Freebase Cubed: Text-based Collection Queries for Large, Richly Interconnected Data Sets

David F. Huynh
Metaweb Technologies, Inc.
631 Howard Street, Suite 400 San Francisco, CA 94105
david@metaweb.com

ABSTRACT

Any large data set such as Freebase that contains a large number of types and properties accumulated over actual use rather than fixed at design time poses challenges to designing easy-to-use faceted browsers. This is because the faceted browser cannot be tuned with domain knowledge at design time, but must operate in a generic manner, and thus become unwieldy.

In this work, we propose that support for a particular kind of text-based queries can let users perform faceted browsing and set-based browsing operations on such data sets with the ease and familiarity of conventional keyword search. For example, the text query "german car companies founders" can replace the actions of filtering all Freebase data by type to "company", by industry to "car", and by country to "Germany", and then pivoting to those companies' founders. From there, the user can perform faceted browsing actions to refine the already narrow collection further. We describe an algorithm for parsing these *collection queries* and demonstrate an implementation that works on Freebase.

Categories and Subject Descriptors

H.5.2 [**User Interface**]: Interaction Styles, Natural Language.

General Terms

Algorithms, Design, Human Factors, Languages.

Keywords

Search, pidgin, faceted browsing, set-based browsing, graph data.

1. INTRODUCTION

The faceted browsing paradigm has been very effective in letting casual users browse through large data sets by performing simple actions of picking suggested filters to apply. This paradigm remains effective as long as the schemas in the system are known a priori so that the interface can be configured based on the schemas. For example, an online retailer can be expected to know all types of product that it offers, as well as important aspects of each type of product (e.g., resolution for televisions, maximum zoom for cameras). Such domain knowledge helps configure the faceted browser, making it optimal for the domain in question.

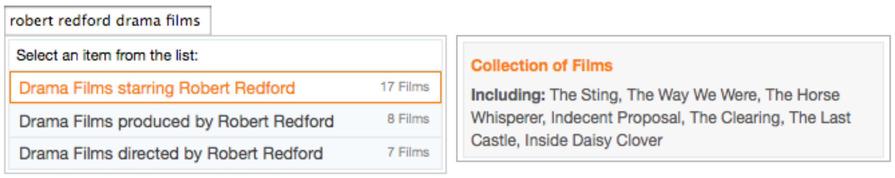
Domain knowledge is difficult to obtain and use in such a data set as Freebase in which new schemas are added over actual use rather than fixed at design time. Freebase currently contains some 3,000 types (e.g., company, book author) and over 30,000 properties (e.g., country where a company is founded, books written by an author). Users can add new topics to existing types or they can add entirely new types and properties. Providing a faceted browser over even just the stable types and properties is still a challenge for two reasons.

- Consumer-facing faceted browsers typically have one over-arching type under which all data can be organized. For example, online retailers deal primarily with products; libraries deal with books. In Freebase, there is no over-arching type: users are as likely to search for companies as they are for book authors, or any one of the 3,000 types.
- Types in existing faceted browsers are isolated. When users search for televisions in an online retailer, there's little chance that they would be concerned with cameras at the same time. In contrast, on Freebase where types are highly interconnected, users might want collections defined by multiple types, such as "pharmaceutical companies funding republican politicians' campaigns." The more interconnected the types are, the more the potential ways to define collections, and the more facets the browser has to offer.

These challenges are inherent in any data set that resembles Freebase. That includes other comprehensive semantic web data sets such as Dbpedia, or even smaller, personal semantic web data sets accumulated by gathering tidbits from several data sources using something like Tabulator [3] or Piggy Bank [4].

We note that these challenges are acute at the beginning of any faceted browsing session when the collection to deal with is still large and/or heterogeneous. After applying a few filters, the collection gets small and homogeneous enough for faceted browsing to be effective again.

We propose that just when the user wants to search a large, interconnected data set, a particular class of text-based queries can be supported for filtering the data down to a manageable


robert redford drama films

Select an item from the list:

Drama Films starring Robert Redford	17 Films
Drama Films produced by Robert Redford	8 Films
Drama Films directed by Robert Redford	7 Films

Collection of Films

Including: The Sting, The Way We Were, The Horse Whisperer, Indecent Proposal, The Clearing, The Last Castle, Inside Daisy Clover

Figure 1. As the user types a query into the Freebase Cubed suggest widget, the query is interpreted and the interpretations are shown in a drop-down menu. When an interpretation is hovered using the mouse or selected using the keyboard, a fly-out appears to show the interpretation's details.

collection. These *collection queries* have a syntax that maps to faceted browsing operations (filtering) and set-based browsing operations (pivoting), and are sufficiently natural for casual users. An example is “german car companies founders”, which maps to filtering by type to company, by industry to car, by country to German, and then pivoting from those companies to their founders. Other examples include:

- frank wright buildings
- female african american politicians
- kate winslet drama films directors

We observe that some existing web searches already follow this syntax. Issuing such a query today performs a keyword text search, whereas we propose that it be translated to and performed as a structured query.

We describe an algorithm for parsing collection queries, demonstrate a prototype called “Freebase Cubed,” and show how, as a simple search textbox, it fits into more places than faceted browsing UIs can (Figure 1). We plan to use this widget on Freebase.com and in Freebase-powered web applications to introduce users early to collections and surface the richness of the data within Freebase even in places where full-blown faceted browsing UIs cannot be afforded.

2. GENERAL DEMAND

While Freebase as an enormous, richly-interconnected data source needs something more than the conventional faceted browsing paradigm to be accessible to the general public, we wanted to further determine if that solution will also benefit the Web itself, and the Data Web to come. To know if web users already issue web searches resembling collection queries, we conducted a preliminary investigation over web searches relevant to the types in Freebase using Google Insights [1] and Google Suggest API [2]. For each of the few hundred most populated types in Freebase, we pluralized its name (e.g., /government/polician to “policians”) and submitted that text to Google Insights, from which we could download a table of web searches that Google deemed related to that text (e.g., “indian politicians”, “female politicians”). For each of those related web searches, we ran it through the Google Suggest API to retrieve its search volume. While there is no official documentation for the Google Suggest API, developers on the Web have assumed that the “num_queries” values that it returns are search volumes. For example, Google Suggest API returns 102 million “num_queries” for “black politicians”. If this assumption holds, our investigation indicates sizeable search volumes for web searches related to type names in Freebase, and many of these web searches do take the form of collection queries.

From this investigation, we contend that these non-negligible search volumes on what we consider to be collection queries are evidence that some web users already want to search for information on collections (e.g., “black politicians”) rather than single, specific topics (e.g., “barack obama”). Not only does this early result signify demand, but it also suggests that if we were able to support collection queries, then users would be ready to use them without much learning.

3. INTEGRATION USE CASES

In addition to the need to make Freebase browse-able and the general demand for supporting collection queries, we also have another family of use cases in mind. Freebase is a rich source of data that can be used to augment web sites. For example, a prolific political blogger might wish that her readers can easily browse through or search her blog posts not just by the names of politicians mentioned, but also by the politicians’ attributes, such as their parties, their religions, their states, etc., which might or might not be mentioned explicitly in each post. All such data is or can be maintained in Freebase, and if each blog post carries Freebase identifiers of politicians, or any other topic that it also mentions, then compelling search and browse interfaces can be built on top of the two data sources: the blog’s index of topic identifiers for each post, and Freebase.

Depending on each particular integration scenario, there might or might not be enough screen real estate for a full-blown faceted browser. There might be just enough room for a search text field, but a desire to support semantically rich searches nevertheless. Even if there is room for a faceted browser, configuring the browser any way would not leverage all the data in Freebase. So while the blogger might anticipate the need for facets such as “political party,” “state,” and “religion,” the reader looking for “movie actor politicians” won’t be satisfied by those facets. A more free-form mechanism is needed, and supporting text-based collection queries is one possible answer.

Supporting collection queries in a particular embedding scenario has different requirements than on the Freebase site. Specifically, we want to support shortcuts such that, say, on a book review blog, the user can just type “african american authors” rather than “african american authors’ books” to search for books by African American authors.

4. ALGORITHM

A text query might be targeting a single topic or a collection of topics. Thus, given a text query, we would need to make either single topic suggestions or collection suggestions, or both. In order to make collection suggestions, we need to interpret the query as a collection query; this is discussed in 4.1. In order to suggest some combination of single topic suggestions and collection suggestions, we need an overarching algorithm for generating such combination; this is discussed in 4.2.

The entire discussion is posed in the context of Freebase, but it should be applicable to any similar heterogeneous graph data set. The natural language in which text queries are posed is assumed to be English. The generic prototype called Freebase Cubed is accessible at http://cubed.freebaseapps.com/, and a book embedding demo is available at http://cubed.freebaseapps.com/embed-books.

4.1 Collection Query Interpretation

A text query is a collection query when, by our definition, it can be translated to a sequence of filtering and/or pivoting operations. Assuming that a text query is a collection query, we interpret it in three phases.

4.1.1 Chunking

First, we decide how the text query should be broken down (chunked). For example, “robert redford drama films” can be chunked in many ways, e.g.,

- robert + redford drama films

- robert redford + drama films
- robert redford drama + films
- robert + redford + drama films
- robert redford + drama + films

In each chunking solution, each chunk is supposed to correspond to a topic (such as /en/robert_redford), or a type (e.g., /film/film), or a property (e.g., /film/film/directed_by). To determine what each chunk matches, we query for the chunk's text against Freebase's text search service. Several search matches are kept per chunk, but the best match dictates the chunk's *form* (type, property, or topic). Any chunk whose text is a plural noun (e.g., "films" rather than "film") is biased to type or property form more than topic form. Demonyms (e.g., Canadian) are resolved to their corresponding countries (e.g., /en/canada).

Chunking solutions can be ordered by how well the chunks in each solution match against data and schema in Freebase. They are fed in that order into the next phase.

4.1.2 Seeding

Given a chunking solution which consists of an ordered list of chunks, this phase picks one of those chunks to be the starting point from which filtering and pivoting operations are applied. There are two types of seed:

- seed collection defined by a type chunk, such as "authors" in "french authors' books"
- seed topic defined by a topic chunk, such as "jfk" in "jfk's children".

As there are substantially fewer types than topics, a type match in the chunking phase is much less ambiguous than a topic match. Thus, as a rule of thumb in picking seeds, we favor type-form chunks over topic-form chunks.

A chunking solution plus the choice of a seed form a seeding solution. A structured query called the seed query is formulated to represent the seed collection or the seed topic.

4.1.3 Growing

Given a seeding solution, this next phase interprets the rest of the chunks, one by one, as filtering and pivoting operations. For example, having chunked "french authors' books" into french + authors + books and picked the "authors" chunk as the seed collection, this phase interprets the "french" chunk as a filtering operation (filtering by nationality), and the "books" chunk as a pivoting operation (pivoting from authors to their works).

The chunk immediately to the left of the already interpreted chunks is considered first, and then the chunk to the right, in order to favor the adjective-noun ordering of English. For example, starting from the seed "films" in the collection query "drama films actors," we consider filtering the films by genre first before considering to pivot to the films' actors. If the user phases the query as "films drama actors," we can still interpret it.

Type-form chunks and property-form chunks are interpreted as pivoting operations, and topic chunks are interpreted as filtering operations. Applying such an operation means extending the current structured query either by adding a constraint for filtering or by wrapping the current query as a nested query for pivoting.

The current structured query denotes a single topic of some types or a collection of topics sharing some common types. The chunk to be considered for growing that query is also associated with one or more types. For example, in the query "drama films actors," the chunk "drama" is typed /film/genre, and the chunk "actors" is typed /film/actor. The seed collection "films" is typed /film/film. To grow the seed collection with the chunk "drama", we retrieve properties that point from type /film/film to /film/genre. There is only one such property: /film/film/genre. Next, to grow the collection of drama films with the chunk "actors", we retrieve properties that point from type /film/film to /film/actor, and we get /film/film/starring.

There are cases where we get more than one connecting property. Consider the query "robert redford films" in which "robert redford" is typed /film/actor, /film/director, and /film/producer. In growing the seed collection "films" with the chunk "robert redford", we get three different connecting properties: /film/film/starring, /film/film/directed_by, and /film/film/produced_by. These lead to three final interpretations of the query which are shown as three suggestions (Figure 1); and the user is asked to select one.

Note that for each chunk, the chunking phase keeps several search matches. These are useful when the chunk's best match depends on other chunks in the text query. For example, in the query "apple products", the universal best match of "apple" is the fruit, but in the context of "products", the best match is Apple Inc. the company.

While we typically grow the structured query one chunk at a time, when we encounter a property-form chunk, we can use it to qualify the next immediate chunk, and in doing so, grow the query by two chunks in one shot. For example, in the query "robert redford directed films," the chunk "directed" matches the property /film/film/directed_by, which we use to qualify the connection between "films" and "robert redford". This leads to a single interpretation (rather than three previously).

Given a seeding solution, when all chunks have been used up in growing the seed, we have one possible complete *interpretation* of the original text query, which can be expressed as a structured query. A single seeding solution can yield several interpretations.

4.1.4 Decision Tree and Greedy Implementation

All three phases—chunking, seeding, and growing—involve many decisions to make, each of which has many possible solutions. The whole process can be viewed as a decision tree in which the leaves are the complete interpretations of the original text query at the root of the tree. In our prototypical implementation, this decision tree is traversed depth-first, and at each node, branches are ordered by local scores, yielding fast but greedy performance.

4.1.5 Pseudo-types

Implicit in our discussion so far is the assumption that a user's notion of a general collection of topics (e.g., "films") is modeled as a type in Freebase (/film/film). This is not always true. For example, we might expect "volcano" to correspond to a type in Freebase, but it does not. Rather, it corresponds to a kind of mountain, and "mountain" corresponds to the type /geography/mountain. Thus, "volcanoes" is translated to a structured query for topics of type /geography/mountain and having /geography/mountain/mountain_type /en/volcano. This is our concept of *pseudo-types*, which bridge the gap between the user's notion of types and the actual types in Freebase.

4.2 Unified Suggest Widget

The previous sub-section discusses the core algorithm for interpreting a collection query. In real use, the user might enter a single topic query, or, as discussed in section 3, might enter an abbreviated collection query. In order to accommodate as all three kinds of query, we need an overarching algorithm, which consists of many more phases as discussed below.

4.2.1 Unifying

Given a text query, we submit it as-is to the Freebase search service and retain some top results based on some threshold criteria. Next, we try to match the query against past collection queries that other users have issued. For example, when the user just types "french," we can get the partial matches "french authors," "french wines," etc. These partial matches both save the user from typing as well as hint the user of our novel collection query support. If all of these matches are partial, then we interpret the query using the core algorithm discussed in 4.1. The output of this phase is a list of zero or more single topic search matches, zero or more partial matches of past collection queries, and zero or more interpretations of the text query as a collection query.

4.2.2 Extending

To support collection query in an embedding scenario within a specific context, as discussed in section 3, we also need to understand abbreviated queries. The suggest widget can be configured to give hints about which types to expect from a query, and if it is abbreviated, how to generate a full query from that. For example, in a book embedding scenario, full queries should be of type /book/written_work, and abbreviated queries can be of type /book/author or /book/literary_genre. Knowing these types helps us quickly find appropriate partial matches from past queries in the unifying phase.

4.2.3 Condensing and Approving

This phase eliminates duplicate interpretations from the previous phase as well as interpretations that resolve to empty collections (due to lack of data in Freebase or in the real world).

4.2.4 Explaining

This last phase generates natural language text explaining each interpretation back to the user. For example, the text query "robert redford films" can be interpreted in three different ways, and explained back to the user as "films starring Robert Redford," "films directed by Robert Redford," and "films produced by Robert Redford." The algorithm for generating a textual explanation from the structured query of an interpretation is complicated and not yet sufficiently fleshed out to be explained here.

5. RELATED WORK

Kaufmann's doctoral thesis [5], which investigates natural language interfaces to semantic web data, is a highly related body of work. One of the systems she built, NLP-Reduce, translates text-based natural language queries into structured queries also using a pattern-matching approach like ours. NLP-Reduce is even more forgiving in that it removes even more stop words and allows for full questions or fuller question fragments. But whereas NLP-Reduce is aimed to address generic questions (e.g., "how big are the lakes in Illinois?"), our work aims to only retrieve collections of topics. Our focus allows us to make assumptions about shortcuts that users would tend to make, particularly that they would phrase queries as lists of keywords that map to filtering and pivoting operations. We believe that this is closer to how web users use existing search engines. Furthermore, we are unable to verify how well NLP-Reduce would work on a large and heterogeneous data set as Freebase. Kaufmann's systems have only been evaluated on 3 data sets, each having no more than 10 types, 20 properties, and 10,000 topics (instances). On the other hand, Freebase has 3,000 types, 30,000 properties, and almost 9 million topics. This difference in magnitude should have implications on both data processing performance as well as the effectiveness of heuristics: large, heterogeneous data sets tend to have more name collisions, and the more flexible the query is allowed to be, the more explosive the number of interpretations there are.

6. FUTURE WORK

While we contend in section 2 that some web users already formulate collection queries, it is not clear how they would react to precise collections as search results as opposed to million fuzzy keyword matches that existing search engines return. It is also not clear if suggestions from partial matches against past collection queries are enough to make web users aware of this new capability and confident to formulate new, similar collection queries themselves. If they are enough, then we can prime the past collection query index with pre-canned queries generated from some typical patterns such as "<nationality> <profession>". Finally, the Freebase Cubed suggest widget requires some UI iterations and usability testing to make sure that users understand the difference between single topic suggestions and collection suggestions. Those are some of the research tasks to be done next.

7. REFERENCES

[1] Google Insights. http://www.google.com/insights/search/.

[2] Google Suggest API.

[3] Berners-Lee, T., Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets. Tabulator: Exploring and Analyzing Linked Data on the Semantic Web. ISWC, 2006.

[4] Huynh, D., S. Mazzocchi, and D. Karger. Piggy Bank: Experiencing the Semantic Web inside Your Web Browser. ISWC 2005.

[5] Kaufmann, E. Talking to the Semantic Web? Natural Language Query Interfaces for Casual End-users. Doctoral thesis, 2008.