When You're Lost for Words: Faceted Search with Autocompletion

Holger Bast
Max-Planck-Institut für Informatik
Saarbrücken, Germany
bast@mpi-inf.mpg.de

Ingmar Weber
Max-Planck-Institut für Informatik
Saarbrücken, Germany
iweber@mpi-inf.mpg.de

ABSTRACT

In this paper, we show how the autocompletion data structure of [2] can be used to answer faceted-search queries efficiently. Specifically, we have built a fully-functional browser-based search engine that can index collections with arbitrary category information and that, *after each keystroke from the user, computes and displays the following* information: (i) words or phrases that begin with the last query word and would lead to good hits; (ii) the most relevant categories for those hits; (iii) any category names that match the query as typed so far; (iv) the most relevant hits for the query as typed so far. By appropriately rewriting the faceted-search queries as autocompletion queries according to [2], we obtain very fast query processing times, improving those obtained by standard approaches by an order of magnitude. On 11,685 scientific articles from the DBLP collection, with their full text and categorized by author, conference, and year, the average query processing time is about 25 milliseconds, on a single machine and with the index on disk. For the 2,172,832 articles of the latest dump of the English Wikipedia, with their full text and categorized by Wikipedia's own category labels, we achieve an average query processing time of about 350 milliseconds.

Categories and Subject Descriptors

H.5.2 [**User Interfaces**]: Interaction Styles, Prototyping, Theory and Methods; H.1.2 [**User/Machine Systems**]: Human factors; H.3.3 [**Information Search and Retrieval**]: Information Filtering, Query Formulation, Search Process, Selection Process

General Terms

Algorithms, Design, Human Factors, Performance

Keywords

Faceted Search, Autocompletion Search, Text Database Exploration

SIGIR'06 Workshop on Faceted Search, August 10, 2006, Seattle, Washington, USA.

1. INTRODUCTION

When it comes to finding documents in large collections, the 1-box web search interface approach seems to be predominant. On the other hand, in settings where the focus is more on data exploration than on information retrieval, a hierarchical organization of the documents is often used, allowing the user to browse and drill down into subcategories. Online shopping sites nowadays present the user with both options at the same time: the option of entering queries as usual while also allowing the user to refine his search by drilling down into the matching categories. By pro-actively supporting this kind of query refinement, the user needs to know less about the structure and the items in the database and is spared the experience of over-specifying a request using an "advanced search" form.

In [2] we presented an efficient algorithm which allows applying the same ideas to the query formulation process. Namely, while the user is typing a query, after every keystroke we present a list of completions leading to a hit for the prefix typed so far, along with a list of the matching documents. Besides the obvious advantage that the user has to type less, it also spares him the frustrating experience of over-specifying a query and then being presented with no relevant hits.

In this paper, we demonstrate how our algorithm from [2] can be directly applied to obtain both features: pro-active support for both the query formulation (by presenting relevant completions) and for the query refinement through categories (by presenting matching categories). We have built and tested a prototype with these capabilities and verified its practicability in terms of efficiency by experiments with a collection of scientific articles and with the English Wikipedia. Figure 1 provides a screenshot of our search engine in action and gives an explanation of its main features. See `http://search.mpi-inf.mpg.de/faceted` for a live demo.

2. RELATED WORK

The number of websites which offer some kind of faceted search is enormous and spans all information domains.

If one loosely defines a faceted search interface as one which, in addition to showing ranked results for keyword queries as usual, also organizes query results by categories, then nowadays almost every online shopping portal offers a faceted search interface[1].

The archetypical, fairly intuitive user interface shows a

[1]http://www.ebay.com, http://www.amazon.com

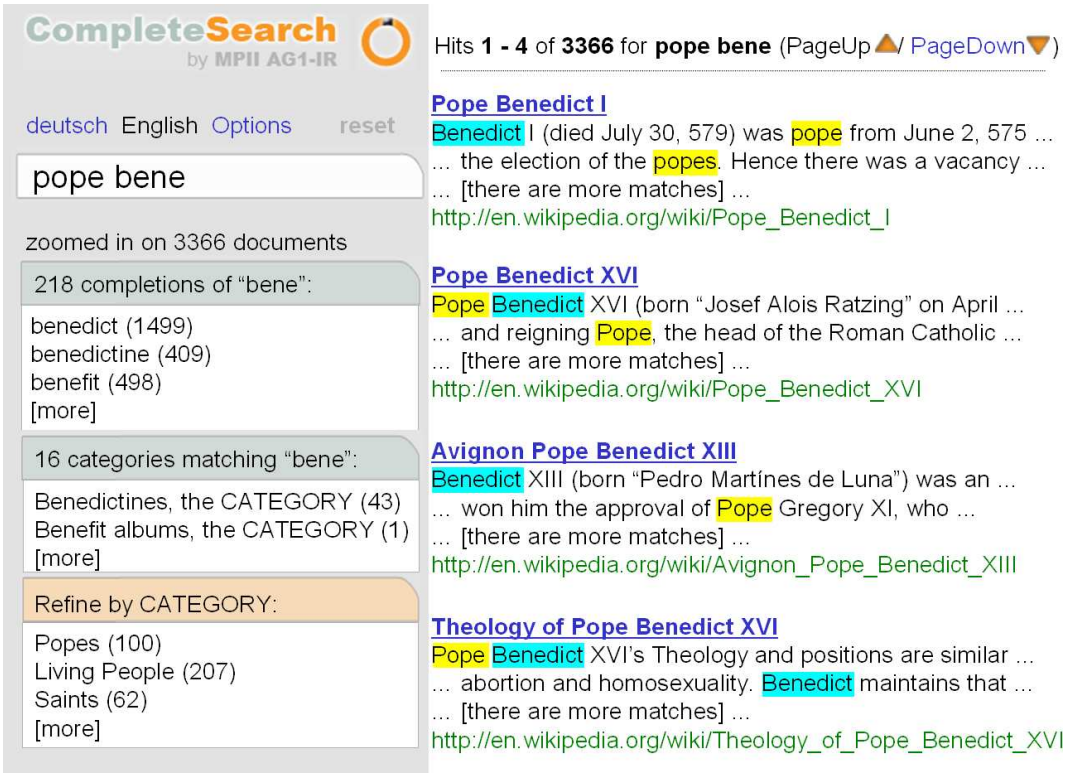

Figure 1: A screenshot of our search engine for the query `probab lat`, searching a subset of the DBLP collection. All the information displayed is automatically and instantly updated after each keystroke, hence the absence of any kind of search button. The number 1289 is the number of documents that contain a word starting with `probab` as well as a word starting with `lat`. The first line below the input field (Complete to: ...) shows words starting with `lat` that occur in documents that also contain a word starting with `probab`; this includes ordinary words like "latency", phrases like "latent semantic", and (sub)category names like "Charles Lattimer". Each of the words can be clicked on, in which case it will replace the current last word `lat` in the query. The other lists show the most relevant categories for the mentioned 1289 hits. For example, there are 40 SIGIR papers containing a word starting with `probab` as well as a word starting with `lat`. Again, each of these words can be clicked on, in which case the hits will be restricted (refined) to the respective category.

search box on top, a subset of the most relevant results below and a list of matching categories (usually organized in a hierarchical manner) on the left. These matching categories can then be used for further refining the query. Variations of this interface include extensions to search in the category names themselves. See, e.g., the demos of the Flamenco Project[2]. These type of interfaces are in their simpler form also used in other domains such as medical databases[3] (provided by Recommind), news archives[4] (provided by Endeca) and tagged webpages[5] (provided by RawSugar). Other systems, such as Facetmap, provide no search facility but only allow faceted browsing[6].

A number of content providers support the user in finding relevant terms for his query. Some sites[7] use a precompiled list of plausible queries similar to Google Suggest[8], from which they display completions to the user. These lists are typically only useful if one is looking for "mainstream" information and for spelling suggestions. Such interfaces are usually referred to by the term "live search" (see [6] for a discussion and more related links). In the domain of library search, the AquaBrowser interface[9] uses data mining, machine translation and spelling corrections to find other related query terms on-the-fly. These are then presented to the user in a star-like graph. However, these terms seem to be often (i) quite general and (ii) unrelated. For the query `information retrieval`, broad and partly irrelevant terms such as "service", "technology" and "freedom" are displayed as associations, along with the supposed spelling variant "Euro-travel" and the supposed translation "enlightenment". Following a fully term-centered approach, it is also plausible to disregard the hierarchical structures for the documents altogether and rather focus on organizing the terms

[2]http://flamenco.berkeley.edu
[3]http://www.medlineplus.gov
[4]http://browse.guardian.co.uk
[5]http://www.rawsugar.com
[6]http://www.facetmap.com
[7]http://fastsearch.com/search.aspx, http://kayak.com

[8]http://www.google.com/webhp?complete=1
[9]http://aqua.queenslibrary.org

in the collection. Such a taxonomy can then be used to guide the user through the query formulation process [3].

For the case of a well-controlled database, whose structural data integrity can be ensured, a query language is presented in [7], which allows answering faceted database queries in time quadratic in the number of items with linear space complexity. The problem of faceted search can also be viewed more generally as a problem of multidimensional visualization and navigation. To show the relation between two independent dimensions the use of 2-D heat maps was proposed in [1] and their benefit is demonstrated by a small user study. Similar ideas are explored in [8]. Larger user studies to demonstrate the advantages of the Flamenco faceted search user interface have also been conducted in [4, 9].

In scenarios where hierarchical structures or any kind of categorization are not a priori given, it is still possible to apply the paradigm of organizing the result set in a structured way by clustering the results on-the-fly. In [5] the relative (dis-)advantages of result clustering and faceted categories are compared. Besides the lack of quality of the resulting clusters (meaning that clusters can be very heterogeneous), even for state-of-the-art systems[10], other disadvantages include the lack of predictability (a user does not know in advance how his results will be organized) and the diverse mix of the obtained subcluster hierarchies (many facets get mixed when a cluster is broken down).

In our work, we completely factor out the issue of how (hierarchical) labeled categories can be obtained. If one hopes to accumulate a large set of documents, manually labeled, the only viable approach is to have a dedicated community of people all contributing to this project. Such a contribution can be made by manually inserting a web document into a taxonomy, as in the case of the open directory project[11], or by sharing (organized) bookmarks and assigning short tags to a currently visited web site[12]. The latter approach is often referred to as "social tagging" with the resulting structures being nick-named "folksonomies"[13].

3. AUTOCOMPLETION SEARCH

In this section, we summarize the work presented in [2], where we give an efficient indexing data structure and algorithm for the following autocompletion search problem:

DEFINITION 1. *An autocompletion query is a pair (D, W), where W is a range of words (all possible completions of the last word which the user has started typing) and D is a set of documents (the hits for the preceding part of the query). To process the query means to compute the subset $W' \subseteq W$ of words that occur in at least one document from D, as well as the subset $D' \subseteq D$ of documents that contain at least one of these words.*

As an example, suppose a user is searching the Wikipedia for information about the current pope. However, either he has forgotten the exact name or he is unsure about spelling variants ("benedict", "benedikt", "benedictus", ...). He starts typing the query `pope bene` and is then presented with a list of matching documents, as well as a list of matching completions, which occur in documents also containing the term "pope". He now has the option of (a) refining his search further by continuing to type his query or by selecting a particular spelling variant, or (b) going through the result list for the broader query as it is. Figure 1 shows a user interface which provides this feature along with a number of extensions.

It is not difficult to see how standard inverted lists can be used to obtain this information: simply iterate over all words w in W and compute an intersection of the list of documents for w with the document list D. Report all words w with non-empty intersection (i.e., they occur in at least one of the matching documents for the preceding part) as the set W' and report the union of all such intersections as the set D' (i.e., all documents which contain at least one completion in W'). Drawbacks of this approach are discussed in [2].

For the autocompletion search problem, the same paper presents a new data structure and algorithm called HYB, which is faster both in theory and in practice, while not using more space than a compressed inverted index.

The basic idea of the data structure is to precompute inverted lists for fixed *word ranges*, rather than for individual words. If the word ranges are chosen appropriately, this allows (a) not having to merge any document result sets and (b) reducing the total number of intersections per query to 1. Exactly one random disk access is required in this process.

4. FACETED SEARCH WITH AUTOCOMPLETION

The focus of our work in this paper is on showing how the algorithm HYB for the "ordinary" autocompletion search problem (see Definition 1), can be extended to include faceted search capabilities. Specifically, we show how the following three features can be obtained:

- a display of matching completions for query terms (occurring in at least one hit) and their use for query refinement,
- a display of matching categories (containing at least one hit) and their use for query refinement, and
- a display of matching category names (starting with the prefix and containing at least one hit for the remaining query) and their use for query reformulation.

Each of these features will be discussed in the following three subsections.

4.1 Finding Matching Completions

The problem as defined in Definition 1 is at the heart of our engine.

In practice, we further enrich the completion capabilities by extending the definition of a "completion" to include common word combinations or phrases (e.g., "huffman encoding" or "latent semantic indexing"). See the "Complete to" field in Figure 1 for an illustration of this procedure. Such combinations are extracted before the index is built and artificial words such as "huffman_encoding" or "latent_semantic_indexing" are then added to the documents. Through these words, the phrases suddenly become ordinary words, at least from the the point of view of any completion

[10] http://www.vivisimo.com
[11] http://www.dmoz.org
[12] http://www.rawsugar.com, http://www.flickr.com, http://del.icio.us
[13] http://en.wikipedia.org/wiki/Folksonomy

algorithm. How such combinations can be found is orthogonal to our work here. But in practice even a simple heuristics often gives satisfactory results: If both the combined word "latent_semantic" or "latentsemantic" (without a space as a separator) as well as the adjacent word pair "latent semantic" appear in the database it is probably a common phrase.

Given such an enriched set of documents, the data structure HYB is built and used as usual.

Usually, it is desirable to present only the most promising completions to the user, e.g., ranked by the number of documents in which they occur. Ranking of completions and documents is also discussed in [2] and is part of our prototype.

Note that since we consider every prefix as a valid query term, the need for a `search` button disappears (!) and no such item can be seen in Figure 1. To be more concrete, once the user has typed a prefix of a certain minimal length (3 letters), a query is automatically executed once the user stops typing for 500 milliseconds. This idea of starting to search automatically, without the explicit command by the user, is also used by some online search interfaces. For example, the "Live Search" on the FAST homepage[14] has this feature and avoids having a search button altogether. Similarly, the desktop search tool Copernic[15] also starts the query process automatically, but still treats itself to the unnecessary search button.

4.2 Finding Categories Containing Matches

To be able to find and display matching categories as well, we simply add the information about categories to our index by inserting an artificial term, e.g., `cat:living_people`, into a Wikipedia article about a living person before our index structure is built. The same is done for other categories and documents. The colon `:` serves to distinguish artifical words from ordinary words.

To process a query, e.g., `pope bene`, we first run the usual autocompletion search query as outlined above and explained in detail in [2]. This gives the set of matching documents in form of a ranked list. Then, in a second step, we run the query `pope bene cat:` *as a regular autocompletion search query*. Due to the way we inserted the artificial terms, the completions for the term `cat:` will now exactly correspond to the list of categories containing a matching document for the query `pope bene`.

When these matching categories are presented to the user, he then has the option of refining his search by limiting the search scope to matches within a chosen category. In our example, he might choose the Wikipedia category "Popes". Figure 1 shows matching categories listed as "Refine by" options. Here the name of an author and a year of publication are also considered categories.

In our user interface, the category selection is done by simply clicking on a relevant category. This follows the standard conventions for providing such a feature and is implemented in a similar manner in all major search interfaces for faceted search[16].

Observe that selecting a particular category is, from the point of view of our algorithm, the same as adding an additional query term, where the query term in this case is an artificial term.

In scenarios where the total number of matching categories can be very large, as is the case for the Wikipedia collection, we only present the most relevant ones to the user. Again, for this we can employ the very same ranking mechanism which we already used to select the most promising completions.

To ensure efficient query processing, all artificial terms of the form `cat:*` should be in the same block for HYB. That is, our data structure is built such that we have the sorted list of documents containing at least one of these special words precomputed. Given such a choice of blocks, we can then answer a query of the form `pope bene cat:` above with a single list intersection.

4.3 Finding Matching Category Names

In some cases, the user intention for the query `pope bene` might be of a different nature: Maybe when typing such a query the user is looking for *categories* starting with `bene` and containing a document with the term `pope`. For the Wikipedia such matching categories would include "Benedictines" or even "Benefit_albums". The "Complete to" field in Figure 1 contains an example where a particular author is suggested as a matching category.

Using exactly the same setup as before with the same artificial words included in the index, such matching completions can be found by answering the autocompletion query `pope cat:bene`.

Again, the user can now choose from any of these matching categories. But, different from before, a selection of a particular category does not correspond to a query refinement with respect to the base query `pope bene`, as now one query word (`bene`) gets *replaced* by another (`cat:bene`) whereas before a query word was *added*.

The selection of such a desired category is again done by clicking on the name of the category in our user interface. This feature is less common but, if present, is usually provided via the same mechanism, as in the Flamenco System[17].

If the blocks for HYB are chosen optimally as for the feature above, then we can again answer a query of the form `pope cat:bene` above with a single list intersection and avoid any merge operations. In practice, these types of query will be easier to answer than the query types of the preceding subsection, as the relevant word range (all category names starting with `bene`) is narrower than before (where *all* category names had to be considered).

5. IMPLEMENTATION AND EXPERIMENTS

We built a fully functional prototype for faceted search with autocompletion. It works over the internet with any standard web browser that supports JavaScript. The query processor is written in C++ and runs on a Opteron dual 2.4 GHz processor machine with 8 GB of RAM, with the (compressed) index on disk; see [2] for more details. In our experiments, we measured the time for processing the derived autocompletion queries, as described in Sections 4.1, 4.2, and 4.3. We did *not* measure the time for the transmission of the query and the results over the network.

5.1 Collections and Queries

[14] http://fastsearch.com/search.aspx

[15] http://www.copernic.com

[16] http://www.rawsugar.com, http://flamenco.berkeley.edu, http://www.ebay.com, http://www.amazon.com, ...

[17] http://flamenco.berkeley.edu

Our first data set, DBLP, consists of 11,685 scientific articles listed in DBLP[18], both the full text and the DBLP meta data, including information about authors, conferences and year of publication.

Our second data set, WIKIPEDIA, consists of the full set of 2,172,832 articles of the May 2006 dump of the English Wikipedia[19]. As meta data for this collection, we took the Category information which the Wikipedia articles themselves provide (at the bottom). This information is much more diverse than for DBLP: some articles carry a dozen of different category labels, about half of the articles are listed under no category at all.

For both collections we generated 500 standard keyword queries with a realistic distribution of query length (short queries are more common) and keyword selection (content-bearing words are more common than very frequent words). Each such query was then "typed" letter by letter (beginning with a 3-letter prefix for each query word), resulting in a chain of autocompletion queries. For example, the keyword query `pope benedict` gives rise to the 8 autocompletion queries `pop`, `pope`, `pope ben`, `pope bene`, ..., `pope benedict`. Like this we obtained 6024 autocompletion queries for DBLP, and 5320 autocompletion queries for WIKIPEDIA.

From each such autocompletion query, then three queries were derived: the query itself (to find relevant documents and word completions as described in Section 4.1), the query with the query word prefix `cat:` added at the end (to find matching categories as described in Section 4.2), and the query with the last query word prefixed by `cat:` (to find matching category names, as described in Section 4.3).

5.2 Results

Table 5.2 shows the average running times per query on both DBLP and WIKIPEDIA, with a breakdown for the three sub-queries described above. Three main observations are to be made. First, for both collections, queries are processed in a fraction of a second, which yields the desired interactive behavior. Second, the query processing time is dominated by the second type of subqueries. This is easy to understand, since for the second type of query, we have to screen each matching document for its category labels. This is a work-intensive task, but as explained in Section 3, this is exactly the kind of queries where the HYB data structure from [2] shines, and improves over the standard inverted-index based approach by an order of magnitude (see [2] for details). Third, the last type of sub-queries hardly takes any processing time, since for most autocompletion queries, the last query word does not match any category name at all (in which case the cost for this sub-query is zero).

Collection	DBLP	WIKIPEDIA
Average time	24 millisecs	341 millisecs
- ordinary	6 millisecs	53 millisecs
- categories	17 millisecs	274 millisecs
- cat.names	1 millisecs	14 millisecs

Table 1: Average running times for the faceted autocompletion queries on DBLP and WIKIPEDIA, with a breakdown with respect to the three subqueries discussed in Section 4.

6. FUTURE WORK

The user interface could be improved in a number of ways. For example, categories with a hierarchical order should be displayed in a tree-like fashion and not as a flat list, as we currently do. Another nice-to-have feature would be a button to remove query words that have been added by a refine-by-category operation. So far we concentrated on providing certain new features (combining autocompletion search with faceted search), and showing that we can support them very efficiently. It would be interesting to conduct a user study to verify the usefulness of the set of features we have presented.

[18]http://dblp.uni-trier.de
[19]http://en.wikipedia.org

7. REFERENCES

[1] W. A. Arentz and A. Øhrn. Multidimensional visualization and navigation in search results. *Lecture Notes in Computer Science*, 3212:620–629, 2004.

[2] H. Bast and I. Weber. Type less, find more: Fast autocompletion search with a succinct index. In *SIGIR '06: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, to appear, 2006. http://www.mpi-inf.mpg.de/~bast/publications.html.

[3] C. Binding and D. Tudhope. KOS at your service: Programmatic access to knowledge organisation systems. *Journal of Digital Information*, 4(4), 2004. http://rapid.isd.glam.ac.uk/FACET/.

[4] J. English, M. Hearst, R. Sinha, K. Swearingen, and K.-P. Yee. Hierarchical faceted metadata in site search interfaces. In *CHI '02: extended abstracts on Human factors in computing systems*, pages 628–639, 2002.

[5] M. A. Hearst. Clustering versus faceted categories for information exploration. *Communications of the ACM*, 49(4):59–61, 2006.

[6] J. Rønn-Jensen. Live search explained. http://justaddwater.dk/2006/01/26/live-search-explained/.

[7] K. A. Ross and A. Janevski. Querying faceted databases. *Lecture Notes in Computer Science*, 3372:199–218, 2005.

[8] B. Shneiderman, D. Feldman, A. Rose, and X. F. Grau. Visualizing digital library search results with categorical and hierarchical axes. In *DL '00: Proceedings of the fifth ACM conference on Digital libraries*, pages 57–66, 2000.

[9] K.-P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408, 2003.