

Improved Wikipedia Search Experience

Aspasia Beneti, Jiří Iša, Markos Mylonakis, Günther Starnberger, Jeroen Stegink, Lars Wortel
University of Amsterdam

Abstract—

WIKIPEDIA, the free encyclopedia, is quickly increasing its influence on the Internet. More and more people link to it for term definitions, historical event descriptions and even the news. At the same time the content of the encyclopedia grows at a rapid rate. Together with the increasing amount of available information it is becoming difficult to actually find the content of interest. The internal search engine is down very often, suggesting the users to use Google or Altavista instead. And they do. And they will do it next time as well, because the results “feel better”. Is there a way to build a search engine which could use specific features of Wikipedia to improve the users’ satisfaction?

I. WEB DOCUMENT CLUSTER COHERENCE EVALUATION FOR LARGE SEMI-STRUCTURED WEB DOCUMENTS COLLECTIONS

A. Introduction

LARGE volumes of semi-structured web data are difficult to manage and explore. The English Wikipedia pages [1] [12] are one example. This is a collection of over one million web documents (encyclopedia articles), each exploring a knowledge topic. There is an on-going effort, *Wikipedia categories*, to manually cluster together documents that are related to each other and thus provide this web collection with a degree of structure [6]. These categories usually group together articles in two ways.

- They group articles that fall in the same semantic context, for example by sharing a historical or geographical context, or belonging to the same knowledge field. For the article ‘*Albert Einstein*’, examples of this type of category are ‘*german physicists*’, ‘*humanitarians*’ or ‘*contributors to general relativity*’.
- They group articles that share a common property, which is not largely related to their semantical context. Examples for the same article are ‘*articles with unsourced statements*’, ‘*patent clerks*’ and ‘*1879 births*’.

What is particularly special about Wikipedia categories is that they are freely edited by anyone who would like to contribute to the categorization effort. As views on what can be grouped together vary, the results is a sometimes chaotic categorization scheme [13], that hinders the efficient use of it. For example, in a relevance feedback query system for Wikipedia’s article collection, in order to be able to present the user a list of categories where relevant articles belong to semantically refine his search, we need to select categories that semantically describe these articles. While Albert Einstein was both a patent clerk and born in 1879, displaying the relevant categories to the user makes no sense. What the user would like to be shown is categories that are more informative about the article, i.e. share more information content with it.

A document cluster’s information content is formed by (1) the documents that fall under it and (2) it’s title. To evaluate the information content overlap between a document and one of the clusters that it belongs we need:

- A way to assess the semantic overlap of documents belonging to a cluster. We refer to this in the rest of the article as *semantic coherence*.
- A way to assess the semantic overlap of a particular document with the category’s title.

Using English Wikipedia as a testbed, we propose methods that try to address both problems. In the following section we present a general use method for measuring web document cluster coherence using link structure and implement it for Wikipedia categories. In section I-C, we describe a method for evaluating the relatedness of a category with an article by using the category’s title using information extraction techniques. In section I-D, we explore a solution for efficient clustering of categories based on their titles. Finally, we present the results of our evaluation for the fore mentioned methods in section I-E and our conclusions and future work in section I-G.

B. Cluster Coherence Metrics

In this section, we introduce a method for using web document collection link structure to assess the semantic coherence of clusters of these documents. Our aim is to measure the amount of semantical information overlap between documents of a web cluster. To address this problem, we employ web link structure. Our main assumption is that co-citation of web documents through linking implies semantical relatedness. We explore web link structure in three levels, assuming semantic relatedness of documents of a cluster when:

- *Inward link structure*: A document not belonging to a cluster links to two or more documents of the cluster
- *Outward link structure*: A document not belonging to a cluster is linked by two or more documents of the cluster
- *Internal link structure*: A document belonging to the cluster links to one or more documents of the same cluster

We now propose a method to measure semantic coherence of web document clusters using link structure in these three levels. An important part of the method is the use of *binary combinations of links* to quantify the impact of simultaneous co-citations. In this fashion, we formulate three metrics, each exploring one of the fore mentioned three levels of link structure. A formal description of the metrics is included in Appendix A.

We used our method to evaluate the semantical coherence of Wikipedia’s categories. The link data examined exceeded 37 million links between articles of Wikipedia and the link structure follows similar patters to the rest of the Web [4].

In order to measure the semantical coherence of each of the more than 90 thousands categories we had to solve two issues, which can be relevant to other web document collections as well.

Firstly, when a document links many articles belonging to a particular category, this does not always imply high semantic relatedness. This is particularly true for articles that are merely lists of links to other articles, for example a *list of people born in 1879*. In order to avoid this, we did not take into account links from or to a particular article, when their number exceeded what could be considered ‘normal’. For each category and for each metric, we used the histogram of the number of documents that have a specific number of relevant (for the metric) links. When this histogram leveled to zero for many consecutive columns, we set up a cut-off point and considered pages with an even greater number of links as ‘untrustworthy’.

In addition, the evaluation of the coherence metrics for large categories takes considerable time. While it is possible to directly evaluate the metrics, we implemented an approximative technique using sampling. For each category exceeding a threshold n of number of articles belonging to it, we randomly sample n articles from it and use this subset to calculate the coherence metrics. We found out that this method, when n is not very low (above 500), outputs results not far from the ones deriving from exact computations.

We found the three metrics to often complete each other. When there was poor or no evidence to reliably evaluate one, the others could be used to assess coherence. We used the *harmonic mean* of the three semantic coherence metrics to combine them in a single measure. The fact that harmonic mean eliminated efficiently outliers and the results of empirical experiments led us to this decision. The results of using this method with the Wikipedia categories were quite promising and are presented in section I-E.

C. Definition extraction

A documents semantical relation with an assigned named cluster can also be estimated by using the cluster’s title. Any relevant technique can be used for this task [10] [5]. In the case of Wikipedia, by convention, encyclopedic articles begin with a sentence that defines the entity which is the theme of the article. The definition usually follows a form of the verb *to be* [11]. Extracting the noun phrase of the definition of each article is an information extraction task that can provide us with useful semantical information. In order to extract the noun phrase of a sentence, one has to parse it first. However parsing over 1 million sentences of arbitrary length is computationally expensive and time consuming. To overcome this difficulty, we introduced a process of reducing the sentence to parse to an *equivalent* for our purposes sentence of smaller length. Thus, the definition extraction process works in 3 steps:

- 1) Wikipedia markup and additional tags are removed. The first sentence of each article is extracted.
- 2) Each sentence is reduced to an equivalent which retains the noun phrase to be extracted but is smaller in terms of length. For this purpose several empirical rules have been applied after observing a large amount of articles.

- 3) The reduced in size sentence is parsed using the Stanford parser [8]. From the parse tree the noun phrase and the noun next to the form of the verb *to be* are extracted.

For example the first sentence extracted from the article with page title: ‘Street luge’ is: *Street luge is an extreme gravity-powered activity that involves riding a street luge board (sometimes referred to as a sled) down a paved road or course*. The reduced in size sentence that our system produced is: *Street luge is an extreme gravity-powered activity*. Finally the noun phrase and the noun extracted are *extreme gravity-powered activity* and *activity* respectively.

The definitions extracted from every article are used to assess the semantic overlap with a category’s title and select the single most semantically descriptive category for an article. For every article at query time we retrieve its definition and the list of its categories ranked according to the coherence metrics harmonic mean. Then, for each of the categories, starting from the first in order and keep going down the list, we compare its name with the noun and the noun phrase and we generate a resemblance metric. In order to improve the results of the comparisons, we made use of *Porter’s algorithm* for stemming. Categories with higher resemblance metrics have priority to be selected. Whereas we have not implemented it yet, it seems natural to use a method similar to the one employed in section I-D to enhance the results of the resemblance metric.

A last thing to note here is that it is not always possible to extract the definition. A common reason for this is that in many articles the verb ‘to be’ is absent from the first sentence and thus there is no clear pattern to use to detect the definition. To avoid extracting wrong definitions that might influence negatively the procedure of the category selection, the software is designed in such a way that it prefers not to select a category at all, than to select the wrong one. In other words, the system sacrifices part of its recall in order to achieve very high precision.

D. Category Title Clustering Using WordNet

It is often the case that it is essential to group similar documents based on their titles. This could be done by matching words that appear in more than one title. However, if the amount of documents is not large, this process is not feasible because there will not be found a lot of matches between the words and, of course, semantic relatedness extends past common words. One way to solve this problem is instead of comparing only the words themselves, to take into account all their possible synonyms. A well known ontology of the English language, WordNet [9], makes available a big collection of words with their synonym trees and it is the perfect tool for this purpose [7].

In the context of Wikipedia and in the context of providing the user with relevant categories given a list of articles, we decided to make use of such a method. We wanted to cluster together similar categories and give to the user a clearer and more easily explorable category list. The method works as follows. For each of the categories that have to be clustered, we first split its title into words and for each word we get all the possible synonyms and hypernyms of one level up the

WordNet semantic tree. This results in a list of semantically related words for each category. However, in order to perform comparisons we should have a list of categories for each synonym. This is achieved by making use of the *index inversion process*. Then, one can start grouping together categories based on the same synonyms. Different decisions can be made about the exact grouping scheme depending on the specific case. For example it is often useful to make a list of stop-words such as ‘of’, which appear very frequently and should not be used for grouping as they do not reveal any common semantics. Another decision that could be made at this point is which level of granularity (number of documents per group) is adequate and acceptable, for which a threshold can be set. Primary work on this method gave us very promising results and proved that semantic dictionaries as WordNet are very helpful in this context.

E. Evaluation

F. Category Coherence Metrics

We evaluated the performance of our methods by measuring its efficiency to solve a practical problem. The problem was to select from a (frequently very long) list of categories that an article belongs one that has a large semantic overlap with it. The evaluation used 152 articles in total. It was tested when the system was in its final version, taking into account the article definitions and also linking each article to the parents of its original categories in the categorization scheme, to allow a more abstract semantic representation. A human assessor decided at first which are the most descriptive categories for each of the articles. Then the system returned the category with the highest measure and the precision at one was calculated.

	Category selection	Random guess
--	--------------------	--------------

Precision @ 1:	0.6842	0.2522
----------------	--------	--------

1) *Definition Extraction*: We also evaluated the process of definition extraction on its own. The evaluation was performed using 150 articles. As it was desirable, the system achieves precision greater than 95 percent.

	Noun phrase	Noun
Recall:	0.626	0.626
Precision:	0.978	0.957

G. Conclusions And Future Work

Whereas the semantical coherence metrics for web document clusters that we defined already work quite well, there is still room for improvement. One particular aspect that we have noticed is that, in practice, they penalize clusters with many documents. For example, the inward links coherence metric normalizes using the binary number of links we would measure if all the pages linking to documents from the cluster actually link to all of them. From a practical point of view, in a real web document collection, it is less probable that a cluster of 1000 documents will have all of its documents linked by a single page than one with 10. The same concept applies to the other two metrics as well. We consider as future work to remove this bias against large clusters without sacrificing the intuitive way that our metrics work.

Although we tested the effectiveness of our method on Wikipedia categories, we made no assumptions related to this particular semi-structured web document collection. We believe that the semantic coherence mechanism introduced in this document is generic and we would like to evaluate this by applying it to other web documents collections. Finally, we introduced semantic coherence metrics for already defined clusters. It seems natural to also try to employ these measures to actually form semantically coherent clusters from web data, by trying to maximize the coherence metrics for each cluster.

II. INDEXING & SEARCHING

A. Search Backend

OUR search engine is build upon two main parts:

- An indexer which reads the data from a Wikipedia dump and stores it into an inverted index
- A search engine which makes this data available to the search-frontend

Both parts are build upon the Lucene Information Retrieval library ¹.

B. Indexer

The indexer is responsible for the extraction of Wikipedia articles from a Wikipedia-database dump and for the creation of the inverted index. Initially Jython² was only intended for usage in the first prototype. As the performance of the Jython prototype was very satisfactorily we decided to also base the final version upon Jython. For the same reason we did not implement any optimizations apart from the features which are already build into Lucene.

1) *Input parsing*: The Wikipedia project provides regular dumps of the article database. While the dumps are formatted in XML, the individual articles are formatted using the proprietary MediaWiki markup language. We have written a simple SAX parser which extracts the articles from the dump and which adds the content of the articles as well as the content of the individual sections of each article to an inverted index. We use regular expressions to remove the MediaWiki markup and to convert the articles into plain ASCII text.

2) *#REDIRECT handling*: Wikipedia makes extensive use of the MediaWiki #REDIRECT command to declare alternative titles for an article. An article containing the #REDIRECT command is a normal article by itself, but when a user opens the article he is redirected to the destination page specified in the command.

An example is the article called *Amsterdam*. Several other articles which are synonyms for *Amsterdam* use the #REDIRECT command to forward users to the *Amsterdam* article. Two of this alternative articles are *Amsterdam*, *Holland* and *NLAMS*.

¹<http://lucene.apache.org>

²Jython is a Python version which is written in 100% Java and runs atop the JVM. It allows the usage of Java libraries and rapid prototyping of Java applications.

The current search-engine which is used by Wikipedia handles redirects rather poor. They are handled like every other article. Firstly this results in a bad score as the alternative title is not combined with the content for the scoring³. No snippet is displayed on matching redirects. Further a user might receive several redirects to the same page in the search results.

We handle redirects in a special way to overcome this problem. In a first pass we scan the Wikipedia dump for redirect information and store the data into a MySQL database. In the second pass—when we do the actual indexing—we check for each article if alternative titles are stored in the database. If this is the case we store these titles together with the article. The two-pass handling of the input is required as Lucene does not allow adding terms to documents once they are part of the index.

For reasons which are discussed in the `searcher` section of this document we cannot use a multi-valued field for the alternative titles. Instead we store the main title in a field called `title` and each optional title in a field called `title-N` where N is an Integer equal or greater two which indicates the number of the title.

3) *Categories*: Each Wikipedia article may belong to a number of categories. The indexer uses category information which is preprocessed by our Category-Team. For each article which is added to the index the relevant categories are retrieved from MySQL database. The name of the categories is then stored in a multi-valued keyword field.

4) *Sections*: In order to provide more precise results we not only index whole articles but also individual sections. We use a regular expression to split each processed article into its sections. Afterwards we add the article as well as the sections to the inverted index. The `type` field indicates whether the stored document is an article or a section.

C. Searcher

The search backend receives queries from the frontend, processes them using the inverted index and sends the results back to the frontend. The backend is implemented as standalone server. The communication between the backend and the frontend uses XML-RPC as communication protocol⁴.

1) *Workflow*: The workflow of the searcher consists of two main steps:

- At first a search is done on all the articles in the index which match the query.
- In a second step the relevant sections which belong to the articles in the first result-set are retrieved.

After a new search request arrives Lucene constructs a filter which basically restricts the search-range on the set of articles in the index. A Boolean-Query is constructed which includes sub-queries for searching both the title and the content of the article for the query.

Afterwards a second search on the stored sections is done. To narrow the search-range we use a Lucene filter which matches on all sections which belong to an article in the first result-set.

³because the content consists only of a single `#REDIRECT` statement

⁴for an exact description of the used XML-RPC API see Appendix B

After both the articles and the sections are retrieved the backend converts the article-set in a Java structure build upon Vectors and Hashtables. This allows the XML-RPC library to automatically serialize the data to the equivalent XML-RPC structure. During the conversation a further Hashtable is build, where the key consists of the Wikipedia-ID of an article and the content consists of a pointer to the article in the first structure. During the iteration over the articles we also generate a small snippet which gives the user of the search-engine a short information about the matching document.

The next step is an iteration over the set of matching sections. The first part is to check if the title of a section is common. Examples of common section names are: *See also*, *External links*. We can do this by directly extracting the absolute keyword frequency from the index and dividing it by the number of stored documents. If the results exceeds a certain threshold we ignore the section and skip to the next section. Otherwise we make a lookup on our Wikipedia-ID Hashtable and assign the subsection to its article by storing it in a Vector which is part of the article structure.

To be sure we do not miss a relevant subsection in a large irrelevant document the default number of retrieved documents is set rather high (60).

2) *Multiple titles*: A stored article may contain a set of optional title. While Lucene allows the usage of multi-valued fields, the technical implementation of this feature does not allow us to use it for the alternative titles. Lucene internally implements multi-valued fields by concatenating the values of all individual fields together. When a search is done on such a field it is effectively treated as one large string.

Instead of only providing a better score when a query matches an optional title, the score is actually worse⁵ if the query matches the main title. The reason is that because of the concatenation of the titles a query which matches one of the titles, does not match the optional titles and therefore the resulting score gets lower.

Our first approach to the solution of this problem was to override several of Lucene's classes which are used for the scoring. During the implementation a hint on the Lucene mailing-list pointed us to an easier solution, which we finally implemented.

The `DisjunctionMaxQuery` class returns the maximal score of several issued queries. To be able to differentiate between the individual titles we need to store each of the titles into its own field. The field-name of the main-title is `title` while each of the alternative titles is stored in a field called `title-N` where N is a Integer equal or greater two which indicates the number of the title. Initial concerns that the large number of sub-queries might have a negative impact on the overall performance proved to be wrong. We were not able to measure any considerable performance penalties.

D. Regular titles

Because we are dealing with encyclopedia, the desired outcome would be an article with the title of the given query. Ideally there is an entry just for what the user is looking for.

⁵than without multi-valued fields

For this reason we boost the words in the title by a higher value than the rest of the document. We are not explicitly dealing with other elements of the markup.

E. Discarded ideas

1) *Graph-based Visualization*: One of our initial goals was to provide a graph-based visualization of the inter-article links to the user. Because of time constraints and because the category information already provides a good overview about the classification of an article we discarded the idea.

2) *PageRank*: PageRank provides a way to calculate the popularity of a given web-page. On the web, where thousands of pages exist for a given topic, this makes sense as it allows to also take the quality of a page into account when scoring an article.

While we did not implement PageRank ourselves we were able to obtain PageRank data for the English version of Wikipedia. The data showed a large bias towards year-numbers and country-names. This is not unanticipated as links in Wikipedia usually do not link to popular articles but to relevant articles. While we did not formally evaluate result-sets which take the PageRank score into account, we concluded that an implementation of the PageRank algorithm is not likely to provide us with better results.

3) *Bayesian Networks*: The hierarchical structure of Wikipedia resembles the structure of the Bayesian networks. We tried to define a network structure which would allow the relevance propagation. If one article is relevant to the given search query, it might propagate its importance through the network to the strongly related nodes both through the links and shared categories. However, Bayesian networks use the strong assumption of nodes independence. During our experiments we have observed, that this would be broken in Wikipedia, which would degrade the performance and the experience of the user. The problem is with subsections of a single article - they are not independent one from another, even given the article. If one searches for “*Toledo AND Golf*” the article “*European cars of the year*” contains both of the words, but in different subsections (“*Small cars*” and “*Family cars*”), hence the relevance of the whole article cannot be build upon the relevance of subsections (each of them does not contain the other word, which is known to be in the other section) or vice versa.

III. HOW DOES IT ALL FIT TOGETHER

A. Interaction

FIGURE 1 in Appendix C shows the interaction between the parts of the engine. The user’s search query is passed to the Searcher described in II-C using XML-RPC. At the same time the request to the spellchecking service is made. After receiving the structure of the articles and the subsections described in Appendix B, the list of all associated categories is passed to the Category backend, which suggests the clustered coherent categories in reply.

B. Frontend

The frontend with descriptions is shown on the figure 2 in Appendix C. On the left-hand side, right below the search box, the category clusters are shown. Each of the category names is a clickable link to the category description in Wikipedia. The user can select individual categories or the whole cluster to restrict the search.

The main area is dominated by the search results. For every article there is a block with several parts: right under the title, which is a link to the article in Wikipedia, there are the alternative titles (“redirects”), labeled as *Also known as*. These are not links as they do not provide extra documents. The snippet of the article created by Lucene shows the neighborhood of the words in the query. A list of subsections selected as described in II-C.1 is shown, if available. Each of these subsection titles points the user to the relevant subsection inside the article. The Score of the article and of the subsections is visualized to give more information about the relevancy comparison both between different articles and between an article and its subsection.

IV. EVALUATION

IN this section we will discuss how we evaluated our Search Engine, what our results are, and finally we will discuss those results.

A. Method

To test our search engine, we first needed to have a database of queries and their relevant results. To build this database, we selected 50 queries from spylog. We made a distinction here between general queries, like *food*, and specific ones like *american Staffordshire terriers*.

To gather relevant results we ran the queries through three different search engines, all searching the Wikipedia content: Google [2], Wikiii [3] and Wikipedia’s own search engine. We selected only the top ten results of each search engine. Of course, the resulting list of documents were not all relevant, hence we manually needed to mark which ones were.

To answer the question if a document is relevant or not, we asked ourselves if the result under consideration would be what a user is looking for when entering that query. More specifically, this came down to the following set of guidelines:

- Specific results are not relevant for general queries.
- General results are not relevant for specific queries.
- Names are relevant. A user looking for *flowers* might actually be looking *Tommy Flowers*.
- Disambiguation pages are not relevant. A search engine in theory is especially there to prevent the user from needing one.
- Redirects are not relevant. In Wikipedia a couple of titles might link to the same page. Only one of them (the page that is redirected to) is relevant.

Not only the relevant results for each query needed to be assessed. For the category refinement (i.e. restricting the search with relevant categories) we also needed to mark relevant categories for each query in the database. For that we

considered two approaches: one was to strictly mark relevant categories only, the other was to filter out the categories that had absolutely nothing to do with the query, and mark the rest.

Ideally, the first approach should be used but would that be at the expense of recall, the latter is the preferred one. Small scale testing revealed the first approach did not hurt the recall (nor did it improve it).

We consider a category relevant (in the strict sense) if it has something to do with either of the relevant search results. Of course, different search results may belong to entirely different categories.

To get the list of categories, we used the front-end for our search engine with a limit of 60 search results. This number was set this high because the main purpose of the category refinement is to get ‘page two relevant results’ up in the list. Would we show less than 60 results (e.g. 10) the relevant categories for the ‘page two results’ would not have been calculated by the earlier in the report described techniques.

For the actual evaluation, we ran each query with our search engine and compared the top 10 results to our database with relevant results. We then calculated 5 Evaluation Measures:

- (Mean) Recall
- (Mean) Average Precision (MAP)
- (Mean) F1-Score
- (Mean) R-Precision (R-prec)
- p@1 to 10.

B. Results

In the table below the results from the 4 of the Evaluation Measures are listed. We calculated those measures not only for our own search engine (*Team 1B*) and our search engine with category refinement (*Team 1B, refined*), but also for Google, Wikiii and Wikipedia’s own search engine for comparison.

Search Engine	Recall	MAP	F1-score	R-prec
Team 1B	0.461	0.363	0.201	0.347
Team 1B, refined	0.461	0.376	0.219	0.367
Wiki	0.432	0.265	0.192	0.267
Wikiii	0.517	0.376	0.265	0.355
Google	0.726	0.630	0.347	0.627

One evaluation measure is still missing: p@1 to 10. The table below lists the results for the same search engines as above.

n	Team 1B	Refined	Wiki	Wikiii	Google
1	.56	.58	.28	.52	.84
2	.42	.43	.25	.41	.66
3	.32	.32	.21	.36	.56
4	.27	.26	.20	.31	.48
5	.24	.23	.18	.28	.41
6	.21	.20	.17	.26	.37
7	.19	.18	.15	.23	.34
8	.17	.17	.15	.20	.31
9	.16	.15	.15	.18	.29
10	.15	.14	.14	.11	.26

C. Discussion

First, a comment on the results from Google, Wikiii and Wikipedia’s own search engine. Because we used those three search engines to find our relevant results, the precision- and recall values are biased in their favor. We are basically checking if those three search engines are finding their own results. Because of this, their scores are possibly a little higher than they are in reality (there may be relevant results neither of those three engines found).

That said, we can start analyzing our own results. First, let us compare the results from our own search engine with, and without category refinement.

The first thing to notice is that the recall has stayed exactly the same. This tells us two things: firstly the category refinement did not remove any relevant results from the list, and secondly, it did not manage to bring up any relevant results from the second (and lower) result pages (i.e., result 11 and lower).

The fact no relevant results are removed from the list is a desirable result. In practice, category refinement could lead to elimination of relevant results, if there are multiple relevant results, each belonging to different categories. This does not happen here because we marked each category that could be relevant for a certain relevant result, as relevant.

The fact no relevant results from second and lower result pages are brought up, is somewhat unexpected. A possible explanation for it, though, is that there simply are no more relevant results that low in our list we could offer. It would mean our result list is already ordered on relevancy quite well.

More evidence backup up the last assumption can be found when examining the difference in precision-related evaluation measures. The MAP, F1-score and the R-precision all seem to go up, but only by a tiny bit. The fact they go up can be explained by the irrelevant results that are filtered out, making room for relevant results to go up a bit in the top 10 result list, improving precision. The fact the measures only go up a tiny bit can be explained by the earlier assumption that the relevant results already were high up the list.

When looking at the p@n values, we see further evidence of some small-scale movement up the list for relevant results. p@1, p@2 and p@3 show slightly improved scores, but further down the list the scores end up a bit lower (that is, our refined search engine has lower p@4 (and lower) scores than our ‘regular’ search engine). With the added information that the average number of relevant results per query is just above 4, this behavior indeed is an indication that relevant results are moving up the list.

When comparing with other search engines, we first should remark again that the scores for the other search engines are a bit biased. Because of that, our own search engine actually might be somewhat in the disadvantage.

We can therefore quite safely conclude that our search engine is performing better on all accounts than Wikipedia, especially where the precision-measures (MAP, R-precision, p@n) are concerned. We can see the same behavior when comparing to Wikiii: although our search engine is outperformed where recall is concerned, it is quite competitive for the precision-measures. Google appears to be in a league of

its own, indicating most relevant results in our database are found by Google.

V. CONCLUSION

OUR project has shown, that in Wikipedia there exist unique features that might be used to improve the search. Big effort was put in the category handling. Instead of automatic text category classification our team focused on categories already existing in Wikipedia. This allows our engine to automatically reflect changes made in Wikipedia by the editors and to point the searching user to the existing category description in Wikipedia. Even though there is a huge amount of categories with low information value, we managed to filter only the meaningful ones. On top of it we show how to support the engine by an automated definition extraction and how to group the obtained categories into the related groups using WordNet.

We have also implemented simple-looking, yet powerful mechanism of subsection handling. The long articles with subsections relevant to the query are promoted to the top of the results list and the user can straightforwardly navigate right to the point of the interest. Already the layout might give a usefull information “this is a part of ...” or “this belongs to ...”.

The discovered power of information in the redirect pages and their titles is strongly recommended for the future projects to exploit. We believe that it is together with the title boosting responsible for the high precision we achieved.

REFERENCES

- [1] English Wikipedia, <http://en.wikipedia.org>.
- [2] Google, <http://www.google>.
- [3] Wikiii, <http://berk.science.uva.nl:8080/wikiii/result.php>.
- [4] A. Capocci, V.D.P. Servedio, F. Colaiori, L.S. Buriol, D. Donato, S. Leonardi, and G. Caldarelli. Preferential attachment in the growth of social networks: the case of Wikipedia. *submitted to PRE*, 2006.
- [5] R. Grishman. Information extraction: Techniques and challenges. In M.T. Pazzienza, editor, *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*, chapter 2, pages 10–27. Springer, 1997.
- [6] T. Holloway, M. Bozicevic, and K. Börner. Analyzing and visualizing the semantic coverage of wikipedia and its authors. *Submitted to Complexity, Special issue on Understanding Complex Systems*, 2006.
- [7] A. Hotho, S. Staab, and G. Stumme. Wordnet improves text document clustering. In *Proceedings of the Semantic Web Workshop at SIGIR-2003*, Toronto, Canada, August 2003. 6th Annual International ACM SIGIR Conference.
- [8] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, 2003.
- [9] G. Miller, R. Beckwith, C. Fellbaum, D Gross, and K. Miller. Five papers on wordnet. Technical report, Stanford University, 1993.
- [10] I. Muslea. Extraction patterns for information extraction tasks: A survey. In *Proceedings of AAAI Workshop on Machine Learning for Information Extraction*, Orlando, Florida, July 1999.
- [11] H. Saggion and R. Gaizauskas. Mining on-line sources for definition knowledge. In *Proceedings of the 17th International FLAIRS Conference*.
- [12] J. Voss. Measuring Wikipedia. In *Proceedings 10th International Conference of the International Society for Scientometrics and Informetrics*, Stockholm, Sweden, 2005.
- [13] Max Vikel, Markus Krtzsch, Denny Vrandecic, Heiko Haller, and Rudi Studer. Sematic wikipedia. In *Proceedings of the 15th international conference on World Wide Web*, Edinburgh, Scotland, May 2006. WWW 2006.

APPENDIX A: CLUSTER COHERENCE METRICS

Definitions

D : The set of all documents.

C_i : A category/cluster of documents.

$Linking(C_i)$: The set of documents that link to at least one document belonging to C_i .

$Linked(C_i)$: The set of documents that are linked by at least one document belonging to C_i .

$E(d_i, d_j)$: Equals to one if document d_i links to document d_j , zero otherwise.

$l_{IN}(d_i, C_j)$: Equals to the number of distinct links from d_i to documents belonging to C_j .

$bl_{IN}(d_i, C_j)$: The number of *binary links* from document d_i to two or more documents belonging to C_j . This function measures the different binary co-citations of two documents belonging to C_j by d_i . It is equal to the binary combinations of l_{IN} , i.e.

$$bl_{IN}(d_i, C_j) = \frac{l_{IN}(d_i, C_j)!}{(l_{IN}(d_i, C_j) - 2)!} = \frac{l_{IN}(d_i, C_j)!(l_{IN}(d_i, C_j) - 1)!}{2} \quad (1)$$

$l_{OUT}(d_i, C_j)$: Equals to the number of distinct links from documents belonging to C_j , to d_i .

$bl_{OUT}(d_i, C_j)$: The number of *binary links* from two or more documents belonging to C_j , to d_i . This function measures the different binary co-citations of d_i from two documents belonging to C_j . It is equal to the binary combinations of l_{OUT} , i.e.

$$bl_{OUT}(d_i, C_j) = \frac{l_{OUT}(d_i, C_j)!}{(l_{OUT}(d_i, C_j) - 2)!} = \frac{l_{OUT}(d_i, C_j)!(l_{OUT}(d_i, C_j) - 1)!}{2} \quad (2)$$

$l_{INTRA}(d_i \in C_j)$: The number of distinct links from d_i belonging to C_j to documents also belonging to C_j . Every document is considered to also link to itself. In this respect, if a document d_i has one distinct link to another document towards a different document of its category, we have $l(d_i) = 2$.

$bl_{INTRA}(d_i \in C_j)$: The number of *binary links* from document d_i belonging to C_j , to other documents belonging to C_j . The same concept as used in l_{INTRA} is used. Again, it is equal to the binary combinations of l_{INTRA} , i.e.

$$bl_{INTRA}(d_i \in C_j) = \frac{l_{INTRA}(d_i)!}{(l_{INTRA}(d_i) - 2)!} = \frac{l_{INTRA}(d_i)!(l_{INTRA}(d_i) - 1)!}{2} \quad (3)$$

Inward co-citation metric

This metric uses the number of binary co-citations of documents linking to documents belonging to C_j . It is then normalized so that it goes from zero to one. Zero means no documents link two or more documents belonging to C_j , one means that all documents linking to documents belonging to C_j actually link to all of them.

$$InwardMetric(C_j) = \frac{\sum_{d_i \in Linking(C_j)} bl_{IN}(d_i, C_j)}{\frac{|C_j|!}{(|C_j| - 2)!} |Linking(C_j)|} \quad (4)$$

Outward co-citation metric

This time the structure of the outward links from C_j is used. Conceptually, we invert the direction of the links and transform this problem back to the previous metric. In other words, we use the binary combination of inverted links from documents being linked by documents belonging in C_j . The same as before apply considering the normalization. That is, we get zero if no two documents belonging in C_j link the same document, one if all linked documents are actually linked by all documents belonging to C_j .

$$OutwardMetric(C_j) = \frac{\sum_{d_i \in Linked(C_j)} bl_{OUT}(d_i, C_j)}{\frac{|C_j|!}{(|C_j| - 2)!} |Linked(C_j)|} \quad (5)$$

Intra category co-citation metric

Now, we use the link structure between the documents of C_j . We use the number of binary combination of links from each document of C_j to other documents of C_j , considering that each document also links to itself. It ranges from zero, when no document links to any other document in C_j , to one, when all documents link to all other documents in C_j .

$$IntraMetric(C_j) = \frac{\sum_{d_i \in C_j} bl_{INTRA}(d_i)}{\frac{|C_j|!}{(|C_j|-2)! |C_j|}} \quad (6)$$

VI. APPENDIX B

A. XML-RPC interface specification

THE XML-RPC interface is described by using Python syntax. The following types of elements are used:

- [] ... is used to indicate a XML-RPC array
- {} ... is used to indicate a XML-RPC struct

The root element in the result-set is build upon the following structure:

```
[{ 'document': DOCUMENT,
  'score': SCORE,
  'sections': [SECTION],
  'snippet': SNIPPET}]
```

Syntax of a DOCUMENT:

```
{ 'categories': [CATEGORYNAME],
  'title': TITLE1,
  'title -2': TITLE2,
  'title -3': TITLE3}
```

Syntax of a SECTION:

```
{ 'score': SECTIONSCORE,
  'title': SECTIONTITLE}
```

The result-set consists of a list of XML-RPC structs. Each struct has the following attributes:

- document ... a struct containing a document
- score ... the score of the document for a given query
- sections ... list of structs describing the sections of a document
- snippet ... snippet of text with matching keywords highlighted

A document has the following attributes:

- categories ... list of strings which contain the names of categories to which the article belongs
- title ... title of the document
- title-N ... optional titles of document

A section has the following attributes:

- score ... score of the section for a given query
- title ... title of the section

APPENDIX C

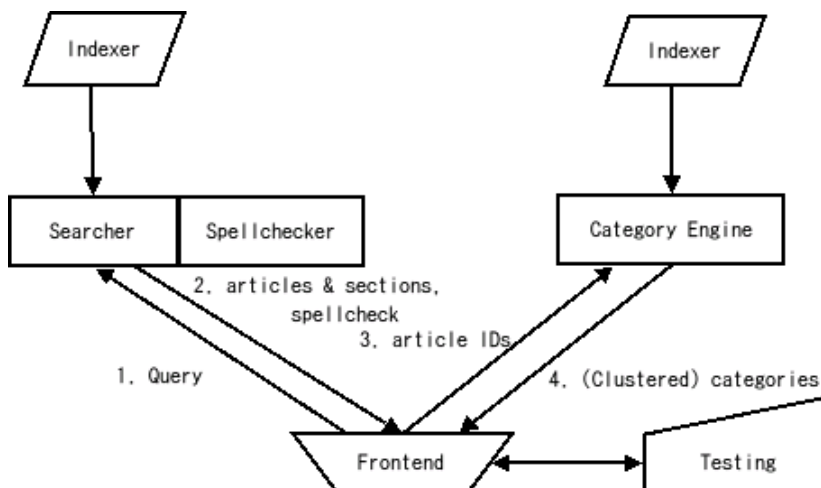


Fig. 1. Schema of the interactions in the engine

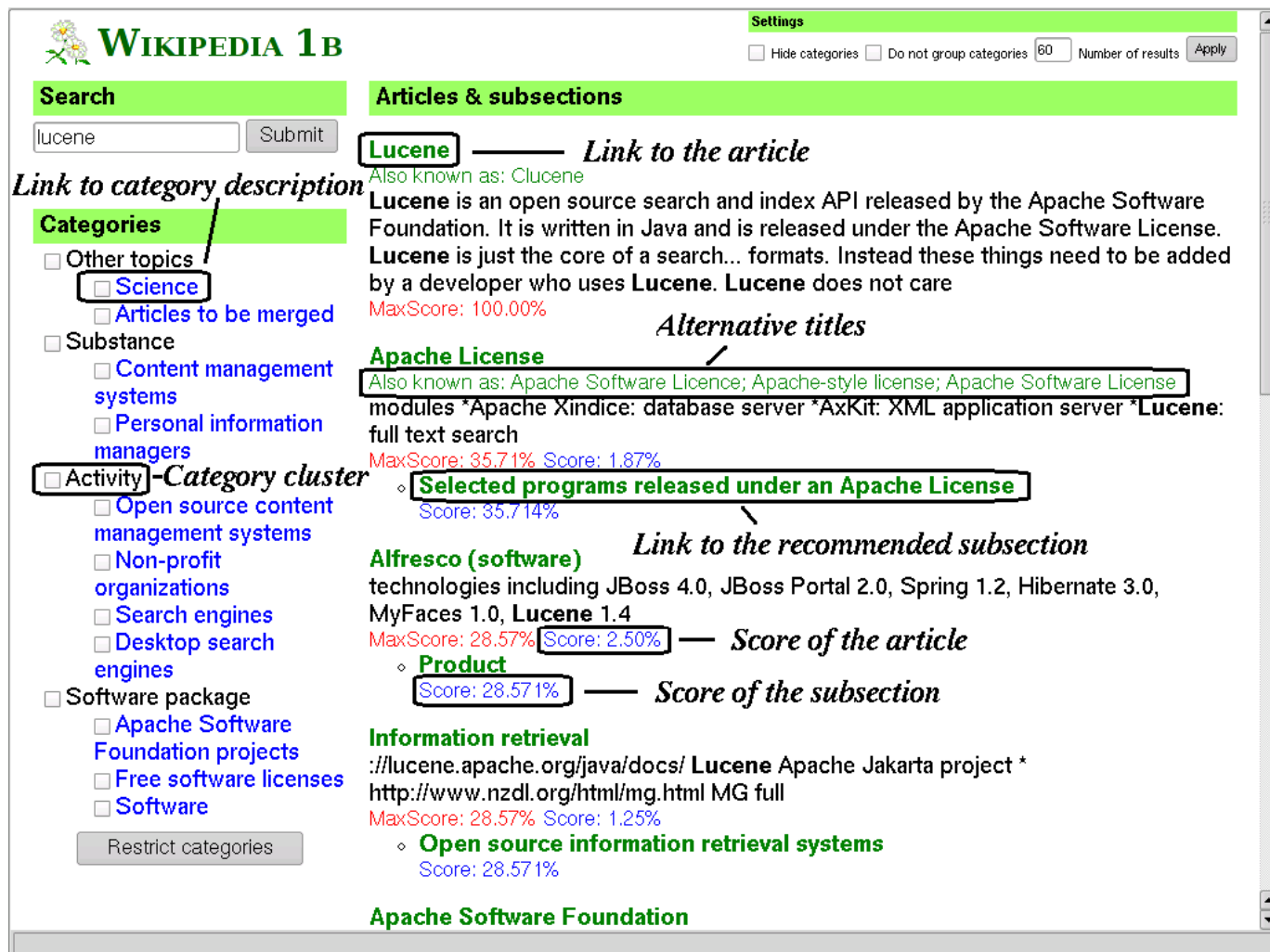


Fig. 2. Screenshot of the frontend page