CourseCloud: Summarizing and Refining Keyword Searches over Structured Data

Georgia Koutrika Zahra Mohammadi Zadeh Hector Garcia-Molina
Computer Science Department, Stanford University
353 Serra Mall, Stanford, CA 94305, USA
{koutrika, zahram}@stanford.edu
hector@cs.stanford.edu

ABSTRACT

In this demo, we show data clouds that summarize the results of keyword searches over structured data. Data clouds provide insight into the database contents, hints for query modification and refinement and can lead to serendipitous discoveries of diverse results. In this demo paper:

- we summarize the main issues for generating data clouds
- we give an overview of our framework for keyword searching with summaries (clouds)
- we describe our system CourseCloud that allows searching for courses and their evaluation.

1. MOTIVATION AND OUTLINE

Our work on summarizing keyword search results using data clouds has been implemented as part of *CourseRank*, a social tool we have developed in InfoLab at Stanford. CourseRank displays official university information and statistics, such as bulletin course descriptions, grade distributions, and results of official course evaluations, as well as unofficial information, such as user ratings, comments, questions and answers. Students can search for classes, give comments and ratings, and organize their classes into a quarterly schedule or devise a four year plan. CourseRank maintains a relational database that stores information about courses, instructors, books, student comments, and so forth. (Figure 2 provides a small, simplified snapshot of the database schema.) A little over a year after its launch, the system is already used by more than 9,000 Stanford students, out of a total of about 14,000 students. The vast majority of CourseRank users are undergraduates, and there are only about 6,500 undergraduates at Stanford.

Students in the university are offered a wide variety of learning opportunities. They can choose among courses required for their degree (e.g., a course on advanced programming), courses outside their degree they can take for credit (e.g., a dance class), seminars, and so forth. In order to facilitate course planning, CourseRank offers two standard interfaces: one for browsing courses based on department and a keyword-based search interface. Keywords are searched in the title and description of courses.

When browsing courses based on department, students have to sift through long lists of courses and read their descriptions in order to find out the topics covered and identify useful courses depending on their needs and preferences. Many courses may cover common topics and different departments may offer courses on similar topics making locating, in the first place, and then sorting out the available options very tedious. Often, students rely on word of mouth to make their course decisions.

Keyword searching offers flexibility and freedom because users can form queries without any knowledge of the underlying database schema or a structured query language but it has limitations as well. An inherent problem of keyword searching is that the user still needs to guess what are the right keywords to describe her information need and how to refine or modify a search depending on the contents of the database in order to get the results that would satisfy her need. Furthermore, following current trends in database keyword searching that think of keyword search results in terms of database tuples (e.g., [3, 4, 5]), standard keyword search in CourseRank returns tuples of the Courses table that contain the search keywords. Nevertheless, people naturally think in terms of entities or objects, not tuples. Hence, when searching for "`Java`"-related courses, students may be interested not only in courses that explicitly mention this word in their title or description but also in courses with implicit references to "`Java`", such as in their comments. Such courses are missed by the system.

CourseCloud has been implemented as part of CourseRank and has many novel characteristics that set it apart from traditional searching and browsing systems. CourseCloud allows thinking of searches more naturally, i.e., in terms of looking for "search entities" rather than tuples. Hence, it enables searching for courses using keywords that can be found in different parts of a course (e.g., title, description, comments, etc). These pieces of information may be physically stored in different relations in the underlying database but the system hides these details from the users.

Furthermore, we couple the flexibility of keyword searches over structured data with the summarization and visualization capabilities of tag clouds to help users search a database. Tag clouds have been traditionally used for navigation and visualization purposes over unstructured data (e.g., [1]). Tags are the most important or popular terms in the underlying content and they are typically listed alphabetically and in

EDBT 2009, March 24–26, 2009, Saint Petersburg, Russia.

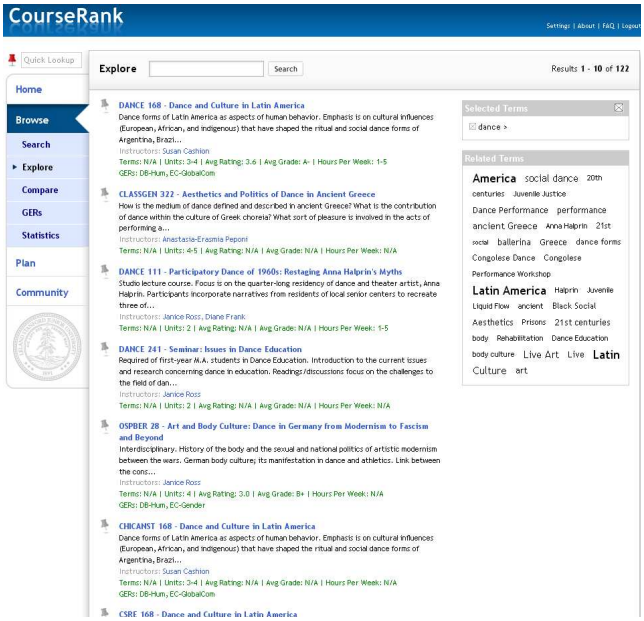

Figure 1: Searching for "dance".

Departments(DepID, DepName)
Courses(CourseID, InstrID, DepID, Title, Description, Units)
Instructors(InstrID, InstrName)
Comments(SuID, CourseID, Year, Term, Text, Rating, Date)

Figure 2: An example database for courses.

different color or font size based on their importance [2]. CourseCloud generates a tag cloud to summarize the results of a keyword search over the course database. In this case, the cloud contains the most significant or representative terms (concepts) found within these results, and we call it *data cloud* [7]. The terms are aggregated over all parts that make a course entity, such as the title, the comments, etc, and may be stored in different tables in the database.

A data cloud can guide users through search results and for search refinement. Terms in the data cloud (as in traditional tag clouds) are hyperlinks. The searcher can click on a cloud term to refine search results. The cloud is updated accordingly to reflect the new, refined, results. Different users may choose different terms from a data cloud refining their searches in diverse ways.

Example. A student interested in taking a dance class can type the keyword "`dance`" and get a list of matching courses along with a cloud summarizing course information in this list, as shown in Figure 1. The keyword "`dance`" is searched in different fields and relations in the database that contain information related to courses. For example, if there are comments that mention "`dance`", the respective courses will appear (in some position) in the results. The cloud provides many concepts related to "`dance`" that are found in the matching courses, such as "`performance`", "`social dance`", and "`Latin`". These words may be found in different parts of the database related to the current search. For example, the term "`performance`" is found in many user comments that refer to "`dance`" courses with live performances.

The data cloud conveniently categorizes courses in a digestible way under different concepts. In this way, the student can find out that there are courses offered not only by the DANCE program (identified by the course code in the results) but also from other programs that study dance from different aspects, such as the DRAMA or HUMANITIES programs, and can get an overall picture for such courses irrespective of their program or department. In addition, the data cloud can identify interesting concepts that the student did not know beforehand. For example, she might not know that there were courses related to "`Latin America dance`". The data cloud can help reveal unexpected or unknown connections and refine searches in serendipitous ways. Figure 3 shows the search result page when refining the results of "`dance`" into "`Latin America`". The cloud now shows terms that occur in the refined results. The matching courses are 13 out of the 122 "`dance`"-related courses that the initial search returned.

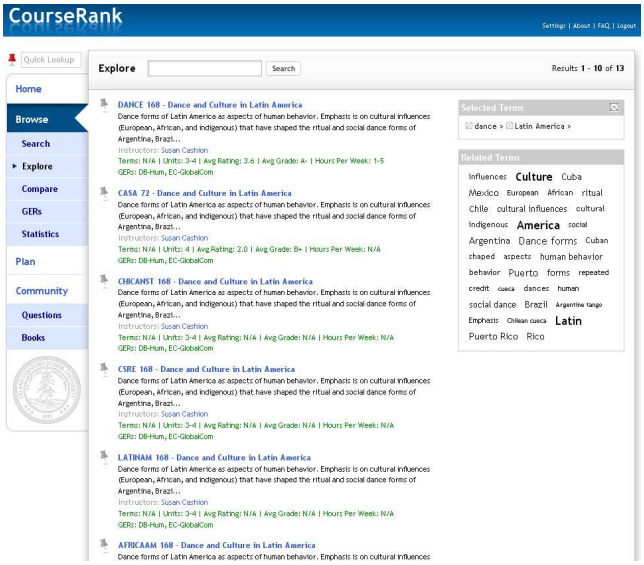

Figure 3: Refining "dance" courses.

As another example to highlight the diversity of terms that arise in clouds, consider the words found in the cloud of a search for "`Greek`"-related courses depicted in Figure 4. Users are given several, diverse, options to refine their searches, such as "`Greek tragedy`", "`Greek vases`" or "`Plato`". Different students may chose different paths and end up in different course selections. The data cloud may reveal hidden relationships, such as the connection between "`Greek`" and "`science`" that a user might not have thought of.

There are a number of interesting questions to tackle in data clouds, such as what is a good word in the results for including in the data cloud, how to support keyword searching for complex objects that span multiple tables, and so forth. The data cloud framework and methods are described in detail in [7]. In the following sections, we briefly touch some issues and give an overview of our current approach and the CourseCloud system.

2. DATA CLOUDS FOR KEYWORD SEARCH

Keyword Search. In information retrieval, the information unit that is searched is well-defined: documents are the units returned for keyword searches. In databases, however, information that "conceptually" refers to a single entity

Greek			
Plato	classical	Greek culture	ancient
literature	philosophy	Greek language	modern
Ideals	Greece	Heritage Language	art
Heraclitus	4th-Century	Modern Greek	culture
Language Learners	Roman	prose composition	Latin
Latin literature	invention	Archaic Greek	Lyric
science	Survey	Advanced Greek	Greek art
tragedy	film	Beginning Modern	heritage
mathematics	Greek tragedy	Beginning Greek	language
Intermediate Modern	Archaic	Augustine	
Greek vases	Beazley	4th-Century Greek	

Figure 4: Words in an example data cloud "Greek".

(unit) may be found in different relations due to database structuring and normalization. For example, in CourseRank, courses can be thought of as having several pieces of information associated with them, such as title, description, comments, and ratings, but these are stored in different tables. Hence, when searching for courses, the system may need to look for the query terms into several places that may be connected to courses.

We model a database D as a collection V of *search entities*. A search entity v is conceptually a complex object with attributes B_1, ... B_n. An attribute B_i can map to a column in the underlying database (e.g., the course title) or to an object or list of objects that essentially group information into one attribute for the search entity v (e.g., the ratings given to a course). The collection V can be thought of as a "view" that collects and groups together information related to an individual entity from the stored relations in D and represent it as a single unit of information.

For example, we can think of a "course entity" as a complex object shown in Figure 5, which has attributes coming from different relations in the database. This course entity reads the title and description of the course from the corresponding tuple in the Courses table, the instructor name by joining the Courses table with the Instructor table on the instructor id, and so forth.

A keyword query q is formulated as a conjunction of keyword terms. A term k may be a single word, e.g., "`dance`", or a phrase, e.g., "`database systems`". Given a query q and a collection V defined over the database D, the answer for q is the set $V_q \subseteq V$, that contains all search entities from V that have all keywords of the query q at least once.

For example, consider the database instance shown in Figure 5 and the query "`Java`". While following a traditional approach to keyword search for databases in CourseRank, we could only locate the course with code C245 that mentions Java both in its title and its description, considering search entities allows go deeper in the database and finding, in addition, C145 because its comments talk about Java.

There are many questions to tackle. How deep in the database should we go when searching for "search entities" that contain the query keywords? For example, when searching for "courses", should we also look into the instructors and books relations? We can have a domain expert specify the default, as in the current CourseCloud system, or automatically determine the "search depth" in the spirit of [6].

A very important issue is how we rank search entities that match a keyword search. Existing approaches to keyword search over databases rank tuples (or sets of joining tuples) that match a keyword search. Thinking of search entities as the equivalent of "documents", we can make use of IR-standard ranking methods. For instance, given the set V_q

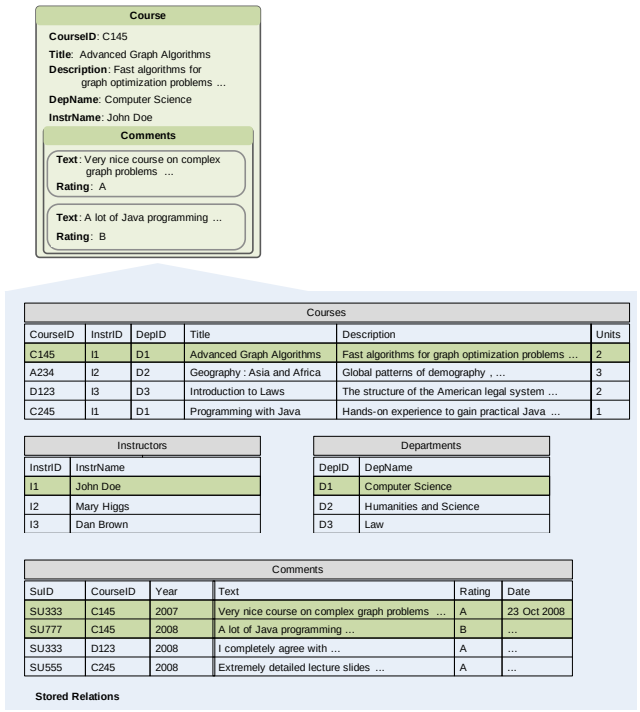

Figure 5: A search entity.

of results for a query q, we can compute the tf*idf weight of any query term k in any entity v in V_q. Then, we can add up the tf*idf weights of all query terms found in v to compute a score for v w.r.t. query q.

$$score(v, q) = \sum_{k \in q} tf_{k,v} * idf_k \tag{1}$$

One issue with this approach is that it does not take into account the position of a query term. For example, when searching for "`Java`", a course that contains "`Java`" in its title should probably be given a higher score than a course that mentions the same word in its comments. For this purpose, we use attribute weights [7]. An attribute weight depicts the significance of a term's occurrence depending on the attribute where the term is found. Then, the formula for computing the $tf_{k,v}$ is refined as follows:

$$tf_{k,v} = \frac{\sum_{B \text{ of } v} w_B * n_B}{n_v} \tag{2}$$

where w_B is the weight of attribute B. We can manually assign weights to attributes in a database. For example, in CourseCloud, we currently give higher weights to attributes, such as the title of a course, and less weight to attributes, such as the text of a comment.

Data clouds. Generating a data cloud for summarizing the results of a keyword query over structured data has many challenges compared to generating tag clouds for more traditional purposes, such as for showing user tags in a social bookmaking system. One challenge is finding "good" words to include in the data cloud.

Consider, for example, a query about "`photography`" and assume that everything in our database is about digital photography. A typical tag cloud shows the most popular, i.e.,

frequent, terms. Following the same approach, we could score words in the results based on the number of their occurrences in the results. Then, the word "`digital`" would surface in the data cloud. However, "`digital`" is not a good term for the cloud: it may be very popular but it is not useful for refining the search results of "`photography`".

Our approach to selecting cloud keywords for a search q is to treat each candidate term x in the search results as a one word query and compute the similarity between that term and each matching entity v in V_q. For this purpose, we can compute the tf*idf weight of term x in v. Then, we sum up over all v in the results V_q for the query in order to find how significant x is for the results of q.

$$score(x, q, V_q) = \sum_{v \in V_q} tf_{x,v} * idf_x \quad (3)$$

When computing the tf values in the formula above, we can consider the term positions in the database using attribute weights, as we discussed above for ranking search entities.

3. SYSTEM OVERVIEW

CourseCloud's system architecture is depicted in Figure 6. In off-line mode, we need to deal with many issues, such as deciding how to tokenize text fields, how to combine the same words found in different fields, what structures and statistics are required in order to support searching with dynamic summaries, and so forth. The *Tokenizer* reads the relations and the fields in the database that should be searchable w.r.t. selecting courses and stores n-grams, with $n \leq 2$. It removes common parts of speech, such as personal pronouns (e.g., "I", "he") and prepositions (e.g., "on", "during"). It also cleans the words found in the database to remove words found in comments that may spam the clouds. The result of this preprocessing is a tuple-based inverted index that stores term occurrences in the database tuples.

We search at the level of search entities. Search entities serve as a useful abstraction but in practice we do not want to materialize them over the physical database. On the other hand, using directly the tuple-based inverted index at query time to generate the set of entities that match a query is not straightforward and is time-consuming. Each occurrence of a query term in the index must be linked to the search entity it conceptually belongs to. For example, if the word "`dance`" is found in a comment in the course database of Figure 2, then the system needs to find which course the comment is attached to. This lookup must be done for all tuples in the index where the query keywords are located. Instead, we store the information of which entity each term belongs in advance in an entity-based inverted index.

An *entity-based inverted index* stores word occurrences per search entity and is used to speed up query processing times. The *Entity-tier* creates an entity-based inverted index based on the tuple-based inverted index and with the help of a set of parameterized queries that attach entity ids to each term recorded. An entity id maps to a primary key in the actual database. The *Statistician* computes additional statistics, such the *idf* weight per word, and stores them in the entity-based inverted index and in auxiliary tables. Finally, it generates all required database indexes to speed up searches at query time.

The *Search&Summarize* is the online component that implements the searching and summarizing algorithms and supports different ranking methods for entities and terms. It completely relies on the entity-based inverted index and the set of auxiliary statistics.

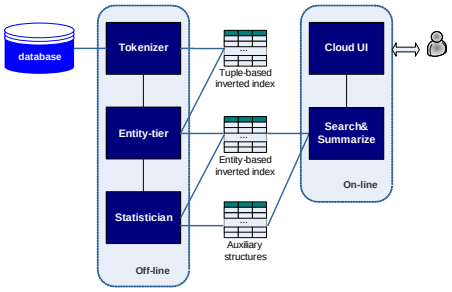

Figure 6: CourseCloud system architecture

4. DEMONSTRATION

In the demo, users can play a student role and search for courses offered in Stanford using the CourseCloud. They will be able to refine their searches by adding or removing terms in a search and navigate using the data clouds. Different search scenarios will be demonstrated, such as finding a course for English, learning about civil rights or taking a course that would help control weight, and we will show how we can find diverse and serendipitous results for different types of users. In addition, two more traditional interfaces will be offered, one based on standard keyword search (using no clouds) and a catalog browsing interface. In this way, users will be able to compare different options live and understand the circumstances under which each interface may work better.

5. REFERENCES

[1] Flickr: url: http://www.flickr.com/.

[2] Wikipedia: http://en.wikipedia.org/wiki/tag_cloud.

[3] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE*, pages 5–16, 2002.

[4] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.

[5] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, pages 431–440, 2002.

[6] G. Koutrika, A. Simitsis, and Y. Ioannidis. Précis: The essence of a query answer. In *ICDE*, pages 69–78, 2006.

[7] G. Koutrika, Z. Mohammadi Zadeh, and H. Garcia-Molina. Data Clouds: Summarizing keyword search results over structured data. In *EDBT*, 2008.