From Keywords to Keyqueries: Content Descriptors for the Web

Tim Gollub Matthias Hagen Maximilian Michel Benno Stein

Bauhaus-Universität Weimar
99421 Weimar, Germany
<first name>.<last name>@uni-weimar.de

ABSTRACT

We introduce the concept of keyqueries as dynamic content descriptors for documents. Keyqueries are defined implicitly by the index and the retrieval model of a reference search engine: keyqueries for a document are the minimal queries that return the document in the top result ranks. Besides applications in the fields of information retrieval and data mining, keyqueries have the potential to form the basis of a dynamic classification system for future digital libraries—the modern version of keywords for content description.

To determine the keyqueries for a document, we present an exhaustive search algorithm along with effective pruning strategies. For applications where a small number of diverse keyqueries is sufficient, two tailored search strategies are proposed. Our experiments emphasize the role of the reference search engine and show the potential of keyqueries as innovative document descriptors for large, fast evolving bodies of digital content such as the web.

Categories and Subject Descriptors: H.3.3 [Information Search and Retrieval]: Retrieval Models, Query Formulation

General Terms: Algorithms, Experimentation, Performance

Keywords: keyquery, content description, automatic query formulation, exhaustive search, search strategies

1. INTRODUCTION

A content descriptor is a word or a short phrase that expresses the central topical aspect or the domain of a document. Typical examples can be found in the meta section below this paper's abstract: the categories, general terms, and keywords integrate the document into the ACM classification system.

We propose an additional modern means of document content description: search queries. The underlying idea is that those queries that return a given document in their top ranks for some reference search engine "describe" the document's content well enough to stand out from the indexed collection. If the query is maximally general (i.e., no subset of the query has the same property), we call it a *keyquery* for the document. The validity check of a query's suitability as a document's content descriptor is straightforward: submission to the reference search engine.

SIGIR'13, July 28–August 1, 2013, Dublin, Ireland.

In the context of digital libraries, key*queries* constitute an interesting alternative to key*words*, which typically are manually chosen free form content descriptors. The advantage of keyqueries is that their descriptiveness can be automatically tested using the library's search engine. Whenever a candidate is not descriptive enough, additional terms can be suggested or automatically added. Carried out consequently, the keyquery approach includes validity checks for the whole library on every new insertion of a document, and automatic re-establishment when necessary. This way, keyqueries can form the basis of a dynamic and automatically maintained classification system for digital libraries. Especially for fast evolving large-scale bodies of digital content, where manually maintained classification systems need enormous manpower to stay up to date,[1] dynamic classification with keyqueries is an effective alternative.

In information retrieval systems, keyqueries can serve as query recommendations to find related resources for a specified document, or to implement query expansion and relevance feedback techniques. For data mining tasks such as clustering, a "bag of queries" document representation can be based on the keyqueries of the dataset, and the relevance scores of the search engine can be used as the feature weights. As the concept of search queries nowadays is well understood by web users, keyqueries might even be used as cluster labels—especially when clustering search results.

In the following section, we review literature related to the concept of keyqueries and give pointers to the state of the art in various potential applications. Section 3 provides algorithms for finding the keyqueries of a document and explores options for pruning the search space. In Section 4, we report on our experiments with keyqueries of scientific papers using different reference search engines. Our findings reveal that the concept of keyqueries is sound and effectively applicable. We conclude with an outlook on future work.

2. RELATED WORK

The state-of-the-art technique for the automated generation of content descriptors is keyword or keyphrase extraction. Actually, we chose the term *keyquery* in dependence on these two concepts. A survey of current research in the field is given in the overview paper of the 2010 SemEval competition on keyphrase extraction [9]. In particular, and as will be discussed in detail in Section 3, we use keyword extraction in a subroutine to efficiently find a small subset of diverse keyqueries. For the respective experiments presented in Section 4, we employ the TextRank algorithm [10]. TextRank is an unsupervised keyword and keyphrase extraction technique that represents a document as a graph and determines a keyword ranking by applying the PageRank algorithm.

[1] A famous manually maintained, large-scale classification system is the Yahoo! directory, which went offline in Europe recently.

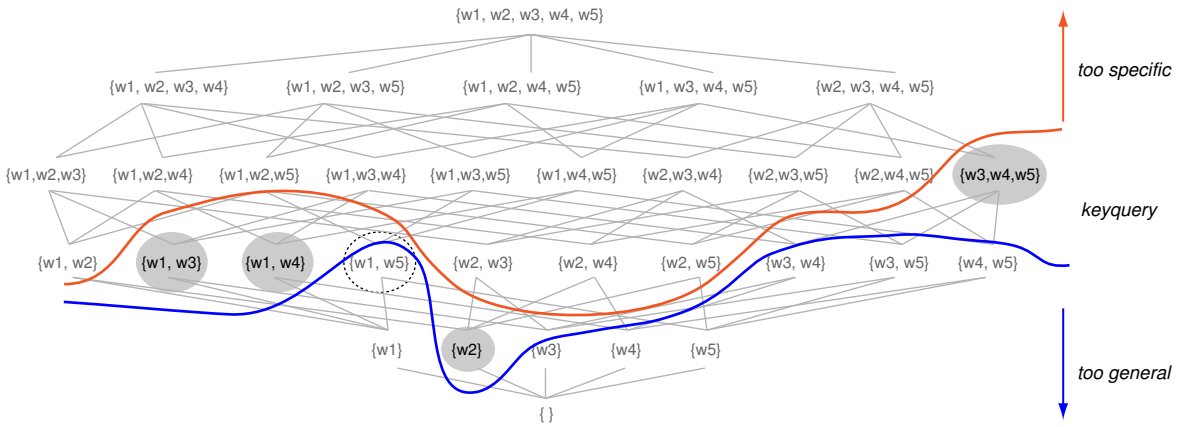

Figure 1: The search space $\mathcal{Q}$ for a five-word vocabulary, divided into the three subspaces *too generic*, *keyqueries* $\mathcal{Q}^*$, and *too specific*.

A dynamic classification system based on keyqueries requires an efficient means to store and update the keyqueries. An adequate data structure has been proposed by Pickens et al. in 2010: the reverted index [11]. As its name suggests, the reverted index is related to the inverted index used in search engines, but it stores queries in document postlists—instead of documents in term postlists. In their experiments on query recommendation and relevance feedback, Pickens et al. populate the reverted index with unigram queries and suggest the usage of n-gram queries or queries from query logs for further performance improvements. Keyqueries now contribute a third sophisticated alternative with a sound theoretical justification.

Once keyqueries are stored in a reverted index, the next step towards a dynamic classification system is to derive a hierarchical structuring of the queries. Bonchi et al. introduce this task as topical query decomposition [3], since the queries of a common class should represent "coherent, conceptually well-separated topics." In their work, Bonchi et al. achieve good results by using a set cover algorithm with red-blue metric for the problem.

Also in the field of data mining, inspiring related work exists. In their optimum clustering framework (OCF) [4], Fuhr et al. suggest to represent documents as "bags of queries" and to use the relevance scores or retrieval ranks for each query as the feature weights. Keyqueries fit very naturally into the proposed scheme. That they might be even more appropriate than arbitrary queries in an OCF-style clustering approach can be motivated from a finding of Azzopardi and Vinay: in their analysis of document retrievability with search engines, they observed that simple bag-of-words representations with $tf \cdot idf$-based weights are biased towards a small number of documents [2]. These documents appear in the top results for significantly more of the basic index terms than other documents. In the context of the optimum clustering framework, a set of arbitrary queries would thus come with a bias, obviously harming the overall effectiveness. Instead, a comparably unbiased feature set can be formed by including the same number of keyqueries from each document. The documents' retrievabilities then are approximately equal and the feature weights' distribution is rather unbiased.

3. COMPUTING KEYQUERIES

In this section, we formalize the keyquery concept and present an exhaustive search algorithm to find all keyqueries for a given document. We also explore possibilities to reduce the search space and present two heuristic search strategies that are tailored to scenarios where a few diverse keyqueries suffice.

Keyqueries.

Given the vocabulary $W_d = \{w_1, w_2, \ldots, w_n\}$ of a document d, let $\mathcal{Q}_d$ denote the family of search queries that can be formulated from W_d without word repetitions; i.e., $\mathcal{Q}_d$ is the power set of W_d, $\mathcal{Q}_d = 2^{W_d}$. Note that no distinction is made with respect to the ordering of the words in a query. If it is clear from the context, we omit the subscripts and just use W and $\mathcal{Q}$ to denote the vocabulary and the potential queries from d.

A query $q \in \mathcal{Q}$ is a *keyquery* for d with respect to a reference search engine S iff: (1) d is among the top-k results returned by S on q, and (2) no subset $q' \subset q$ returns d in its top-k results when submitted to S. The parameter k controls the level of keyquery generality and is usually set to some small integer, such as 10 in our case. Let $\mathcal{Q}^*$ denote the set of keyqueries for d.

Figure 1 shows a visualization of $\mathcal{Q}$ and $\mathcal{Q}^*$, which we adapted from work done by Hagen and Stein on query formulation [6]. The keyquery search space $\mathcal{Q}$ divides into three subspaces. The subspace of too-general-queries that do not retrieve the desired document in the top-k results, the subspace $\mathcal{Q}^*$ of keyqueries, and the subspace of too-specific-queries all of which are supersets of a shorter keyquery. An interesting query in the example is $\{w1, w5\}$ (third row): it is still too general but cannot be extended without becoming too specific.

Exhaustive Search.

Note that an exhaustive search in $\mathcal{Q}$ is required in order to find all keyqueries for a given document. Previous work in the field of query formulation used an adapted version of the Apriori algorithm [1] to identify queries not returning too many results [7]. The Apriori algorithm stems from the field of frequent itemset mining and is considered one of the top ten data mining algorithms [12]. It is more efficient than enumerating $\mathcal{Q}$ in an arbitrary level-wise order. As frequent itemset mining is very similar to our keyquery setting we also employ a tailored Apriori variant.

The pseudo-code listing of Apriori for keyquery identification is given in Algorithm 1. The algorithm starts with submitting all one-word queries to the reference search engine (line 1) and evaluates whether the queries are either keyqueries or too general (lines 2–3). Note that it is assumed that the search engine always retrieves the document at some rank in the result list since all query words are present in the document. If the query combining all the candidate terms in C_1 is still too general (lines 4–5), all shorter queries have to be too general as well and the algorithm can immediately terminate returning $\mathcal{Q}^*$. The main loop of the algorithm (lines 7–15)

Algorithm 1 The Apriori algorithm for keyquery search

Input: document d with vocabulary W
Output: the family $\mathcal{Q}^*$ of keyqueries

```
1: for all w ∈ W do submit(w)
2: Q* ← {w : w ∈ W and w is keyquery}
3: C_1 ← {w : w ∈ W and w too general}
4: submit(⋃_{w∈C_1} w)
5: if ⋃_{w∈C_1} w too general then stop and output Q*
6: i ← 1
7: while C_i ≠ ∅ do
8:     for all q, q' ∈ C_i do
9:         if |q ∩ q'| = i − 1 then q_cand ← q ∪ q'
10:        if (q_cand \ w) ∈ C_i for all w ∈ q_cand then
11:            if q_cand ∉ C_{i+1} and q_cand ∉ Q* then
12:                submit(q_cand)
13:                if q_cand too general then C_{i+1} ← C_{i+1} ∪ q_cand
14:                if q_cand is keyquery then Q* ← Q* ∪ q_cand
15:    i ← i + 1
16: output Q*
```

iterates through the search space in a level-wise manner, i.e., all two-word keyquery candidates first, then the three-word keyquery candidates, etc. During this process, the search space is pruned whenever possible: all queries which are supersets of identified keyqueries can be omitted. The algorithm returns $\mathcal{Q}^*$ if no new candidate query q_{cand} can be formulated.

Search Space Reduction.

Since the search space $\mathcal{Q}$ grows exponentially in the size of the vocabulary, an effective strategy to prune $\mathcal{Q}$ is to reduce the vocabulary ahead of a search. We call strategies that reduce the vocabulary *vertical pruning strategies*, whereas *horizontal pruning strategies* constrain the maximum length of the keyqueries to be found. In the following, we explore pruning strategies of both kinds.

To understand the basic characteristics of contemporary content descriptors, we analyze the 2012 version of the ACM Computing Classification System.[2] Our review reveals that only three parts-of-speech types are typically used for content descriptors: nouns and noun combinations (e.g., *web search*), adjectives (e.g., *personalized* search), and conjunctions (e.g., retrieval models *and* ranking). An appropriate vertical pruning strategy can thus ignore all words that do not belong to any of these three classes. Furthermore, adjectives are only considered when they appear in combination with a noun. For the documents used in our experiments (cf. Section 4), this vertical pruning results in a search space reduction to about $\sqrt{2^n}$, where n is the original vocabulary size.

We also infer a horizontal pruning strategy from the length of the ACM content descriptors. The average descriptor length is about three words, and up to eight words in extreme cases. Hence, we suggest to not consider longer queries.

Heuristic Search Strategies.

For certain applications such as query recommendation, it is often not necessary to compute all keyqueries of a document; two or three keyqueries with no or only a small term overlap could suffice. In such scenarios, heuristic search strategies can simply proceed in a depth-first manner. For this purpose, the TextRank algorithm [10] provides an interesting basis. TextRank ranks each word of a document by applying the PageRank procedure to a document's graph model: nodes represent words and edges connect words that occur next to each other in the text. From the TextRank scores for words and the graph representation, we derive two heuristic search strategies. Note that the vertical pruning described above (vocabulary reduction) is always applied.

Rank-Driven Search. An initially empty query is successively expanded by that word from $\{w : w \in W, w \notin q\}$ with the highest score until the document d is returned among the top-k results. As the derived query q might be too specific, it may need to be minimized. A straightforward solution is to successively remove each word from q and to test whether d is still returned in the top-k (if not, keep the word and try the next one). The reduced q then is guaranteed to be a keyquery. To find another keyquery with different terms, start again with $W = W \setminus \{q^*\}$ until no keyquery can be found or sufficiently many have been returned.

Rank-driven search can also start from keyphrases instead of words. This can be especially advisable for reference search engines using proximity features or even allowing phrasal search—good query segmentation then can help a lot [5].

Graph-Driven Search. A second heuristic search strategy can be based on the TextRank graph. The first term added to the initially empty query q again is the highest scoring word $w \in W$. But then q is extended by adding from all words *adjacent* to w in the TextRank graph the word w^* with the highest score. If no adjacent word exists, the word with the highest score from $\{w : w \in W, w \notin q\}$ is used, similar to rank-driven search. The extension of w^* works analogously until a search with q has the document d among the top-k results. Again, q is minimized as above by trying to remove keywords, and the process restarts with $W = W \setminus \{q^*\}$ until either enough keyqueries have been found or no more are possible.

Graph-driven search obviously favors phrases in the keyquery generation and should work especially well for search engines that include proximity features and allow phrasal search. However, graph-driven search is also applicable with basic engines, then probably losing much of its potential.

4. EMPIRICAL ANALYSIS

Our empirical analyses are conducted in the setting of collections of scientific papers and focus on the following two questions.

1. How many queries have to be submitted to determine one keyquery for a paper against a contemporary search engine?
2. Does citation count influence the number of submitted queries or the keyqueries' lengths?

The first query addresses the efficiency or "costs" of keyquery computation in general (typically query submissions require non-negligible amounts of time) while the second query aims at evaluating a potential factor influencing efficiency (Google often shows highly cited papers in the top ranks).

Our experimental study is conducted on all the papers published at the SIGIR and CIKM conferences in the years 1999–2012. To address our first question, we sample the document set SIGrandom containing 50 random papers published at SIGIR. To address our second question, we use the document set SIGcited of the 50 SIGIR papers having the highest citation numbers in the ACM Digital Library (DL). The three reference search engines are Google as representative of current web search and two variants of a local Lucene indexing the 3796 papers (phrasal search enabled): with and without a logarithmic citation boost on the actual relevance score (i.e., the score is multiplied with the $\log_2$ of the paper's citation count obtained from the ACM DL). The granularity parameter

[2] http://dl.acm.org/ccs.cfm

Table 1: Keyquery statistics for two document sets and three reference search engines.

Search engine	Document set	Query submissions	Success ratio	Keyquery length	Retrieval rank	Result list length
Google	SIGrandom	18.8	60%	7.3	3.1	3.2 million
	SIGcited	10.6	54%	5.7	4.2	3.5 million
Lucene (citation boost)	SIGrandom	4.4	100%	3.2	4.1	89
	SIGcited	14.4	100%	3.6	4.9	161
Lucene (no boost)	SIGrandom	5.5	100%	3.4	4.0	88
	SIGcited	3.4	100%	2.8	3.5	234

is set to $k = 10$. Since Google imposes usage restrictions, we constrain the number of queries that can be submitted per document to 128 for all the engines. This would allow exhaustive search with seven words but in the experiments presented here, we focus on the graph-driven heuristic and evaluate finding one keyquery per document. The results of our experiments are shown in Table 1. All values are averaged over the document sets.

Using Google, a keyquery is found for about 54–60% of the documents in the collections (success ratio). This does not mean that the documents cannot be found using Google, but that within our restricted budget of 128 query submissions no keyquery could be identified. As expected, the average number of required queries in case of success is lower for highly cited papers than for random ones (10.6 vs. 18.8). Keyqueries for highly cited papers are also shorter (5.7 vs. 7.3). The documents themselves are returned higher in the ranking for random papers which is not that surprising as the longer keyqueries are more specific; the total result list lengths are of comparable magnitude (3.5 million). Given the size of Google's search index, the observed keyquery length is small; it is comparable to manually created content descriptors in the ACM DL. However, the success ratio of only 54–60% supports Azzopardi and Vinay's finding that retrievability of documents often is an issue [2].

The low success ratio using Google cannot be further explored due to the black-box characteristic of the search process. It is thus interesting to compare the findings with the two Lucene instances that we can control completely. As expected, the small size of the index yields a perfect success ratio with Lucene and also enables more efficient construction (less query submissions) and shorter keyqueries. Not surprisingly, the number of results of a keyquery is much lower for Lucene than for Google.

Comparing the two document sets on the two Lucene instances yields an interesting observation. Without citation boost, it is much more difficult to find keyqueries for the highly cited papers than for a random one (14.42 vs. 4.44 queries). A possible explanation is that the highly cited papers are part of large research branches with many papers on similar topics—the highly cited papers probably being the most influential ones. Without boosting highly cited papers, it is much more difficult to retrieve them from the rest. Random papers on the other hand often do not have that many other papers on similar topics such that keyqueries are easier to find. The average keyquery result list length also supports this explanation as the result lists are longer for SIGcited keyqueries (161.34 vs. 88.56). When the citation boost is included, the SIGcited papers are much more easy to find (11 queries less than without boost) and ranked higher in even longer keyquery result lists. The random papers instead require a little more effort than before but their keyquery characteristics remain comparable.

5. SUMMARY

With the concept of keyqueries we introduce a dynamic means to address the fast evolving bodies of digital content in our society. The range of potential applications in information retrieval, data mining, or digital library tasks underline the innovation and relevance of our idea: relying on the acceptance, the agreed semantics, and the approved indexing of keyword-based search engines, the state-of-the-art search technologies become a viable replacement for manually constructed content descriptors. Our experiments show that keyqueries can be effectivly constructed for the majority of the analyzed documents when using a graph-driven search, and that the role of the reference search engine as context provider can be exploited to favor content with specific properties.

As for future work, it would be very interesting to study basic characteristics of keyqueries for larger document corpora and to further investigate the retrievability issues we observed for our restricted budget with Google. In the digital library setting also the actual acceptance of keyqueries with human users could be evaluated. Other interesting research directions are the potential applications of keyqueries as suggestions to find related documents or as cluster labels in search scenarios.

Last but not least, note that the shortest possible keyquery for our paper against the SIGIR-CIKM test corpus used in our experiments is very simple: keyquery.

6. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB 1994*, pp. 487–499.

[2] L. Azzopardi and V. Vinay. Retrievability: An evaluation measure for higher order information access tasks. In *CIKM 2008*, pp. 561–570.

[3] F. Bonchi, C. Castillo, D. Donato, A. Gionis. Topical query decomposition. In *KDD 2008*, pp. 52–60.

[4] N. Fuhr, M. Lechtenfeld, B. Stein, T. Gollub. The optimum clustering framework: Implementing the cluster hypothesis. *Information Retrieval*, 15(2):93–115.

[5] M. Hagen, M. Potthast, A. Beyer, B. Stein. Towards optimum query segmentation: In doubt without. In *CIKM 2012*, pp. 1015–1024.

[6] M. Hagen and B. Stein. Capacity-constrained query formulation. In *ECDL 2010*, pp. 384–388.

[7] M. Hagen and B. Stein. Candidate document retrieval for web-scale text reuse detection. In *SPIRE 2011*, pp. 356–367.

[8] E. Hatcher and O. Gospodnetic. *Lucene in Action*. Manning Publications Co., 2004.

[9] S. N. Kim, O. Medelyan, M.-Y. Kan, T. Baldwin. SemEval-2010 task 5: Automatic keyphrase extraction from scientific articles. In *SemEval 2010*.

[10] R. Mihalcea and P. Tarau. TextRank: Bringing order into texts. In *EMNLP 2004*, pp. 404–411.

[11] J. Pickens, M. Cooper, G. Golovchinsky. Reverted indexing for feedback and expansion. In *CIKM 2010*, pp. 1049–1058.

[12] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, Dec. 2007.