Data Clouds: Summarizing Keyword Search Results 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

Keyword searches are attractive because they facilitate users searching structured databases. On the other hand, tag clouds are popular for navigation and visualization purposes over unstructured data because they can highlight the most significant concepts and hidden relationships in the underlying content dynamically. In this paper, we propose coupling the flexibility of keyword searches over structured data with the summarization and navigation capabilities of tag clouds to help users access a database. We propose using clouds over structured data (*data clouds*) to summarize the results of keyword searches over structured data and to guide users to refine their searches. The cloud presents the most significant words associated with the search results. Our keyword search model allows searching for entities than can span multiple tables in the database rather than just tuples, as existing keyword searches over databases do. We present several methods to compute the scores both for the entities and for the terms in the search results. We describe algorithms for keyword searches with data clouds and we present our system, *CourseCloud*, that offers a unified search and browse interface to a course database. We present experimental results showing (a) the appropriateness of the methods used for scoring terms, (b) the performance of the proposed algorithms, and (c) the effectiveness of CourseCloud compared to typical search and browse interfaces to a course database.

1. INTRODUCTION

Imagine accessing the contents of an e-museum or a digital collection, such as the Addison Gallery of American Art [24], to learn about art and artifacts or accessing the electronic course catalog of a university to explore learning opportunities. The typical options to access such databases are browsing, e.g., based on the available collections or departments, or searching, e.g., based on title, artist, semester, and so forth. Browsing interfaces offer a structured way to guide people but they do not naturally lend themselves to serendipitous or diverse explorations. Search interfaces leave the burden to the user to think of ways to explore a database. As a result, exploring such databases can be overwhelming for many reasons: their size, the number and diversity of choices, the lack of intuitive interfaces and the lack of knowledge or experience from the user side of how or where to find interesting information in a particular database.

Keyword searches have recently attracted attention because they facilitate users searching structured databases. On the other hand, tag clouds are very popular for navigation and visualization purposes over unstructured data. Their main advantage lies in their ability to highlight the most significant concepts and hidden relationships in the underlying content dynamically [14]. Since human beings tend to think in concepts and models, it's easier to get an idea of presented content if the main concepts are in digestible pieces and prioritized by their significance.

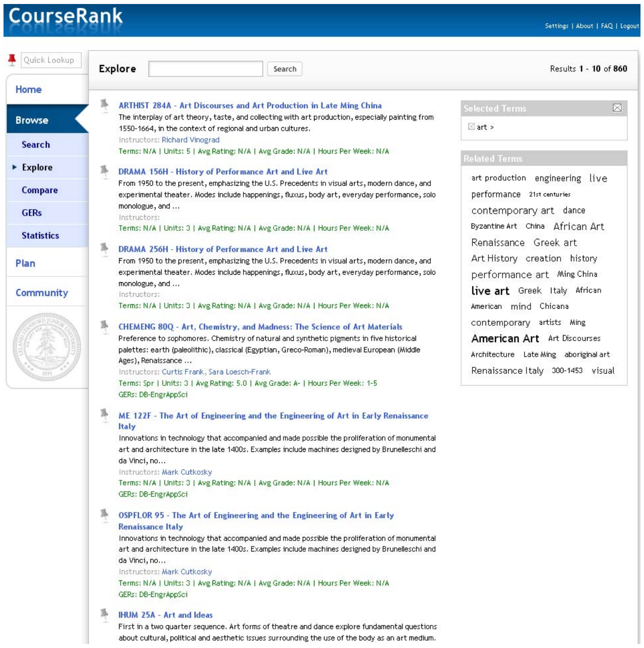
In this paper, we propose coupling the flexibility of keyword searches over structured data with the summarization and navigation capabilities of tag clouds to help users search a database. We describe *data clouds* that *summarize the results of keyword searches over structured data and guide users to refine their searches.* Data clouds provide insight into the database contents, hints for query refinement and can lead to serendipitous discoveries of diverse results.

1.1 Motivation and Outline of Work

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. 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. CourseRank maintains a relational database that stores information about courses, instructors, books, student comments, and so forth. (Figure 2 provides a small, simplified snap-

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



Results **1** - **10** of **860**

ARTHIST 284A - Art Discourses and Art Production in Late Ming China
The interplay of art theory, taste, and collecting with art production, especially painting from 1550-1664, in the context of regional and urban cultures.
Instructors: Richard Vinograd
Terms: N/A | Units: 5 | Avg Rating: N/A | Avg Grade: N/A | Hours Per Week: N/A

DRAMA 156H - History of Performance Art and Live Art
From 1950 to the present, emphasizing the U.S. Precedents in visual arts, modern dance, and experimental theater. Modes include happenings, fluxus, body art, everyday performance, solo monologue, and ...
Instructors:
Terms: N/A | Units: 3 | Avg Rating: N/A | Avg Grade: N/A | Hours Per Week: N/A

DRAMA 256H - History of Performance Art and Live Art
From 1950 to the present, emphasizing the U.S. Precedents in visual arts, modern dance, and experimental theater. Modes include happenings, fluxus, body art, everyday performance, solo monologue, and ...
Instructors:
Terms: N/A | Units: 3 | Avg Rating: N/A | Avg Grade: N/A | Hours Per Week: N/A

CHEMENG 80Q - Art, Chemistry, and Madness: The Science of Art Materials
Preference to sophomores. Chemistry of natural and synthetic pigments in five historical palettes: earth (paleolithic), classical (Egyptian, Greco-Roman), medieval European (Middle Ages), Renaissance ...
Instructors: Curtis Frank, Sara Loesch-Frank
Terms: Spr | Units: 3 | Avg Rating: 5.0 | Avg Grade: A- | Hours Per Week: 1-5
GERs: DB-EngrAppSci

ME 122F - The Art of Engineering and the Engineering of Art in Early Renaissance Italy
Innovations in technology that accompanied and made possible the proliferation of monumental art and architecture in the late 1400s. Examples include machines designed by Brunelleschi and da Vinci, no...
Instructors: Mark Cutkosky
Terms: N/A | Units: 3 | Avg Rating: N/A | Avg Grade: N/A | Hours Per Week: N/A
GERs: DB-EngrAppSci

OSPFLOR 95 - The Art of Engineering and the Engineering of Art in Early Renaissance Italy
Innovations in technology that accompanied and made possible the proliferation of monumental art and architecture in the late 1400s. Examples include machines designed by Brunelleschi and da Vinci, no...
Instructors: Mark Cutkosky
Terms: N/A | Units: 3 | Avg Rating: N/A | Avg Grade: N/A | Hours Per Week: N/A
GERs: DB-EngrAppSci

IHUM 25A - Art and Ideas
First in a two quarter sequence. Art forms of theatre and dance explore fundamental questions about cultural, political and aesthetic issues surrounding the use of the body as an art medium.

Selected Terms
☒ art >

Related Terms
art production engineering live performance 21st centuries contemporary art dance Byzantine Art China African Art Renaissance Greek art Art History creation history performance art Ming China live art Greek Italy African American mind Chicana contemporary artists Ming American Art Art Discourses Architecture Late Ming aboriginal art Renaissance Italy 300-1453 visual

Figure 1: Searching for "art".

shot of the database schema.) 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 the 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. However, they still need to figure out the right search keywords depending on the contents of the database in order to get the results that would satisfy their information need. Furthermore, current trends in database keyword search think of keyword search results in terms of database tuples (e.g., [7, 9, 11]), whereas people naturally think in terms of entities or objects, not tuples. Hence, when searching for "`Java`"-related courses in CourseRank, students may be interested not only in courses that explicitly mention this word in their title or description (i.e., in an attribute of the Courses relation) but also in courses with implicit references to "`Java`", such as in their comments. Standard keyword search in CourseRank would return only tuples of the Courses table that contain the keywords.

With all the issues mentioned above in mind, we have implemented *CourseCloud* as part of CourseRank to help students make informed and personalized choices about classes. CourseCloud's novel characteristics that set it apart from traditional keyword search and browsing systems include:

- *Search for "objects" that span multiple tables.* CourseCloud allows thinking of searches more naturally, i.e., in terms of looking for "search entities" rather than tuples.

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

Figure 2: An example database for courses.

Hence, it enables searching 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.

- *Use of data clouds for summarizing search results over structured data.* Tag clouds have been traditionally used for navigation and visualization purposes over unstructured data. In CourseCloud, 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. CourseCloud generates a data cloud to summarize the results of a keyword search over structured data. The data cloud contains the most significant or representative terms (concepts) found within these results. The terms are aggregated over all parts that make a course entity, such as the title, the comments, etc, and can be stored in different tables in the database.
- *Use of data clouds for navigation and search refinement.* Terms in the data cloud (as in traditional tag clouds) are hyperlinks. The searcher can click on a term from the data cloud to refine search results. The cloud is updated accordingly to reflect the new, refined, results. Different students may choose different terms from a data cloud refining their searches in diverse ways. Hence, data clouds provide the opportunity to gain insight into the database contents, can guide users through the results and can lead to serendipitous discoveries of diverse results. Overall, CourseCloud provides both browsing and searching in a unified way.

For instance, a student has to take a class on art based on her program requirements but she is not familiar with this field. She types the "`art`" and gets a list of matching courses along with a cloud summarizing course information in this list, as shown in Figure 1. The keyword "`art`" is searched in different fields and relations in the database that contain information related to courses. For example, if there are comments that mention "`art`", the respective courses will appear (in some position) in the results. The cloud provides many diverse concepts related to "`art`" that are found in the matching courses, such as "`performance`", "`art production`", and "`Renaissance`". 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 "`art`" 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 ARTHISTORY 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 "`Byzantine Art`". The data cloud can help reveal unexpected connections and refine searches in serendipitous ways. Figure 3 shows the search result page when refining the results of “art” into “Architecture”. The refined results are only 99 out of the initial 860 returned for the initial search and the cloud shows only terms that occur in these results.

There are many issues to tackle that do not exist neither in typical keyword searches nor in traditional tag clouds found in social sites. In contrast to the former, we allow users search at the level of entities and hide the schema of the database. A search entity may be defined over multiple tables and a query term can be found in any part of a search entity. It is impractical to materialize search entities over the physical database but we still need to be efficient.

In contrast to social sites, where the set of user tags to show in a tag cloud is given, we need to deal with issues, such as deciding how to tokenize text fields, how to aggregate the same words found in different fields, what structures and statistics are required in order to support searching with dynamic clouds, and so forth. Furthermore, in traditional tag clouds, tag “significance” is often understood as tag popularity and it is captured by the term frequency. Showing in the data cloud terms that are popular in the results of a search may not be very useful for refining search results. In addition, our search entities have structure. Should the position of a term affect its significance?

Finally, putting all pieces together, data clouds are dynamically computed over search results, which means that execution time is critical. At query time, we need to compute the results and the data cloud efficiently reducing the amount of additional processing required.

1.2 Contributions

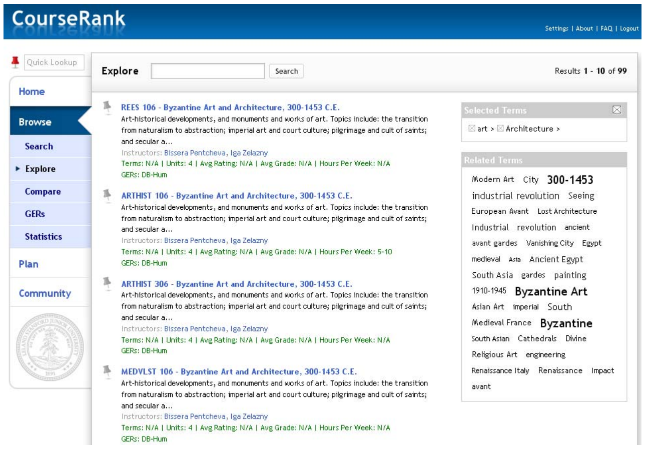
In summary, our contributions are the following:

- We consider that keyword searches return “search entities” (rather than tuples), which may span multiple database tables, and a keyword may be found in any part of a matching entity. Given a set of entities returned for a search, we rank the terms found in them and select the high-ranked ones to be presented as a summary over the results in the form of a data cloud.
- We describe algorithms that compute or refine the set of results for a search and a data cloud for this search. Our algorithms search deeper in the database for entities rather than in a limited number of attributes.
- We describe the *CourseCloud* system, which allows searching courses using keywords that can be found in different parts of a course (e.g., title, description, comments, etc).
- We evaluate the performance of the algorithms and we compare the different ranking methods for computing scores for words in the results. Finally, we present results of a user study comparing CourseCloud to the standard search and browse interfaces offered by CourseRank.

2. RELATED WORK

In this section, we present the state of the art for the main research areas related to our work.

Tag clouds and summaries of results. A tag cloud is a visual depiction of text content. Tags are words typically listed alphabetically and in different color or font size based on their importance [5]. Tag clouds have been attributed to Coupland [13] but have been popularized by the web site Flickr [2] launched in 2004. They have since appeared on numerous Web sites including Technorati [4], del.icio.us [1], and so forth. PubCloud uses tag clouds for the summarization of results from queries over the PubMed database of biomedical literature [22]. PubCloud responds to queries of this database with tag clouds generated from words extracted from the abstracts returned by the query. Tag cloud drawing has recently received attention [17, 20]. Generating summary keywords has been applied to emails [16]. This is different from our problem where we want to dynamically generate summary keywords for results that match a query. Recently, a number of tools that generate “word clouds” from text a user provides instead of tags have emerged. The word cloud gives greater prominence to words that appear more frequently in the source text. For example, in ManyEyes [3], a visualization tool for datasets, the user can choose to generate a cloud of frequent words from free text. The cloud will strip out punctuation, calculate the frequency of each word, and draw the word at a size that is based on its frequency. Wordle [6] is another tool for generating word clouds from user-provided text.


CourseRank
Settings | About | FAQ | Logout
Quick Lookup
Home
Browse
Search
Explore
Compare
GERs
Statistics
Plan
Community
Explore
Search
Selected Terms
☒ art > ☒ Architecture >
Related Terms
Modern Art City 300-1453
industrial revolution Seeing
European Avant Lost Architecture
Industrial revolution ancient
avant gardes Vanishing City Egypt
medieval Asia Ancient Egypt
South Asia gardes painting
1910-1945 Byzantine Art
Asian Art imperial South
Medieval France Byzantine
South Asian Cathedrals Divine
Religious Art engineering
Renaissance Italy Renaissance Impact
avant

Results **1** - **10** of **99**

REES 106 - Byzantine Art and Architecture, 300-1453 C.E.
Art-historical developments, and monuments and works of art. Topics include: the transition from naturalism to abstraction; imperial art and court culture; pilgrimage and cult of saints; and secular a...
Instructors: Bissera Pentcheva, Iga Zelazny
Terms: N/A | Units: 4 | Avg Rating: N/A | Avg Grade: N/A | Hours Per Week: N/A
GERs: DB-Hum

ARTHIST 106 - Byzantine Art and Architecture, 300-1453 C.E.
Art-historical developments, and monuments and works of art. Topics include: the transition from naturalism to abstraction; imperial art and court culture; pilgrimage and cult of saints; and secular a...
Instructors: Bissera Pentcheva, Iga Zelazny
Terms: N/A | Units: 4 | Avg Rating: N/A | Avg Grade: N/A | Hours Per Week: 5-10
GERs: DB-Hum

ARTHIST 306 - Byzantine Art and Architecture, 300-1453 C.E.
Art-historical developments, and monuments and works of art. Topics include: the transition from naturalism to abstraction; imperial art and court culture; pilgrimage and cult of saints; and secular a...
Instructors: Bissera Pentcheva, Iga Zelazny
Terms: N/A | Units: 4 | Avg Rating: N/A | Avg Grade: N/A | Hours Per Week: N/A
GERs: DB-Hum

MEDVLST 106 - Byzantine Art and Architecture, 300-1453 C.E.
Art-historical developments, and monuments and works of art. Topics include: the transition from naturalism to abstraction; imperial art and court culture; pilgrimage and cult of saints; and secular a...
Instructors: Bissera Pentcheva, Iga Zelazny
Terms: N/A | Units: 4 | Avg Rating: N/A | Avg Grade: N/A | Hours Per Week: N/A
GERs: DB-Hum

Figure 3: Refining “art” courses.

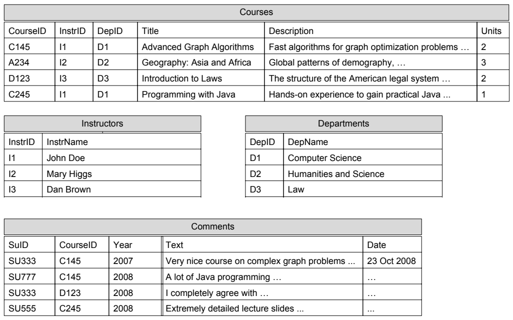
Keyword searches over databases. The need for keyword searching in databases is growing. Recent approaches include systems, such as BANKS [11], DISCOVER [19], DBXplorer [7], and ObjectRank [9]. Each of these approaches works on some kind of graph (e.g., data graph [11] or schema graph [7, 19]). Based on this graph, an answer to a given set of terms is interpreted as a sub-graph connecting the tuples that contain the query terms. Précis queries go one step further and expand the answer with information found around the initial subgraph that may be related to the query [21]. Faceted search allows navigating in a resource space through a pre-computed set of dimensions, which represent significant features of the resources (e.g., [10]).

In our work, we consider that keyword searches return “search entities”, which may span multiple database tables, and a keyword may be found in any part of a matching entity. Furthermore, we generate data clouds over the keyword search results showing significant terms in these results. Finally, we consider various methods to score the terms apart from using popularity, as in traditional tag cloud, taking into account the structure of search entities.

3. FRAMEWORK

3.1 Keyword Search

A database *D* comprises a set of stored relations. A stored



Courses

CourseID	InstrID	DepID	Title	Description	Units
C145	I1	D1	Advanced Graph Algorithms	Fast algorithms for graph optimization problems …	2
A234	I2	D2	Geography: Asia and Africa	Global patterns of demography, …	3
D123	I3	D3	Introduction to Laws	The structure of the American legal system …	2
C245	I1	D1	Programming with Java	Hands-on experience to gain practical Java ...	1

Instructors

InstrID	InstrName
I1	John Doe
I2	Mary Higgs
I3	Dan Brown

Departments

DepID	DepName
D1	Computer Science
D2	Humanities and Science
D3	Law

Comments

SuID	CourseID	Year	Text	Date
SU333	C145	2007	Very nice course on complex graph problems ...	23 Oct 2008
SU777	C145	2008	A lot of Java programming …	…
SU333	D123	2008	I completely agree with …	…
SU555	C245	2008	Extremely detailed lecture slides ...	...

Figure 4: A database instance with tuples

relation R (denoted R_i when more than one relation is implied) has a set $\mathbf{A}$ of columns. We will use $R.A_j$ to refer to a column in $\mathbf{A}$ or simply A_j when R is understood. A tuple in D is denoted t. Furthermore, we consider the tuple graph T_D for the database D where nodes map to the tuples in the database and for each pair of adjacent tuples t_i, t_j on the graph, where $t_i \in R_i$, $t_j \in R_j$ and there is a primary-to-foreign key relationship between R_i and R_j, there is an edge between them if $(t_i \bowtie t_j) \in (R_i \bowtie R_j)$. Figure 4 shows a database instance and Figure 5 shows an example tuple graph with four tuples.

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 be atomic, mapping to a column in the underlying database (e.g., the course title), or composite mapping to an object or list of objects that essentially group information into one attribute for the search entity v (e.g., the set of ratings given to a course could be conceptually thought as an attribute of the course entity.) 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.

A search entity v is represented by its entity id B_0. The entity id typically maps to the primary key A of a relation R_0. For this reason, R_0 is called the *primary entity relation.* The primary relation may also provide additional attributes for the entity other than its id. Other relations in the database that directly or indirectly join to R_0 can be used to provide additional information for v, and are called *secondary entity relations.*

For example, we can think of a "course entity" as the complex object shown in Figure 6, which has attributes coming from different relations in the database. The primary relation is Courses which provides the course id (i.e., the entity id). The course entity gets additionally the title and description of the course from the corresponding tuple in the Courses table. Three other relations play the role of secondary relations augmenting the information of the course entity. The course entity gets the instructor name by joining the Courses table with the Instructors table on the instructor id, the text of the comments by joining the tables Courses and Comments on the course id and so forth.

There may be other tables in the database that do not supply any information for an entity. The primary and secondary relations are predetermined by a domain expert or the application designer, as in the current CourseCloud system. Alternatively, a domain expert can define only the primary entity relation and let the system automatically determine which parts of the database should be searched in the spirit of [21]. This direction is left for future work. A single database may contain search entities of different types (e.g., course entities, instructor entities, etc). Hence, we can imagine a search interface that allows searching for different kinds of entities. In the rest of the paper, for simplicity in the presentation and without loss of generality, we consider that we have a single collection of search entities.

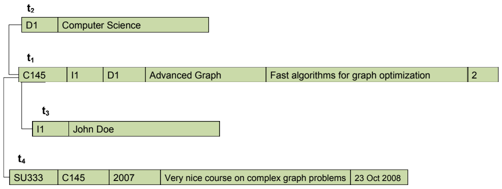

t_2
D1 | Computer Science
t_1
C145 | I1 | D1 | Advanced Graph | Fast algorithms for graph optimization | 2
t_3
I1 | John Doe
t_4
SU333 | C145 | 2007 | Very nice course on complex graph problems | 23 Oct 2008

Figure 5: A small tuple graph

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 term k and a search entity v, which takes its entity id from the stored tuple t in the primary relation R_0, v *contains the query term* k, if one of the following hold:

(a) t contains the term k in one of its attribute values, or

(b) there is a tuple t_i in a relation R_i that contains k and a path on the tuple graph T_D that connects t and t_i.

Based on the above definition, 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 includes all search entities from V that *contain all query terms* in the query q at least once, i.e.:

$$V_q \ := \ \{v | v \in V \land \forall \ k \in q, v \text{ contains the query term } k\}$$

For example, consider Figure 6 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 in its title and its description, considering search entities allows to go deeper in the database and find, in addition, `C145`, whose comments talk about Java.

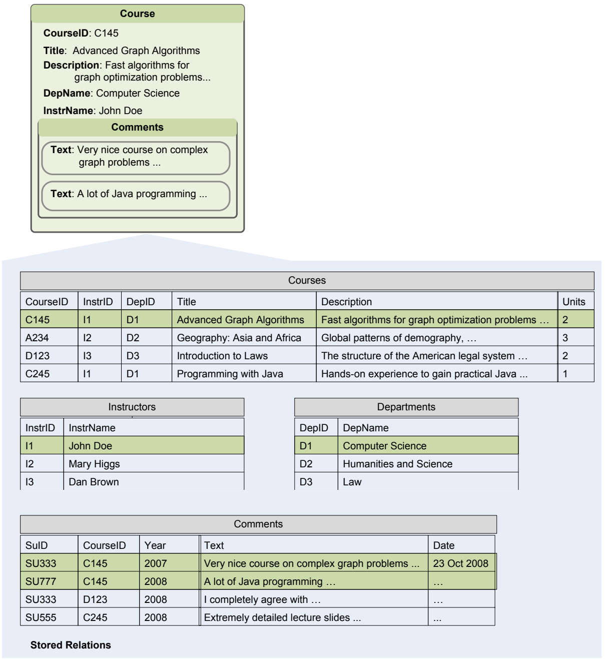
Ranking search entities. An important issue is how we rank search entities that match a keyword search. We consider the set V_q of search entities that match a query q. Existing approaches to keyword search over databases rank tuples (or joining trees of tuples) that match a keyword search. Thinking of search entities as the equivalent of "documents", we can make use of IR-standard ranking methods[1]. For instance, we can compute the tf*idf weight of any query term k in any entity v in V_q. The term frequency tf can be computed using this formula:

$$tf_{k,v} = \frac{\sum\limits_{B \text{ of } v} n_B}{n_v} \tag{1}$$

where n_B is the number of occurrences of k in an attribute B of v and n_v is the number of terms in v. The inverse document frequency idf for k is:

$$idf_k = ln(\frac{N}{N_k}) \tag{2}$$

[1] IR-ranking methods have been used for ranking joining trees of tuples that match a keyword search (e.g., [18]). A joining tree of tuples is more restricted than search entities, which can contain query terms in any of their parts.



Course

CourseID: C145

Title: Advanced Graph Algorithms

Description: Fast algorithms for graph optimization problems...

DepName: Computer Science

InstrName: John Doe

Comments

Text: Very nice course on complex graph problems ...

Text: A lot of Java programming ...

Courses

CourseID	InstrID	DepID	Title	Description	Units
C145	I1	D1	Advanced Graph Algorithms	Fast algorithms for graph optimization problems …	2
A234	I2	D2	Geography: Asia and Africa	Global patterns of demography, …	3
D123	I3	D3	Introduction to Laws	The structure of the American legal system …	2
C245	I1	D1	Programming with Java	Hands-on experience to gain practical Java ...	1

Instructors

InstrID	InstrName
I1	John Doe
I2	Mary Higgs
I3	Dan Brown

Departments

DepID	DepName
D1	Computer Science
D2	Humanities and Science
D3	Law

Comments

SuID	CourseID	Year	Text	Date
SU333	C145	2007	Very nice course on complex graph problems ...	23 Oct 2008
SU777	C145	2008	A lot of Java programming …	…
SU333	D123	2008	I completely agree with …	…
SU555	C245	2008	Extremely detailed lecture slides ...	...

Figure 6: A search entity.

where N is the number of search entities in the database and N_k is the number of entities containing k. Then, we can add up the tf*idf weights of all query terms in v to compute a score for v w.r.t. query q.

$$score(v, q) = \sum_{k \in q} tf_{k,v} * idf_k \tag{3}$$

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`", should a course that contains "`Java`" in its title be given the same score with a course that mentions the same word in its comments? One approach to tackle this is to use position weights as in the case of HTML pages [12]. A position weight depicts the significance of a term's occurrence depending on its position in a document. We can transfer this idea to search entities and refine Formula 1 using attribute weights:

$$tf_{k,v} = \frac{\sum_{B \text{ of } v} w_B * n_B}{n_v} \tag{4}$$

where w_B is a weight for attribute B. We can manually pre-assign weights to attributes in the 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. Attribute weights can be also determined automatically based on a set of rules. For example, single-value attributes, such as title and description may weigh more than lists of values, such as comment texts. Automatic techniques for scoring attributes in search results have been also proposed based on their usefulness or visibility [15, 23]. As part of our ongoing work, we are interested in exploring how to reflect the "search depth" in ranking, i.e., the length of the path that connects the entity id tuple in the primary relation with the relation where the query term was found in order to rank this search entity.

3.2 Data Clouds for Keyword Search

We consider a query q and a collection V defined over the database D. The answer for q is the set V_q. We want to compute a significance score for each term k contained in the search entities of V_q. The top terms can be then used as a summary S of the results and guide search refinement.

(**Popularity-based**) One approach is to compute a score based on the number of times k co-occurs with all the terms in q. This is given by the following formula, which sums up all k's occurrences in all the attributes that describe the entities found in the query results V_q for the query q:

$$score(k, q, V_q) = \sum_{v \in V_q} \sum_{B \text{ of } v} n_B \tag{5}$$

where n_B is the number of occurrences of term k in an attribute B of an entity v in the results.

The formula above essentially measures the popularity of terms in the results of a query. Popularity is a very typical measure of tag significance when showing user tags in social bookmarking systems. However, showing in a data cloud terms that are very popular in the search results may not be very useful for the purposes of search refinement. Consider, for example, a query for "`photography`" and assume that most of our results are about "`digital photography`". Showing this term in the data cloud probably would not be very useful, since if the user clicks on it, the results would not change. In addition, globally frequent terms, i.e., terms that are not very representative of the particular set of results will surface in the data cloud. Consequently, term popularity may not always be a good measure of term "usefulness" in the context of refining searches.

(**Relevance-based**) In order to select terms that are more representative of the particular set V_q, an approach is to select those terms that would make good queries for the entities in V_q. We can treat each candidate term k as a one word query and compute the similarity between that term and each matching entity v in V_q. A high score shows that the term and the entity match, hence that entity would be a relevant result for the term k (if issued as a query). Then, we sum up over all entities in the results V_q for the query in order to find how good overall k is for V_q.

To compute the similarity between that term and each matching entity v, we can compute the tf*idf weight of a term k contained in v and then we can sum up over all entities in the results V_q.

$$score(k, q, V_q) = \sum_{v \in V_q} tf_{k,v} * idf_k \tag{6}$$

For computing the tf values for the formula above, we can use the unweighed Formula (1) or the weighted (4) to count for the term position in the database.

(**Query-dependence**) A user query provides an indication of the user intention. The data cloud is generated over the results for a query, not just a random subset of the database. Hence, taking also into account the initial search query that generated the results in the computation of the scores of the candidate summary words may generate a data cloud that is closer to the user information need. For example, consider a search about "`photography`". A search entity in V_q may not be very related, and hence, have a low score w.r.t. "`photography`". Then, terms found in this entity should have lower scores w.r.t. this search as well, even if they are significant for this entity.

We can take into account the relevance of an entity to a user query in the computation of the score for a term k contributed by this entity as follows:

$$score(k, q, V_q) = \sum_{v \in V_q} (tf_{k,v} * idf_k) * score(v, q) \quad (7)$$

4. ALGORITHMS

Given a keyword query q and a database D, we want to compute the set V_q of matching entities and the summary S of top K terms for these results. Although we do not search at the level of tuples but at the level of search entities, search entities serve only as a useful abstraction of the underlying database. In practice, the collection V is never materialized. In this section, we present different methods to generate the answer V_q and its summary S on top of D.

4.1 Searching on the Tuple Graph

Existing approaches to keyword searching over databases use inverted indexes that store information about term occurrences in the database tuples. We call these tuple-based inverted indexes. In this section, we explore a solution that uses a tuple-based inverted index for finding search entities that match a query. Irrespective of how such an index is physically stored, we can think of information stored in it in the form of a tuple $< k, R, A, t, n >$, which captures the fact that the attribute A of tuple t in relation R contains n occurrences of the term k.

The algorithm TBI is shown in Figure 7. Its inputs are the query q and a constraint K on the number of summary words to generate. It accesses the tuple-based index I and the database D. It generates the set of entities V_q and the summary S for the query q. The general idea is the following. The algorithm uses the tuple-based inverted index to find which tuples in the database contain the query terms (ln: 2.2). Each occurrence of a query term in the index is then linked to the search entity it conceptually belongs to (ln: 2.3). At this phase, the search entity inherits any statistics from the query term that will be used for computing how relevant the entity is for the query. Having found for each query term k, the set of search entities that contain this term, the algorithm takes the intersection of the sets for all the query terms to find the entities that match the query (ln: 3). Then, it ranks the search entities aggregating on their inherited statistics (ln: 4). The final part of the algorithm finds all other words contained in the search entities, computes their scores and keeps the top K words (ln: 6-8).

We zoom in now on how term occurrences in the database tuples are linked to search entity ids (ln: 2.3). For example, if the term "`graph problems`" is found in the tuple t_4 in the tuple graph of Figure 5, then the system needs to find the path on the tuple graph to the tuple (t_1) in the primary relation and get the entity id (if such a path exists). This can be found by executing a parameterized query that makes the necessary lookup. For each relation R that joins to the primary relation either explicitly or indirectly through other relations, there is a parameterized query Q_R that connects tuples from R to entity ids in R_0. For example, for the relation Comments which contains our example tuple t_4, the parameterized query could be as simple as this one:

```
select    CC.CourseID
from      Comments CC    where      CC.t ='?'
```

Algorithm TBI

```
Input:    a query $q$, a database $D$, a constraint $K$
          a tuple-based inverted index $I$
Output:   set $V_q$ of search entities, summary $S$ of top $K$ terms
Begin
1.  $V_q := \varnothing$; $S_q := \varnothing$
2.  Foreach term $k$ in $q$
2.1.     $S_k := \varnothing$
2.2.     $I' :=$ all tuples in $I$ that contain terms in $k$
2.3.     Foreach $I_R \subset I'$ with a single relation $R$
2.3.1.       $S := Q_R(I_R)$
2.3.2.       $S_k := S_k \cup S$
3.  $S_q := \cap_k S_k$
4.  Foreach $v$ in $S_q$
4.1.     $s_v := score(v, S_q)$
4.2.     add $(v, s_v)$ in $V_q$
5.  $S' := \varnothing; W := \varnothing$
6.  Foreach secondary relation $R$
6.1.     $T := Q^0_R(V_q, R)$
6.2.     $I' :=$ all tuples from $I$ that join with the tuples in $T$
6.3.     $W := W \cup I'$
7.  Foreach term $w$ in $W$
7.1.     $s_w := score(w, W)$
7.2.     insert $\{w, s_w\}$ in $S'$
8.  $S :=$ top-K terms of $S'$
9.  output $V_q$ and $S$
End
```

Figure 7: Generating the set V_q and the summary S for a query q using a tuple-based inverted index.

where t is the tuple where a keyword is found. In other cases, the parameterized query may be more complex (if more than one join is involved.) This lookup must be done for all tuples in the index where the query keywords are located.

In order to find the words contained in the matching search entities, apart from the query terms, (ln: 6), the algorithm executes a different set of parameterized queries. For each secondary entity relation R, a parameterized query Q^0_R finds which tuples T in R join with the tuples in the primary relation R_0 that map to the search entities for the query q. Then, for the tuples in T, it reads the terms contained in them using the index I. Finally, for all words found in the matching entities, the algorithm computes their scores (ln: 7). The list S' contains one entry per word and is ordered on descending score. The top K words comprise the summary S for the results V_q.

4.2 Entity-based Inverted index

Using a tuple-based inverted index generates an overhead during query processing, because the final result, which comprises of entities (entity ids) not tuples (tuple ids) and the words that they contain need to be constructed on the fly. For this reason, we use an *entity-based inverted index*, which stores information of the form $< k, B_0, R, A, t, n >$, where B_0 is the id of the entity that contains the term k.

To generate this index, we first build the tuple-based inverted index for the database. Then, we execute a set of parameterized queries to generate the B_0 information. The procedure is similar to the one described above using parameterized queries to link term occurrences found in the database tuples with tuples in the primary relation. In this way, we move the processing overhead from the query time to the off-line, preprocessing, phase. If we want to search for more than one type of entities in the database (e.g., course entities and instructor entities), we can store other entities'

Algorithm 3PA

```
Input:    a query $q$, a constraint $K$
          an entity-based inverted index $I$
Output:   set $V_q$ of search entities, summary $S$ of $K$ terms
Begin
1.    $E_q := \varnothing$; $V_q := \varnothing$
2.    Foreach term $k$ in $q$
2.1.      $E_k :=$ all entity ids in $I$ that contain $k$
3.    $E_q := \cap_k E_k$
4.    Foreach $v$ in $E_q$
4.1.      $s_v := score(v, q, I)$
4.2.      add $(v, s_v)$ in $V_q$
5.    $S' := \varnothing$
6.    $W$ := all tuples from $I$ with entity id in $E_q$ except
          those containing query terms
7.    Foreach term $w$ in $W$
7.1.      $s_w := score(w, W)$
7.2.      insert $\{w, s_w\}$ in $S'$
8.    $S$ := top-K terms of $S'$
9.    output $V_q$ and $S$
End
```

Figure 8: Generating the set V_q and the summary S for a query q using an entity-based inverted index.

ids in the entity-based index as well.

The algorithms that we describe in the following sections for computing the set V_q of matching entities and the summary S of top K terms for a query q are based on this index.

4.3 Three-phase Algorithm

The algorithm 3PA, depicted in Figure 8, is a three-phase approach to keyword searching with summaries based on an entity-based inverted index and it is general in the sense that it can be used with any ranking scheme. Its inputs are the query q, a constraint K on the number of summary words to generate and the index I and generates the set of entities V_q and the summary S for the query q. The three phases proceed as follows.

- (*generate matching entities*) The first step of the algorithm identifies the entities, E_q, that contain all the query terms by taking the intersection of all the sets of entities in the inverted index that contain the same single query term from q (ln: 2-3).
- (*rank entities*) The second step computes the score s_v of each matching entity v from E_q based on the query terms and the information recorded in I and adds the entity and its score in the results V_q (ln: 4).
- (*rank words*) The third step retrieves all the words, W, found in the matching entities and computes their scores (ln: 7). The list S' contains one entry per word and is ordered in order of descending score. The top K words comprise the summary S for the results V_q.

Consequently, the algorithm finds the set of matching entities and, then, based on this set, it accesses the index I to compute scores for these entities and for all the words found in them. The index provides all necessary information, so the algorithm does not need to perform any unnecessary accesses to the database. Moreover, since the set of matching entities is reused, it is tempting to materialize it. In the experiments, we study two versions of the algorithm: 3PA$_{no}$ that does not materialize the result and a second one, 3PA$_{mat}$, that creates a temporary in-memory table.

For computing the score of a term using Formula 5, the numbers of term occurrences stored in the index are summed up. For computing the score of a term using Formula 6 using only the number of term occurrences requires a number of different aggregations over the index in order to generate the several parts that are used in the formula. To minimize processing time, we can pre-compute a great number of these aggregations and store in the index for each entity v and term t, the weight of t for this entity. Then, during query time, Formula 6 can be computed over the stored weights.

Algorithm IS

```
Input:    a query $q$, a term $q'$, a constraint $K$
          an entity-based inverted index $I$
Output:   set $V_q$ of search entities, summary $S$ of $K$ terms
Begin
1.      If there is a new query $q$
1.1.        $E_q := \varnothing$
1.2.        Foreach term $k$ in $q$
1.2.1.          $E_k :=$ all entity ids in $I$ that contain $k$
1.3.        $E_q := \cap_k E_k$
1.4.        $I'$ := all tuples from $I$ with entity id in $E_q$
2.      Else
2.1.        rename $IT_{new}$ to $IT_{old}$
2.2.        $E_{q'} :=$ all entity ids in $IT_{old}$ that contain $q'$
2.3.        $I'$ := all tuples from $IT_{old}$ with entity id in $E_{q'}$
3.      $V_q := \varnothing$; $S' := \varnothing$
4.      Foreach tuple $< v, w, n >$ in $I'$
4.1.        If $w$ is a term in $q$
4.1.1.          $s_v := score(v, n, s_v)$
4.1.2.          add or update $(v, s_v)$ in $V_q$
4.2.        Else
4.2.1.          $s_w := score(w, n, s_w)$
4.2.2.          add or update $(w, s_w)$ in $S'$
5.      rename $I'$ to $IT_{new}$
6.      $S$ := top-K terms of $S'$
7.      output $V_q$ and $S$
End
```

Figure 9: Refining searches.

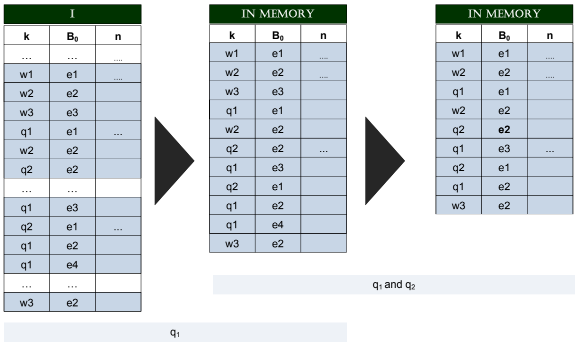
The algorithm 3PA can be easily implemented on top of a database taking advantage of the database's query functionality for performing two separate aggregations over the index: one to compute entity scores and one to compute word scores. Additionally, since entity and term scores are computed separately, this allows showing the results to the user as soon as they are generated and then show the data cloud possibly having a faster first response time.

4.4 Incremental Algorithm

When a search is refined by adding a new term, then the conjunction of the old query and the term is treated as a new query by the algorithm 3PA. A different approach is to actually refine the results and the words shown in the data cloud building on the previous search.

The algorithm IS, presented in Figure 9, builds on a simple observation: the part of the entity-based index that is processed for a search q' that refines a search q is contained within the part of the index processed for q. To illustrate, consider a query q_1 that is refined to q_1 *and* q_2. Figure 10 shows a simplified picture of the index, where k is a term, B_0 stores the entity id and n stores the statistics that we use for ranking. For q_1, we need to read all tuples in the index for the entities e_1, e_2, e_3 and e_4, which match the query. These are shown in color. The tuples that we need to read for the query q_1 *and* q_2 are all tuples for the entities e_1 and e_2, and these comprise a subset of the previous one.

Based on this observation, the algorithm IS maintains two



k	B_0	n
…	…	….
w1	e1	….
w2	e2	
w3	e3	
q1	e1	…
w2	e2	
q2	e2	
…	…	
q1	e3	
q2	e1	…
q1	e2	
q1	e4	
…	…	
w3	e2	

k	B_0	n
w1	e1	….
w2	e2	….
w3	e3	
q1	e1	
w2	e2	
q2	e2	…
q1	e3	
q2	e1	
q1	e2	
q1	e4	
w3	e2	

k	B_0	n
w1	e1	….
w2	e2	….
q1	e1	
w2	e2	
q2	**e2**	
q1	e3	…
q2	e1	
q1	e2	
w3	e2	

Figure 10: Refining searches

sets of tuples in memory: a set IT_{old} that contains the tuples of the previous search (if there is one) and the set IT_{new} that contains a subset of IT_{old} required for the refined search.

If it is a new search, then the algorithm reads everything from the entity-based inverted index. First, it identifies the entities, E_q, that contain all the query terms by taking the intersection of all the sets of entities in the inverted index that contain the same single query term from q (ln: 1.2-1.3). Then, it finds all tuples I' in the index that refer to the matching entities (ln: 1.4). Based on I', the algorithm computes scores for the search entities (ln: 4.1) and for the words contained in them (ln: 4.2). Finally, the set I' of tuples of the index that refer to the matching entities is renamed to IT_{new} and is kept. Hence, as Figure 10 illustrates, for executing the search q_1, the algorithm will access the colored part of the index and materialize it.

If a term q' is added in an existing search q, then the algorithm follows a different procedure to generate the set I' that is used for scoring the entities and their words for a search. The algorithm renames IT_{new} to IT_{old} to keep the results of the previous search q. In order to find the entities that contain both q and q', it just needs to find from IT_{old} the entities that contain q' (ln: 2.2). These are the matching entities for the query q *and* q'. Then, it finds all tuples I' in IT_{old} that refer to the matching entities (ln: 2.3). This is the set to be used for ranking the entities and the words for the refined query (ln: 4). As Figure 10 shows, for executing the search q_1 *and* q_2, the algorithm will access the in-memory part of the index generated previously for q_1 and will generate a smaller one.

To score the entities and the words found for a search, the algorithm processes the set I', which, as we have seen, is generated through two different paths depending on the type of search. For presentation purposes, we denote a tuple in I' as $< v, w, n >$, where v refers to an entity, w is a term found in this entity, and n are the statistics kept for the pair (v, w). A tuple $< v, w, n >$ contributes to the score of the entity v if w is a query term. Otherwise, it contributes to the score of the term w found in v. The algorithm works with ranking functions that are incrementally computable, i.e., they should either be distributive or algebraic. The ranking formulas we are using have this property.

5. SYSTEM OVERVIEW

Our system architecture is depicted in Figure 11. 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.

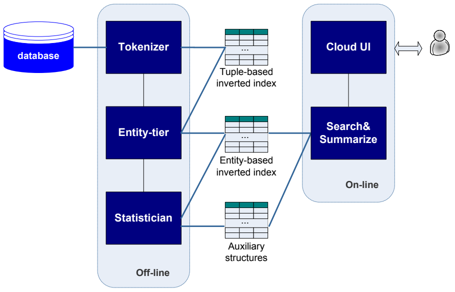


Figure 11: System architecture

We search at the level of search entities. As we have explained in Section 4.1 using directly the tuple-based inverted index at query time to generate the set of entities that match a query is not straightforward. 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 "`art`" 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. Then, additional lookups are required in order to find the words contained in the matching entities. Overall, a large number of accesses to the database and the index are required. To save query time, we prefer to store the information of which entity each term belongs in advance in an entity-based inverted index.

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 as word weights and courses 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.

The *Cloud UI* presents the results with a visualization of the data cloud. For each course, its code, title, and a snippet from its description are shown. There is a variety of ways to implement tag clouds. We have chosen the typical cloud appearance: words are sorted alphabetically and the most important terms are highlighted via an appropriate font size. We show the top 35 words using Formula 7.

6. EXPERIMENTS

In this section, we summarize our experimental findings with respect to the following questions: (a) the appropriateness of the presented ranking methods for summary words, (b) the effectiveness of the CourseCloud compared to the standard search and browse interfaces in CourseRank and (c) the performance of the proposed algorithms.

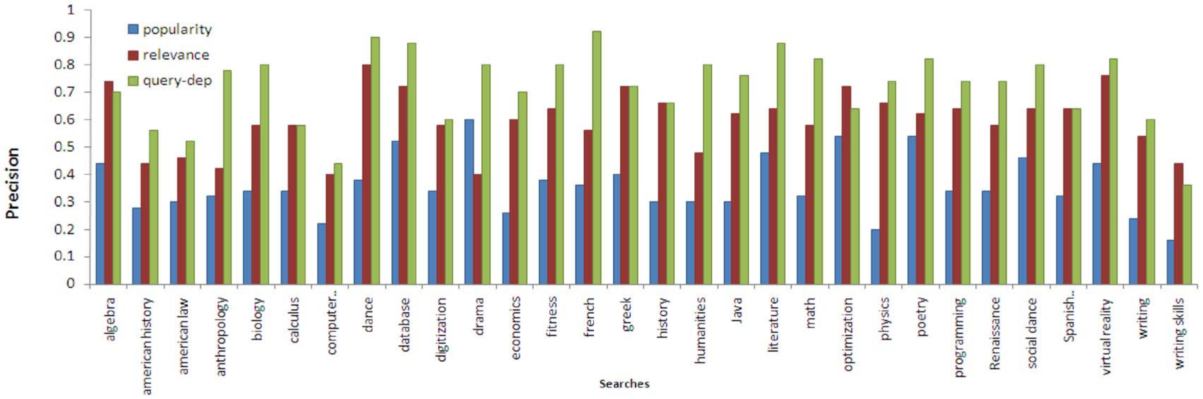


Figure 12: Comparing CourseClouds w.r.t. precision

Keyword Searches		
algebra	american history	american law
anthropology	biology	calculus
computer science	dance	database
digitization	drama	economics
fitness	french	greek
history	humanities	Java
literature	math	optimization
physics	poetry	programming
Renaissance	social dance	Spanish literature
virtual reality	writing	writing skills

Table 1: Searches for evaluating ranking formulas

6.1 Ranking methods

In Section 3.2, we described different approaches for computing a score for a candidate summary word found in the results of a query. We will discuss the results of evaluating Formula 5 (popularity), Formula 6 (relevance), and Formula 7 (query-dependence). In the experiments that we present, we use attribute weights that were selected after experimenting with different sets of weights over CourseRank's database. For the evaluation, we recruited three students and built with them the set of keyword searches shown in Table 1. Each student rotated through the data clouds generated for each search by each formula, without knowing which formula was used each time. As an indication of the "goodness" of a data cloud, we considered precision, i.e., the number of relevant terms vs. the total number of terms in the cloud. The latter is set to 50.

Figure 12 shows the precision of the data clouds for all searches and for all the word ranking methods considered and Figure 13 shows the average precision per ranking method over all searches. For any particular data cloud, we took the average of the opinions of the three individuals. Observing the two figures, we can see that the popularity formula achieved a precision in the cloud in the range of 0.18 to 0.55 with the average precision being below 0.4. The clouds with higher precision were the ones for `poetry`, `optimization` and `database`. Overall, as Figure 13 illustrates, the popularity formula is the worst choice. Using relevance improves the minimum precision to 0.4 and achieves on average higher precision in the data cloud compared to popularity. The data clouds for `algebra`, `dance` and `virtual reality` have the highest numbers of relevant terms, reaching a precision of 0.8. Taking into account the query context, i.e., using the query-dependence Formula 7, achieves overall higher precision in the data cloud (with an average around 0.7). There is one exception: the data cloud for `writing skills`. We believe that it is the type of the query that does not impose a strong context for choosing terms for the cloud: writing skills were discussed in a variety of courses with only a small subset referring to courses focusing on writing (or improving writing skills).

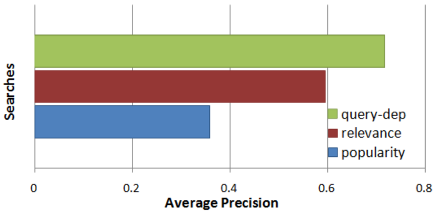
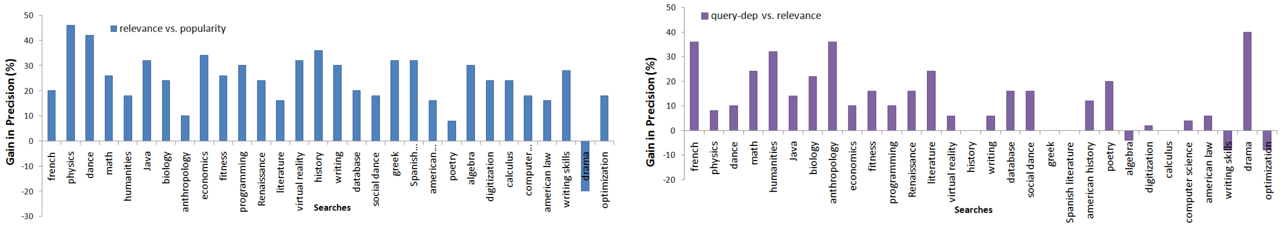


Figure 13: Comparing clouds w.r.t. avg precision

Overall, Figure 13 shows a clear order of the three methods but Figure 12 illustrates significant variations in how much each cloud gains in precision when switching to a better ranking method. Figure 14 highlights this gain in precision when switching from the popularity formula to the relevance formula (Figure 14(a)) and from the relevance formula to query-dependence (Figure 14(b)). We observe that relevance does "most of the job" for some terms, such as `history`, `Spanish literature` and `greek`, whereas the contribution of the query context using query-dependence is negligible for these terms. An explanation for that seems to be that for these words the entities found had high scores, i.e., they were already very relevant to the search. Consequently, there were not many irrelevant entities that could be filtered out by imposing the query context. Actually, the clouds that benefited the most when using relevance instead of popularity were `physics`, `dance`, and `history` because their terms were drawn from entities that scored high for these searches. Interestingly, the data cloud for `drama` using relevance lost in precision compared to the cloud using popularity. The reason for that was that terms that were very rare surfaced in the cloud but these were not popular words for `drama`.

Figure 15 orders clouds based on the overall gain in precision when using the query-dependence formula. In essence, this formula tries to balance three factors that should play



(a) relevance gain over popularity

(b) query context gain over relevance

Figure 14: Gaining in precision.

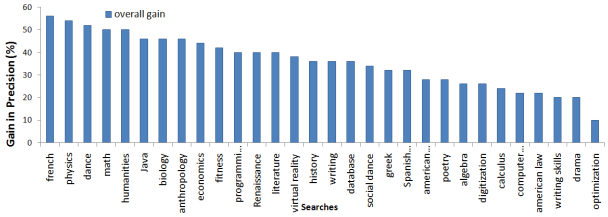


Figure 15: Overall gain

a role in the word significance: popularity in the search results, representability (by bringing *idf* into the picture), and the query context (by weighting the contribution of terms contained in an entity considering how relevant the entity is for the search). This is the formula we use in the current release of the CourseCloud in CourseRank.

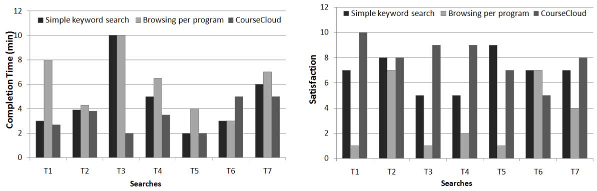
The three evaluators also observed that the quality of the terms improved when using the query context and attribute weights. Figure 16 provides the words found in the data clouds for two searches. We can see that there are very relevant and useful terms, such as `Shakespeare` and `Directing` for `drama`. Also, in the data cloud for `greek`, one could find connections that he might not have thought of, such the connection of `greek` and `science`.

6.2 User study

We have performed a limited user study with 5 users before releasing CourseCloud. As we have mentioned earlier in the paper, CourseRank offers two standard interfaces: one for browsing courses per department and program and a keyword search interface that allows searching courses based on keywords found in the title and their description. We want to have a flavor of the effectiveness of CourseCloud as a unified search and browsing interface and the usefulness of the data cloud for summarizing results and refining searches.

We designed a set of seven tasks for the participants all around the idea of finding courses to satisfy a particular need. The tasks, T_1 to T_7, were assigned to all users and were executed in that order. Examples of these tasks were: "find a class to help me lose weight" (T_1), "learn java programming" (T_2), "learn about civil rights" (T_3) and "every day use of English" (T_4). We measured the completion time (in minutes), i.e., the time required by each user to complete or abandon a task. Each user had at maximum 10 minutes for each task using a particular interface. In addition, we asked them to evaluate their experience giving a degree between 1 and 10 (10 for high satisfaction). Figure 17 shows the average completion time and user satisfaction per task.

We observe that there were cases where all interfaces did equally well in terms of how much time the participants



(a) completion time

(b) satisfaction

Figure 17: User study.

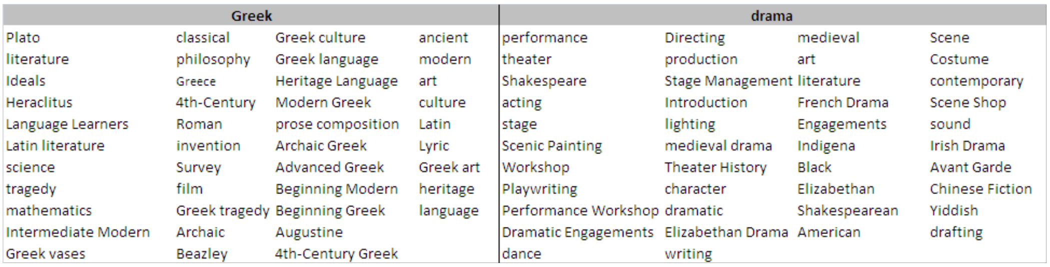
required to complete the task and their satisfaction. For example, for the task T_2, it was quite clear what kind of courses the person was interested in and was easy to locate them based on the department and the title because the most relevant courses have the word `java` and `programming` in their title and they are provided by the computer science department.

For another category of tasks, such as T_1 and T_4, the participants could find courses that they would select using each interface. However, they were less satisfied with the browsing option, because they had to inspect many courses that could be potentially relevant based on the department that offered them. They were more satisfied with CourseCloud rather than simple keyword search for a number of reasons. For example, using simple search for T_1, they typed words, such as `fitness`, and browsed the courses offered by the ATHLETICS program. Expectedly, they found very relevant options, but when using CourseCloud, they located courses they had not thought of, such as a course on nutrition that was offered by a different department. Apart from discovering unexpected options as the example above, the options offered had more variety. For example, for T_4, one participant chose a course on intensive English language and another chose a course on improving written English. Finally, some tasks, such as T_3 and T_5, were abandoned when using the browsing interface, either because the users could not find the right program or/and there were many courses to browse. This fact explains the relatively small completion times shown in the figure for the browsing interface.

These indications show the potential of CourseCloud for enhancing serendipity and diversity. We intend to mine CourseRank's user logs in order to gain deeper insight into the usage patterns and the impact of CourseCloud.

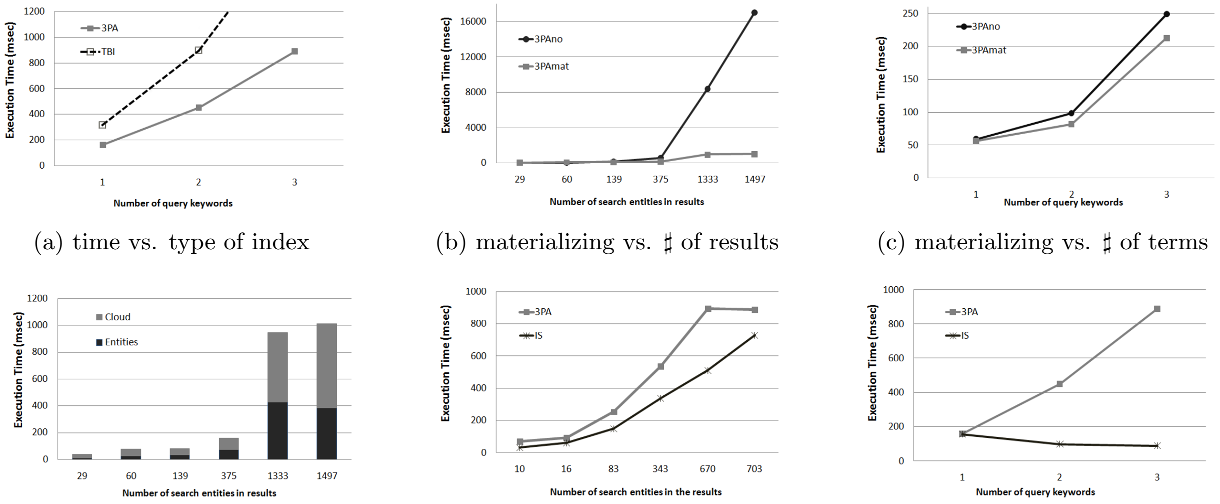
6.3 Performance

We organize performance results around four main questions and we show how execution times are affected by (a) the number of query words and (b) the number of the search entities returned for a search. A third parameter, the (average) number of words found per entity, has a similar effect



Greek				drama			
Plato	classical	Greek culture	ancient	performance	Directing	medieval	Scene
literature	philosophy	Greek language	modern	theater	production	art	Costume
Ideals	Greece	Heritage Language	art	Shakespeare	Stage Management	literature	contemporary
Heraclitus	4th-Century	Modern Greek	culture	acting	Introduction	French Drama	Scene Shop
Language Learners	Roman	prose composition	Latin	stage	lighting	Engagements	sound
Latin literature	invention	Archaic Greek	Lyric	Scenic Painting	medieval drama	Indigena	Irish Drama
science	Survey	Advanced Greek	Greek art	Workshop	Theater History	Black	Avant Garde
tragedy	film	Beginning Modern	heritage	Playwriting	character	Elizabethan	Chinese Fiction
mathematics	Greek tragedy	Beginning Greek	language	Performance Workshop	dramatic	Shakespearean	Yiddish
Intermediate Modern	Archaic	Augustine		Dramatic Engagements	Elizabethan Drama	American	drafting
Greek vases	Beazley	4th-Century Greek		dance	writing		

Figure 16: Example clouds



(a) time vs. type of index

(b) materializing vs. ♯ of results

(c) materializing vs. ♯ of terms

(d) data cloud overhead

(e) refinement vs. ♯ of results

(f) refinement vs. ♯ of terms

Figure 18: Execution times for keyword searching and data cloud generation.

on times as the number of entities in the results. Hence, to save space, we do not discuss this parameter any further.

• What is the benefit of the entity-based inverted index?

The entity-based inverted index is generated once we have the tuple-based inverted index. If we save the extra processing time (although it is off-line time) and we use instead the tuple-based inverted index for keyword searches, will searches be considerably slower? Figure 18(a) compares the algorithm TBI, which uses this type of index, to 3PA, which works with the entity-based index for queries with increasing number of keywords. The performance of TBI deteriorates considerably because for each additional term in a query, it has to perform additional accesses to the index and execute additional parameterized queries to find the search entities that contain the term before intersecting the sets of entities that match different query terms. Similarly, TBI's performance deteriorates with the number of search entities returned because it processes more words contained in these entities executing several parameterized queries over the database. We do not show these results due to space constraints.

• Is the materialization of intermediate results significant?

We study the two versions of the three-phase algorithm: 3PA$_{no}$ that does not materialize the results and a second one, 3PA$_{mat}$, that creates a temporary in-memory table and uses it to compute the entity and the word scores. Figure 18(b) shows the effect of the number of entities on execution times. We observe that, for few entities, 3PA$_{no}$ and 3PA$_{mat}$ have similar performance. As the number of entities returned increases, the execution time increases too. Dealing with more entities means also processing more words for generating the data cloud. While without using materialization, performance decreases significantly, 3PA$_{mat}$'s execution times increase smoothly with the number of entities. Hence, the overall performance is benefited when materializing large result sets. Figure 18(c) shows the effect of the number of query keywords for searches that return approximately 50 entities and 1000 words. Execution time goes up and materializing the results makes only a small difference. Based on the above observations, we use 3PA$_{mat}$ (hereafter referred simply 3PA) due to its smooth behavior both for complex queries (typically queries have around 2 to 3 terms on average [25]) and for many results (containing probably many words to process for generating the data cloud).

• What is the overhead of a data cloud?

Observing execution times in the previous figures, a question that naturally arises concerns the amount of time spent to processing the words in the results in order to generate the data cloud. Figure 18(d) answers this question by dividing the execution times shown in Figure 18(b) for 3PA$_{mat}$ to keyword search time (i.e., the time required to find and rank the entities that match a search) and the data cloud time (i.e., the time required to find the words contained in the matching entities and rank them in order to select the top words for the cloud). The average number of words processed per matching entity for this experiment is 100. The data cloud time is a significant fraction of the overall time.

• Executing a search from scratch or building on the results of a previous search?

When a user refines an existing search, there are two options: (a) consider the modified query as a new query and execute it using the algorithm 3PA or (b) use IS to refine the search and the data cloud based on the results of the previous search that have been cached. Figures 18(e) and 18(f) compare the two options. Figure 18(e) shows execution times for two-keyword queries depending on the number of results they generate. Each query refines a single word query that has been executed and its results are cached so that IS can re-use them. We observe that using IS we achieve better times. Figure 18(f) shows execution times for gradually refining a single word query to a 3-word query. The searches considered need additional time when using $\mathsf{3PA}_{mat}$. Nevertheless, with IS, we achieve to give to the user the feeling that refinement should intuitively provide: as the user refines a search and gets fewer results, time should not increase.

7. CONCLUSIONS

In this paper, we proposed a framework that allows thinking of searches more naturally, i.e., in terms of "search entities" that may span multiple tables rather than tuples and couples the flexibility of keyword searches over structured data with the summarization and navigation capabilities of tag clouds. The cloud presents the most significant words found in the search results. We have described several methods to compute the scores both for the entities and for the terms in the search results and we have described algorithms that compute or refine the set of results for a search and the set of words in the data cloud. We presented experimental results regarding the performance of the algorithms, the appropriateness of the methods for computing scores for words in the results, and the effectiveness of data clouds.

There is a large agenda of interesting issues to tackle. An open question is what are the best methods for scoring and selecting words in the results. In our experiments, we saw that we can increase the number of relevant terms in the data cloud using different scoring schemes but there are other methods as well. For example, we intend to explore the effect of information measures, such as Kullback−Leibler divergence [8]. Using such measures over dynamically computed results is a computational challenge. We are also interested in exploring different criteria for selecting words, such as selecting words based on how well they partition the results. CourseCloud provides a live testbed to tackle such issues and evaluate different solutions. Mining the user logs will provide valuable insight into the impact and the usefulness of the data clouds in a real environment.

8. REFERENCES

[1] Del.icio.us: url: http://del.icio.us/.

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

[3] ManyEyes: http://services.alphaworks.ibm.com /manyeyes/page/tag_cloud.html.

[4] technorati: url: http://www.technorati.com/.

[5] Wikipedia {Tag Cloud}: url: http://en.wikipedia.org/wiki/tag_cloud.

[6] Wordle: http://http://wordle.net/.

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

[8] C. Arndt. *Information Measures: Information and Its Description in Science and Engineering.* Springer, 2004.

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

[10] O. Ben-Yitzhak, N. Golbandi, N. Har'El, R. Lempel, A. Neumann, S. Ofek-Koifman, D. Sheinwald, E. Shekita, B. Sznajder, and S. Yogev. Beyond basic faceted search. In *Proc. of* 1^{st} *Int'l Conf. on Web Search and Data Mining (WSDM)*, pages 33–44, 2008.

[11] 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.

[12] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[13] D. Coupland. Microserfs. In *Flamingo*, 1996.

[14] M. Daconta, L. Obrst, and K. Smith. *The Semantic Web: A guide to the future of XML, Web services, and knowledge management.* John Wiley & Sons, Indianapolis, 2003.

[15] G. Das, V. Hristidis, N. Kapoor, and S. Sudarshan. Ordering the attributes of query results. In *SIGMOD*, pages 395–406, 2006.

[16] M. Dredze, H. Wallach, D. Puller, and F. Pereira. Generating summary keywords for emails using topics. In *IUI*, pages 199–206, 2008.

[17] Y. Hassan-Montero and V. Herrero-Solana. Improving tag-clouds as visual information retrieval interfaces. In *Int'l Conf. on Multidisciplinary Information Sciences and Technologies (InSciT2006)*, 2006.

[18] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003.

[19] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[20] O. Kaser and D. Lemire. Tagcloud drawing: Algorithms for cloud visualization. In *WWW*, 2007.

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

[22] B. Y-L Kuo, T. Hentrich, B. Good, and M. Wilkinson. Tag clouds for summarizing web search results. In *WWW*, pages 1203–1204, 2007.

[23] M. Miah, G. Das, V. Hristidis, and H. Mannila. Standing out in a crowd: Selecting attributes for maximum visibility. In *ICDE*, pages 356–365, 2008.

[24] Addison Gallery of American Art. http://978.andover.edu/addison/about.htm.

[25] Trellian. http://www.keyworddiscovery.com/ keyword-stats.html.