query specification it's usefulness for web users who are familiar with keyword search. This approach needs further attention. We have also demonstrated how this approach can be carried out with the help of a knowledge base consisting of a knowledge repository and a workload repository. We have detailed the steps involved in the generation of a query skeleton, its ranking, and how a complete query can be generated with meaningful user interaction. The project on information integration, InfoMosaic, currently underway is working on this as well as the subsequent phases of query transformation, optimization, and execution.

References

1. Hristidis, V., Papakonstantinou, Y.: DISCOVER: Keyword Search in Relational Databases. In: VLDB, pp. 670–681 (2002)
2. Nie, Z., Kambhampati, S., Hernandez, T.: BibFinder/StatMiner: Effectively Mining and Using Coverage and Overlap Statistics in Data Integration. In: VLDB, pp. 1097–1100 (2003)
3. Cohen, W.W.: A Demonstration of WHIRL. In: SIGIR (1999)
4. Telang, A., Chakravarthy, S., Huang, Y.: Information integration across heterogeneous sources: Where do we stand and how to proceed? In: International Conference on Management of Data (COMAD), pp. 186–197 (2008)
5. Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Optimization of multi-domain queries on the web. In: PVLDB, vol. 1(1), pp. 562–573 (2008)
6. Zloof, M.M.: Query-by-example: A data base language. IBM Systems Journal 16(4), 324–343 (1977)
7. Petropoulos, M., Deutsch, A., Papakonstantinou, Y.: Clide: Interactive query formulation for service oriented architectures. In: SIGMOD Conference, pp. 1119–1121 (2007)
8. zu Eissen, S.M., Stein, B.: Analysis of Clustering Algorithms for Web-Based Search. In: PAKM, pp. 168–178 (2002)
9. Katz, B., Lin, J.J., Quan, D.: Natural Language Annotations for the Semantic Web. In: CoopIS/DOA/ODBASE, pp. 1317–1331 (2002)
10. Madhavan, J., Cohen, S., Dong, X.L., Halevy, A.Y., Jeffery, S.R., Ko, D., Yu, C.: Web-Scale Data Integration: You can afford to Pay as You Go. In: CIDR, pp. 342–350 (2007)
11. Ley, M.: Faceted DBLP. dblp.l3s.de/ (2006)
12. Miller, G.A.: WordNet: A Lexical Database for English. Commun. ACM 38(11), 39–41 (1995)
13. Allen, J.F.: Maintaining knowledge about temporal intervals. ACM Communications 26(11), 832–843 (1983), http://dx.doi.org/10.1145/182.358434
14. Fonseca, F., Egenhofer, M., Agouris, P., Camara, G.: Using ontologies for integrated geographic information systems. Transactions in Geographic Information Systems 3 (2002)
15. Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: Scalable Semantic Web Data Management Using Vertical Partitioning. In: International Conference on Very Large Databases (VLDB), pp. 411–422 (2007)
16. Waldinger, R., Appelt, D.E., Fry, J., Israel, D.J., Jarvis, P., Martin, D., Riehemann, S., Stickel, M.E., Tyson, M., Hobbs, J., Dungan, J.L.: Deductive question answering from multiple resources. In: New Directions in Question Answering. AAAI, Menlo Park (2004)